EXERCICES SUPPLEMENTAIRES 3GIG

INFORMATIQUE

Bourone Jean-Marie, Dziadek Pierre

Version : 15 septembre 2022

TABLE DES MATIERES

AUTEURS

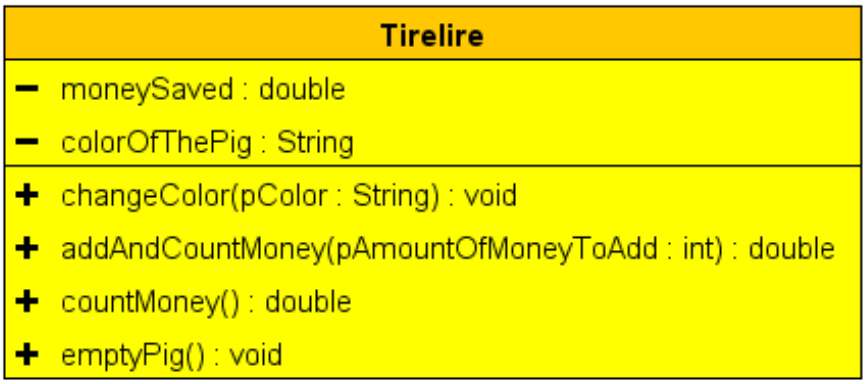
ATERT-LYCÉE REDANGE	Atert Lycée Réiden (ALR)	**Dziadek Pierre**	pierre.dziadek@education.lu
ATERT-LYCÉE REDANGE	Atert Lycée Réiden (ALR)	**Bourone Jean-Marie**	jean-marie.bourone @education.lu

A) EXERCICES THEORIQUES

A.1) NOMMEZ LES DIFFERENTES PARTIES

Attribuez les termes informatiques suivants aux parties correspondantes de l'UML et du code source java sur la page suivante :

1. *Classe*
2. *Instance*
3. *Attribut*
4. *Type de l'attribut*
5. *Type de la méthode*
6. *Initialisation*
7. *Méthode*
8. *Instanciation*
9. *Objet*
10. *Retour d'un résultat*
11. *Paramètre*



```java
/**
 * Write a description of class "Tirelire" here.
 *
 * @author      11TG
 */

// Ma classe tirelire
public class Tirelire
{
      private double moneySaved    = 1.5;
      private String colorOfThePig = "rouge";

      public void changeColor(String pColor)
      {
            colorOfThePig = pColor;
      }

      public double addAndCountMoney(int pAmountOfMoneyToAdd)
      {
            moneySaved = moneySaved + pAmountOfMoneyToAdd;
            return moneySaved;
      }

      public double countMoney()
      {
            return moneySaved;
      }

      public void emptyPig()
      {
            moneySaved = 0;
      }
}
```

A.2) Interpretez le code Java 1

Interprétez le code source Java de cette méthode.

```
private double number = 100;

public void methode1()
{
    number = number * 1.50;
}
```

Utilisez pour cela la place vide ci-dessous et répondez aux questions suivantes :

- Quel est le rôle de la méthode ?
- Si on appelle la méthode deux fois, quelle devient la valeur de ***number*** ?

A.3) INTERPRETEZ LE CODE JAVA 2

Interprétez le code source Java de cette méthode. Veuillez remplir à cet effet les tableaux de variables ci-dessous.

```
public void methode2(int pNumb1, int pNumb2)
{
      int temp = 0;
      temp   = pNumb1;
      pNumb1 = pNumb2;
      pNumb2 = temp;
}
```

Remplissez les trois tableaux de variables

Remplacez les paramètres ***pNumb1*** et ***pNumb2*** par les valeurs indiquées au-dessus des tableaux.

pNumb1 = 5 , pNumb2 = 15

temp	pNumb1	pNumb2

pNumb1 = 6 , pNumb2 = 8

temp	pNumb1	pNumb2

pNumb1 = -5, pNumb2 = 6

temp	pNumb1	pNumb2

Expliquez le rôle de la méthode

A.4) INTERPRETEZ LE CODE JAVA 3

Interprétez le code source Java de cette méthode. Veuillez remplir à cet effet les tableaux de variables ci-dessous.

```java
public int methode3(int pNbr)
{
    int myNumber;
    myNumber = pNbr;
    myNumber = myNumber + myNumber;
    myNumber = myNumber + 10;
    myNumber = myNumber / 2;
    myNumber = myNumber - pNbr;
    return myNumber;
}
```

Remplissez les tableaux de variables

Remplacez le paramètre ***pNbr*** par la valeur indiquée au-dessus de chaque tableau.

pNbr = 1

myNumber	pNbr

Retour : _____

pNbr = 5

myNumber	pNbr

Retour : _____

pNbr = 9

myNumber	pNbr

Retour : _____

Expliquez le rôle de la méthode

A.5) ENCERCLEZ LES ERREURS ET CORRIGEZ-LES A COTE DU CODE SOURCE

Il y a 15 fautes dans le code source en dessous, corrigez celles-ci. Notez qu'on ne considère ici que les fautes reconnues par le compilateur (les erreurs de convention des noms, d'indentation, etc, ne comptent pas comme des fautes).

```
/**
 * @author      dzipi & klegi
/*
public Note class
{
     String subject = Informatique;
     String mark    = 50;

     public String getSubject();
     {
          return subject;
          subject = "Informatique";
     }

     private void setSubject(string pSubject)
     {
          pSubject = subject;
     }

     public int setMark(int pMark)
     {
          markInInformatics = pMark;
     }

     public int getMark()
     }
          return mark
     {

     public void  adjustMark(int Ppoints)
     {
          mark = mark + pPoints;
     }
```

Solution

A.6) CORRIGEZ LES ERREURS DE L'UML

Corrigez les fautes dans le diagramme UML en-dessous. Redessinez ainsi le diagramme UML correct.

+ getSubject : String + setSubject(pSubject : String) : void + setMark(pMark :int) : void + getMark() : double + adjustmark(pPoints : int) : void
+ String: subject + Int: mark = 0
Note

A.7) TABLEAU DE VARIABLES 1

Voici le code source d'une méthode, qui calcule et retourne la somme des deux nombres entiers ***pA*** et ***pB***.

```
1)      public int calculateSum(int pA, int pB)
2)      {
3)              int sum=0;
4)              sum = pA + pB;
5)              return sum;
6)      }
```

Faites un tableau de variables pour ***pA*** = 2 et ***pB*** = 2.

Valeur retournée :		

Faites un tableau de variables pour ***pA*** = 6 et ***pB*** = 4.

Valeur retournée :		

A.8) TABLEAU DE VARIABLES 2

Voici le code source d'une méthode, qui réalise des calculs et retourne un nombre comme résultat.

```
1)      public int calculate (int pA, int pB, int pC)
2)      {
3)              int res=0;
4)              res = res + pB;
5)              sum = res - pA;
6)              sum = res * pC;
7)              return res;
8)      }
```

Faites un tableau de variables pour ***pA*** = 3, ***pB*** = 5 et ***pC*** = 2.

Valeur retournée :			

F Faites un tableau de variables pour ***pA*** = 6, ***pB*** = 2 et ***pC*** = 3.

Valeur retournée :			

A.9) TABLEAU DE VARIABLES 3

Voici le code source d'une méthode, qui calcule la somme des nombres pairs entre deux entiers ***pA*** inclus et ***pB*** non inclus. Par exemple, la somme des nombres pairs entre 1 et 6 est 2 + 4 = 6.

```
1)      public int calculateSumEven(int pA , int pB)
2)      {
3)              int sum=0;
4)              for (int i=pA; i < pB; i++)
5)              {
6)                      if (i % 2 == 0)
7)                      {
8)                              sum = sum + i;
9)                      }
10)             }
11)             return sum;
12)     }
```

Faites un tableau de variables pour ***pA*** = 1 et ***pB*** = 4.

Valeur retournée :					

A.10) TABLEAU DE VARIABLES 4

Voici le code source d'une méthode, qui calcule et retourne la factorielle de n.

```
1)      public double calculateFactorial()
2)      {
3)              double product=1;
4)              int i=1;
5)              while ( i<=n )
6)              {
7)                      product=product*i;
8)                      i++;
9)              }
11)             return product ;
12)     }
```

Faites un tableau de variables pour ***n*** = 3.

Valeur retournée :			

Faites un tableau de variables pour ***n*** = -5.

Valeur retournée :			

A.11) TABLEAU DE VARIABLES 5

```
1)      public double calculatePower()
2)      {
3)              double power=1;
4)              for(int i = 1 ; i <= b; i++)
5)              {
6)                      power = power * a;
7)              }
8)              return power;
11)     }
```

Faites un tableau de variables pour ***a*** = 2 et ***b*** = 4.

Valeur retournée :				

aites un tableau de variables pour ***a*** = 4 et ***b*** = 3.

Valeur retournée :				

A.12) TABLEAU DE VARIABLES 6

```
1)      public int GCD (int pA, in pB)
2)      {
3)              int gcd = 1;
4)              int min = Math.min(a,b);
5)              for(int i = 1 ; i <= min ; i++)
6)              {
7)                      if(a % i == 0 && b % i == 0)
8)                      {
9)                              gcd = i;
10)                     }
11)             }
12)             return gcd;
13)     }
```

Faites un tableau de variables pour ***a*** = 4 et ***b*** = 12.

Valeur retournée :						

Faites un tableau de variables pour ***a*** = 3 et ***b*** = 9.

Valeur retournée :						

A.13) TABLEAU DE VARIABLES 7

```
1)      public int GCD_OptimizedEuclid(int pA, in pB)
2)      {
3)              int help = 0;
4)              while (pB != 0)
5)              {
6)                      help = pA % pB;
7)                      pA = pB;
8)                      pB = help;
9)              }
10)             return pA;
11)     }
```

Faites un tableau de variables pour ***pA*** = 240 et ***pB*** = 64.

Valeur retournée :			

A.14) TABLEAU DE VARIABLES 8

Voici le code d'une méthode.

```
1)      public int LCM_search(int pA, int pB)
2)      {
3)              int result = pB
4)              while (result % pA != 0)
5)              {
6)                      result = result + pB;
7)              }
8)              return result;
9)      }
```

Remplissez le tableau de variables pour les valeurs ***pA*** = 7 et ***pB*** = 25.

Aide : Le résultat attendu est 175.

Valeur retournée :			

A.15) TABLEAU DE VARIABLES 9 (EXERCICE 26 – INTEGERNUMBER)

```
private int n;

1)      public boolean isPrime()
2)      {
3)              int count = 0;
4)              for (int i=1 ; i<=n ; i++)
5)              {
6)                      if (n % i == 0)
7)                      {
8)                              count++;
9)                      }
10)             }
11)             return count == 2;
```

Remplissez le tableau de variables pour les valeurs ***n*** = 4

Valeur retournée :				

Remplissez le tableau de variables pour les valeurs ***n*** = 5

Valeur retournée :				

A.16) TABLEAU DE VARIABLES 10 (EXERCICE 26 – INTEGERNUMBER)

```
private int n;

1)      public int sumOfDividers()
2)      {
3)              int sum = 0;
4)              for (int i=1 ; i<=n ; i++)
5)              {
6)                      if (n % i == 0)
7)                      {
8)                              sum = sum + i;
9)                      }
10)             }
11)             return sum;
```

Remplissez le tableau de variables pour les valeurs ***n*** = 3

Valeur retournée :				

Remplissez le tableau de variables pour les valeurs ***n*** =4

Valeur retournée :				

A.17) TABLEAU DE VARIABLES 11 (EXERCICE 26 – INTEGERNUMBER)

```
private int n;

1)      public int sumOfDividers()
2)      {
3)              int reverse = 0;
4)              int lastDigit = 0;
5)              while (n!=0)
6)              {
7)                      lastDigit = n % 10;
8)                      reverse = reverse * 10 + lastDigit;
9)                      n = n / 10;
10)             }
11)             n = reverse;
12)             return n;
13)     }
```

Remplissez le tableau de variables pour les valeurs ***n*** = 567

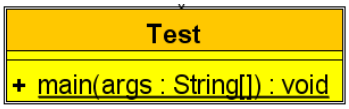
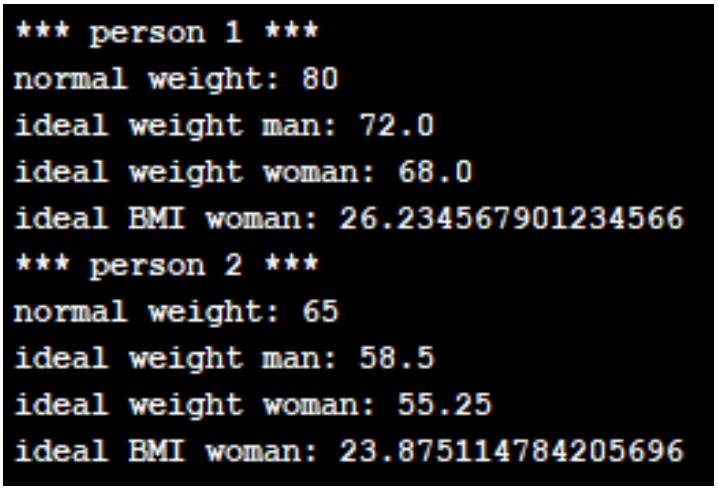
Valeur retournée :			

Remplissez le tableau de variables pour les valeurs ***n*** = 1234

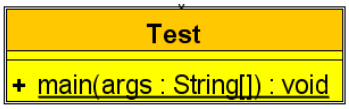
Valeur retournée :			

B) Classes Test pour les exercices officiels

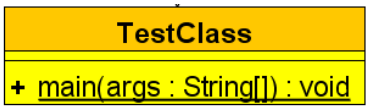
B.1) Exercice 9 – BodyStatistics

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un objet de type ***BodyStatistics*** et nommez celui-ci ***bodyStatistics1***. Pour les paramètres de ***bodyStatistics1*** : • ***pAge*** : 33 • ***pHeight*** : 180 • ***pWeight*** : 85 Créez un deuxième objet de type ***BodyStatistics*** et nommez celui-ci ***bodyStatistics2***. Pour les paramètres de ***bodyStatistics2*** : • ***pAge*** : 40 • ***pHeight*** : 165 • ***pWeight*** : 65
Méthodes à appeler / Tests à réaliser	Faites appel aux méthodes ***getNormalWeight***, ***getIdealWeightMan***, ***getIdealWeightWoman*** et ***getBMI*** pour l'objet ***bodyStatistics1***. Affichez les résultats des quatre méthodes en console. Faites appel aux méthodes ***getNormalWeight***, ***getIdealWeightMan***, ***getIdealWeightWoman*** et ***getBMI*** pour l'objet ***bodyStatistics2***. Affichez les résultats des quatre méthodes en console. Pour l'affichage en console, basez-vous sur l'aperçu du résultat en dessous.
Aperçu du résultat	

B.2) Exercice 10 – Randomizer

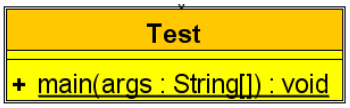
Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un objet de type ***Randomizer*** et nommez celui-ci ***randomizer***. Pour les paramètres de ***randomizer***: • ***pMin*** : 1, • ***pMax*** : 100.
Méthodes à appeler / Tests à réaliser	Faites trois fois appel à la méthode ***getNext*** pour afficher en console les trois nombres aléatoires générés. Respectez l'affichage de l'exemple en-dessous.
Aperçu du résultat	Aperçus possibles: 44 92 92 32 11 96

B.3) Exercice 11 – Test

Classe « TestClass » à créer	Puisqu'une classe avec le nom « Test » existe déjà, créez la classe ***TestClass***. 
Objets à créer	Créez deux objets de type ***Test*** et nommez ceux-ci ***test1*** et ***test2***. Pour ***test1***, utilisez 50 comme paramètre pour la création de l'objet. Pour ***test2***, utilisez un nombre aléatoire compris dans l'intervalle [1 ; 60] comme paramètre.
Méthodes à appeler / Tests à réaliser	Pour les deux objets, faites appel aux méthodes ***getMark*** et ***getEvaluation***. Affichez en console la note de l'objet suivie du résultat fourni par la méthode ***getEvaluation***. Respectez l'affichage de l'exemple en-dessous.

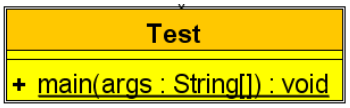
Aperçu du résultat	Aperçus possibles :
	`50: Test passed`
	`35: Test passed`
	
	`50: Test passed`
	`3: Test failed`

B.4) Exercice 12 – Cistern

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un objet de type ***Cistern*** et nommez celui-ci ***cistern***. Pour les paramètres de ***cistern*** : • ***pRadius*** : nombre aléatoire du type double compris dans l'intervalle [1 ; 8[• ***pHeight*** : nombre aléatoire du type double compris dans l'intervalle [1 ; 5[
Méthodes à appeler / Tests à réaliser	Faites appel à la méthode ***toString*** pour afficher en console la quantité d'eau disponible par rapport au contenu maximal de la citerne. Faites appel à la méthode ***add*** et ajoutez ***300*** litres à la citerne. Faites appel à la méthode ***toString*** pour afficher en console la quantité d'eau disponible par rapport au contenu maximal de la citerne. Respectez l'affichage de l'exemple ci-dessous.

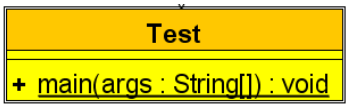
Aperçu du résultat	Aperçus possibles :
	`Fill level : 0.0 / 144477.260896652 liters (0.0%)`
	`Fill level : 300.0 / 144477.260896652 liters (0.20764513262374007%)`
	
	`Fill level : 0.0 / 185806.04510723802 liters (0.0%)`
	`Fill level : 300.0 / 185806.04510723802 liters (0.16145868657117957%)`

B.5) Exercice 13 - Article

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un objet de type ***Article*** et nommez celui-ci ***article***. Pour les paramètres de l'objet ***article*** : • ***pUnitPrice*** : nombre aléatoire du type double compris dans l'intervalle [1 ; 100[• ***pQuantity*** : nombre aléatoire du type int compris dans l'intervalle [1 ; 10]
Méthodes à appeler / Tests à réaliser	Affichez en console : • le prix unitaire et la quantité de l'article (deux nombres aléatoires que vous venez de créer) • le prix total, en appelant la méthode ***getTotalPrice*** Respectez l'affichage de l'exemple ci-dessous.

Aperçu du résultat	Aperçus possibles :
	`unit price (without VAT): 59.34294363049902, quantity: 9`
	`total price (with 15% VAT): 614.1994665756648`
	
	`unit price (without VAT): 80.86316877580884, quantity: 10`
	`total price (with 15% VAT): 929.9264409218016`

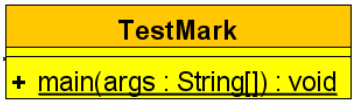
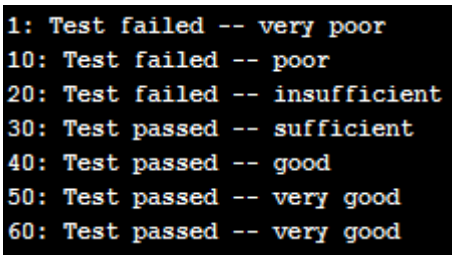
B.6) EXERCICE 14 - NUMBERPUZZLE

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un objet de type ***NumberPuzzle*** et nommez celui-ci ***numberPuzzle***.
Méthodes à appeler / Tests à réaliser	Faites appel aux méthodes : • ***selectSecretNumber*** qui prend ***10*** en paramètre, • ***guess*** qui prend ***5*** en paramètre, et affichez en console le résultat retourné par cette méthode.

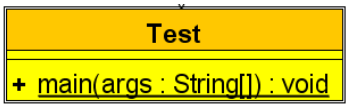
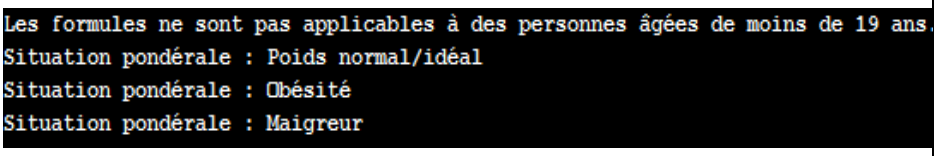
Aperçu du résultat	Aperçus possibles :
	`Your number is too small`
	`Well done! You found the secret number at the 1st guess`
	`Your number is too big`

B.7) Exercice 15 - Test Qualification

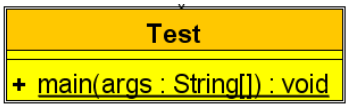
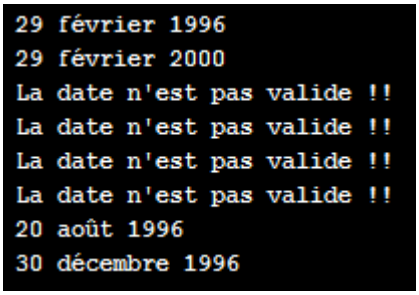
Avant de créer la classe Test, ajoutez un Manipulateur pour l'attribut ***mark***.

Classe « TestMark » à créer	Puisqu'une classe avec le nom « Test » existe déjà, créez la classe ***TestMark***. 
Objets à créer	Créez un objet de type ***Test*** en utilisant la valeur ***1*** comme note.
Méthodes à appeler / Tests à réaliser	Affichez dans la console la note, l'évaluation et la qualification en appelant les méthodes correspondantes. Modifiez la note en appelant le Manipulateur avec les valeurs 10, 20, 30, 40, 50 et 60. Après chaque modification, affichez dans la console l'évaluation et la qualification en appelant les méthodes correspondantes.
Aperçu du résultat	

B.8) EXERCICE 16 - BODYSTATISTICS II

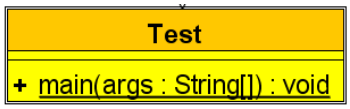
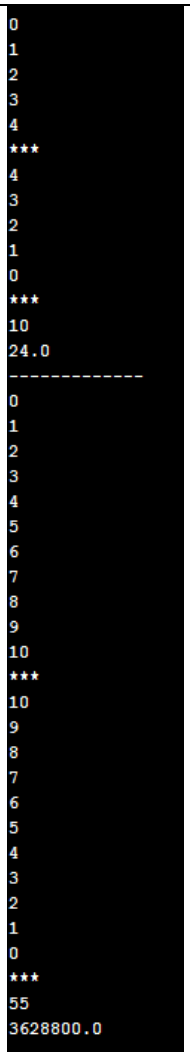
Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 4 objets de type ***BodyStatistics*** en utilisant les valeurs : • Age = 16, taille = 150, poids = 50, féminin • Age = 25, taille = 150, poids = 50, féminin • Age = 60, taille = 180, poids = 120, masculin • Age = 35, taille = 190, poids = 70.6, masculin
Méthodes à appeler / Tests à réaliser	Pour chaque objet, faites appel à la méthode ***getComment*** et affichez le résultat dans la console.
Aperçu du résultat	

B.9) Exercice 17 – AnalyseDate

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 8 objets de type ***AnalyseDate*** en utilisant les dates : • 29-2-1996 • 29-2-2000 • 21-13-2004 • 31-4-2010 • 29-2-2003 • 29-2-1900 • 20-8-1996 • 30-12-1996
Méthodes à appeler / Tests à réaliser	Pour chaque objet, affichez le résultat de la méthode ***toString*** en console.
Aperçu du résultat	

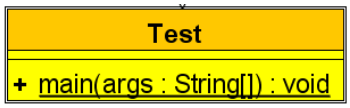
B.10) Exercice 20 et 21 - Calculs avec un entier

Notez que vous pouvez utiliser la même classe « Test » pour les deux exercices, car le nom de la classe et les noms des méthodes sont identiques.

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 2 objets de type ***SimpleCalculationsWithOneInt*** en utilisant les valeurs : • 4 • 10
Méthodes à appeler / Tests à réaliser	Pour chaque objet, appelez toutes les méthodes et affichez les résultats dans la console.
Aperçu du résultat	

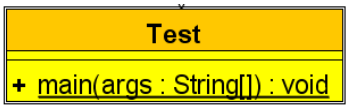
B.11) Exercice 22 - Calculs avec deux entiers

Avant de commencer, ajoutez des accesseurs dans la classe ***SimpleCalculationsWithTwoInts*** pour ***a*** et ***b***.

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez deux objets de type ***SimpleCalculationsWithTwoInts*** en utilisant les valeurs : • 2 et 5 • 6 et 12
Méthodes à appeler / Tests à réaliser	Pour chaque objet, appelez toutes les méthodes. Après l'appel de la méthode ***swap***, affichez les valeurs de ***a*** et de ***b*** dans la console, puis répétez l'appel et l'affichage.

Aperçu du résultat	`2 3 4 5`
	`***`
	`6`
	`32.0`
	`a & b: 5 2`
	`a & b: 2 5`
	`GCD Search (3 versions) 1 1 1`
	`GCD Euclid (2 versions) 1 1`
	`LCM (2 versions) 10 10`
	`------`
	`6 7 8 9 10 11 12`
	`***`
	`36`
	`2.176782336E9`
	`a & b: 12 6`
	`a & b: 6 12`
	`GCD Search (3 versions) 6 6 6`
	`GCD Euclid (2 versions) 6 6`
	`LCM (2 versions) 12 12`

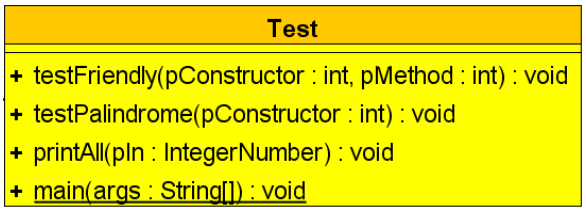
B.12) Exercice 24 - Simulation de jets de deux des

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un objet de type ***DoubleDice***.
Méthodes à appeler / Tests à réaliser	Dans une boucle « for » avec 8 tours, commençant par i = 1 : • Faites appel à la méthode ***rollSeries*** en passant la valeur du compteur de la boucle comme paramètre. • Faites appel à la méthode ***showStatistics***. • Faites appel à la méthode ***reset***.

Aperçu du résultat	Notez que la capture est seulement un exemple, car des valeurs aléatoires sont utilisées dans l'exercice.
	`dice 1: 2; dice 2: 1`
	`after 1 tries, you had 0 doubles, [0.0 %]`
	`dice 1: 5; dice 2: 1`
	`dice 1: 5; dice 2: 5 ***DOUBLE***`
	`after 2 tries, you had 1 doubles, [50.0 %]`
	`dice 1: 4; dice 2: 1`
	`dice 1: 4; dice 2: 5`
	`dice 1: 4; dice 2: 4 ***DOUBLE***`
	`after 3 tries, you had 1 doubles, [33.33333333333333 %]`
	`dice 1: 4; dice 2: 3`
	`dice 1: 2; dice 2: 4`
	`dice 1: 5; dice 2: 5 ***DOUBLE***`
	`dice 1: 5; dice 2: 4`
	`after 4 tries, you had 1 doubles, [25.0 %]`
	`dice 1: 1; dice 2: 1 ***DOUBLE***`
	`dice 1: 4; dice 2: 5`
	`dice 1: 6; dice 2: 1`
	`dice 1: 3; dice 2: 1`
	`dice 1: 2; dice 2: 3`
	`after 5 tries, you had 1 doubles, [20.0 %]`
	`dice 1: 1; dice 2: 4`
	`dice 1: 2; dice 2: 1`
	`dice 1: 4; dice 2: 5`
	`dice 1: 4; dice 2: 6`
	`dice 1: 2; dice 2: 6`
	`dice 1: 2; dice 2: 5`
	`after 6 tries, you had 0 doubles, [0.0 %]`
	`dice 1: 6; dice 2: 6 ***DOUBLE***`
	`dice 1: 2; dice 2: 1`
	`dice 1: 1; dice 2: 4`
	`dice 1: 3; dice 2: 5`
	`dice 1: 2; dice 2: 3`
	`dice 1: 5; dice 2: 6`
	`dice 1: 6; dice 2: 5`
	`after 7 tries, you had 1 doubles, [14.285714285714285 %]`
	`dice 1: 3; dice 2: 4`
	`dice 1: 4; dice 2: 1`
	`dice 1: 4; dice 2: 1`
	`dice 1: 5; dice 2: 6`
	`dice 1: 6; dice 2: 5`
	`dice 1: 4; dice 2: 1`
	`dice 1: 1; dice 2: 6`
	`dice 1: 4; dice 2: 1`
	`after 8 tries, you had 0 doubles, [0.0 %]`

B.13) Exercice 26 - IntegerNumber

Utilise des méthodes « classe comme paramètre » et créez la Classe « Test ».



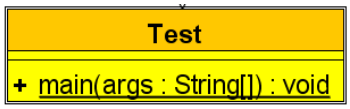
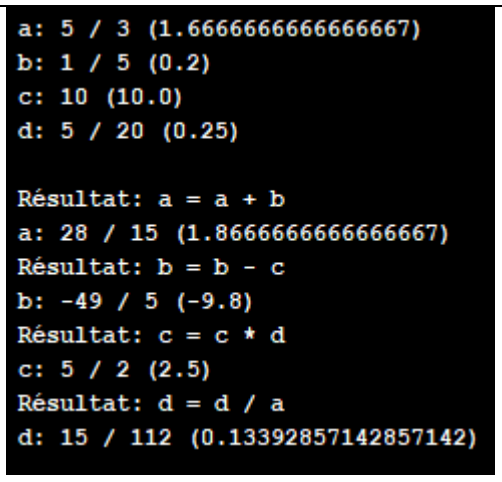
Partie	Description
Méthode : testFriendly	Créez un nouvel objet de type ***IntegerNumber*** en utilisant la valeur de ***pConstructor*** dans le constructeur. Affichez le texte suivant dans la console : <pConstructor> et <pMethod> sont amicaux ? <valeur de retour de la méthode ***isFriendlyTo***> Les textes entre chevrons sont à remplacer par des valeurs concrètes. ***pMethod*** est à utiliser pendant l'appel de la méthode ***isFriendlyTo***. Voici un exemple pour les nombres 10 et 20 : `10 et 20 sont amicaux ? false`
Méthode : testPalindrome	Créez un nouvel objet de type ***IntegerNumber*** en utilisant la valeur de ***pConstructor*** dans le constructeur. Affichez le texte suivant dans la console : <pConstructor> est un palindrome ? <valeur de retour de la méthode ***isPalindrome***> Les textes entre chevrons sont à remplacer par des valeurs concrètes. Voici un exemple pour le nombre 1234 : `1234 est un palindrome ? false`

Partie	Description

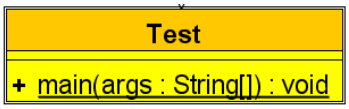
Méthode : printAll	Cette méthode fait appel à toutes les méthodes de la classe ***IntegerNumber*** et affiche leur résultat dans la console.
	Seules les méthodes ***isFriendlyTo*** et ***isPalindrome*** ne sont pas considérées.
	Voici un exemple pour le nombre 12 :
	`Pour 12`
	`Even: true`
	`Prime: false`
	`Sum of dividers: 28`
	`Perfect: false`
	`Deficient: false`
	`Abundant: true`
	`Reverse: 21`
	`Palindrome: false`

Méthode : main	Créez une nouvelle instance de la classe « Test ». Faites appel à la méthode ***testFriendly*** avec les valeurs : ○ 220 et 284 (résultat : true) ○ 10 et 20 (résultat : false) ○ 1184 et 1210 (résultat : true) ○ 30 et 64 (résultat : false) Faites appel à la méthode ***testPalindrome*** avec les valeurs : ○ 1234 (résultat : false) ○ 1221 (résultat : true) Utilisez une boucle « for » avec 20 tours. Pour chaque tour : • Créez un objet de type ***IntegerNumber***. • La valeur pour le paramètre du constructeur est une valeur aléatoire entre 10 et 9999. • Faites appel à la méthode ***printAll***.

B.14) Exercice 27 - Fractions

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 4 objets du type ***Fraction*** en utilisant les noms et les valeurs spécifiés ci-dessous : • a : 5 et 3 • b : 1 et 5 • c : 10 et 1 • d : 5 et 20
Méthodes à appeler / Tests à réaliser	Affichez la valeur des objets dans la console en appelant la méthode ***toString***. Faites une addition entre ***a*** et ***b*** et affichez la valeur de ***a*** dans la console (***a*** = ***a*** + ***b***). Faites une soustraction entre ***b*** et ***c*** et affichez la valeur de ***b*** dans la console (***b*** = ***b*** – ***c***). Faites une multiplication entre ***c*** et ***d*** et affichez la valeur de ***c*** dans la console (***c*** = ***c*** * ***d***). Faites une division entre ***d*** et ***a*** et affichez la valeur de ***d*** dans la console (***d*** = ***d*** / ***a***).
Aperçu du résultat	

B.15) EXERCICE 28 -COMPTES BANCAIRES II

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 3 objets de type ***Account*** en utilisant les noms et valeurs suivants : • ***a*** : 0 et 500 • ***b*** : 0 et 100 • ***c*** : 0 et 1000
Méthodes à appeler / Tests à réaliser	Affichez les objets en console, en appelant la méthode ***toString*** sur chaque objet. Puis faites les étapes suivantes : • Transférez 200 euros de ***a*** vers ***b*** • Affichez le résultat de l'opération en console • Affichez les deux comptes en console • Transférez 301 euros de ***b*** vers ***c*** • Affichez le résultat de l'opération en console • Affichez les deux comptes en console

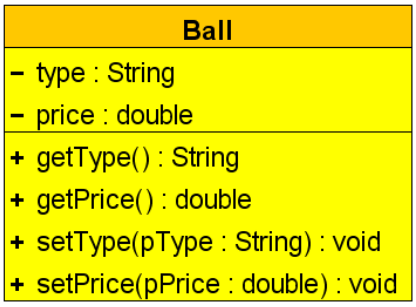
Aperçu du résultat	`a: Account Nr.: 208346629 - Current balance: 500.0`
	`b: Account Nr.: 434326527 - Current balance: 100.0`
	`c: Account Nr.: 987974347 - Current balance: 1000.0`
	`SUCCESS: The transfer has been accomplished successfully!`
	`a: Account Nr.: 208346629 - Current balance: 300.0 ---> b: Account Nr.: 434326527 - Current balance: 300.0`
	`ERROR: Your account's balance is too low for this transfer!`
	`b: Account Nr.: 434326527 - Current balance: 300.0 ---> c: Account Nr.: 987974347 - Current balance: 1000.0`

C) EXERCICES PRATIQUES

C.1) BALL (CLASSE SANS CONSTRUCTEUR)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Créez la classe ***Ball*** qui représente une balle. Ajoutez les attributs et méthodes de l'UML ci-dessous. Enregistrez le projet sous le nom ***Ball (sans constructeur)***.

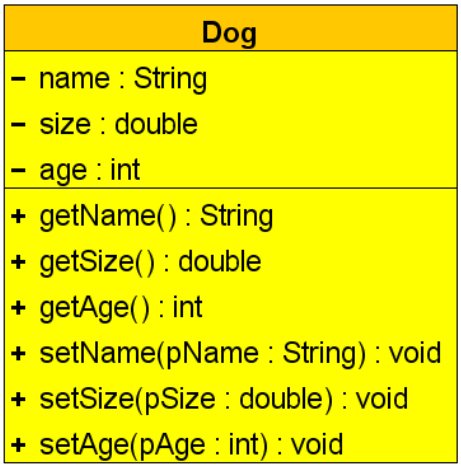


Partie	Description
Attribut : type	Cet attribut représente le type de la balle. Exemples : football, tennis, basketball, ping pong, baseball, … Initialisez l'attribut avec la valeur ***baseball***.
Attribut : price	Cet attribut représente le prix de la balle sans TVA. Initialisez l'attribut avec la valeur ***5,13***.
Accesseurs & Manipulateurs	Les méthodes ***getType*** et ***getPrice*** sont des accesseurs pour les attributs respectifs. Les méthodes ***setType*** et ***setPrice*** sont des manipulateurs pour les attributs respectifs.

C.2) Dog (classe sans constructeur)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Créez la classe ***Dog*** qui représente un chien. Ajoutez les attributs et méthodes de l'UML ci-dessous. Enregistrez le projet sous le nom ***Dog (sans constructeur)***.

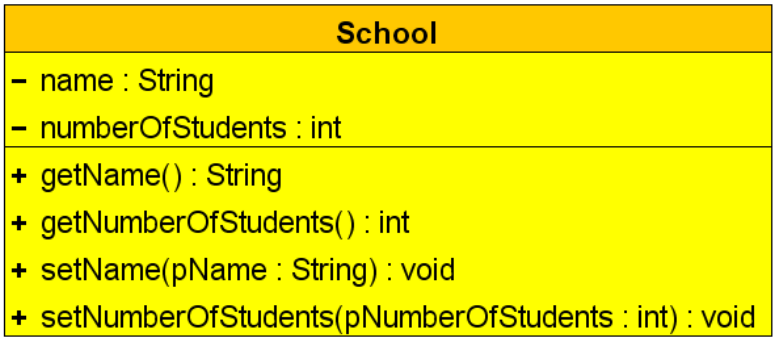


Partie	Description
Attribut : name	Cet attribut représente le nom du chien. Initialisez l'attribut avec la valeur ***Tacco***.
Attribut : size	Cet attribut représente la taille du chien (en m). Initialisez l'attribut avec la valeur ***0,53***.
Attribut : age	Cet attribut représente l'âge du chien (en années). Initialisez l'attribut avec la valeur ***2***.
Accesseurs & Manipulateurs	Les méthodes ***getName***, ***getSize*** et ***getAge*** sont des accesseurs pour les attributs respectifs. Les méthodes ***setName***, ***setSize*** et ***setAge*** sont des manipulateurs pour les attributs respectifs.

C.3) SCHOOL (CLASSE SANS CONSTRUCTEUR)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Créez la classe ***School*** qui représente une école au Luxembourg. Ajoutez les attributs et méthodes de l'UML ci-dessous. Enregistrez le projet sous le nom ***School (sans constructeur)***.



Partie	Description
Attribut : name	Cet attribut représente le nom de l'école. Exemples : Atert-Lycée Redange, Lycée Guillaume Kroll, Lycée Aline Mayrisch, … Initialisez l'attribut avec la valeur ***Atert-Lycée Redange***.
Attribut : numberOfStudents	Cet attribut représente le nombre d'élèves inscrits à l'école. Initialisez l'attribut avec la valeur ***1300***.
Accesseurs & Manipulateurs	Les méthodes ***getName*** et ***getNumberOfStudents*** sont des accesseurs pour les attributs respectifs. Les méthodes ***setName*** et ***setNumberOfStudents*** sont des manipulateurs pour les attributs respectifs.

C.4) THEATRE (CLASSE SANS CONSTRUCTEUR)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Voici le code source de la classe ***Theatre***. Créez l'UML correspondant au code source ci-dessous.

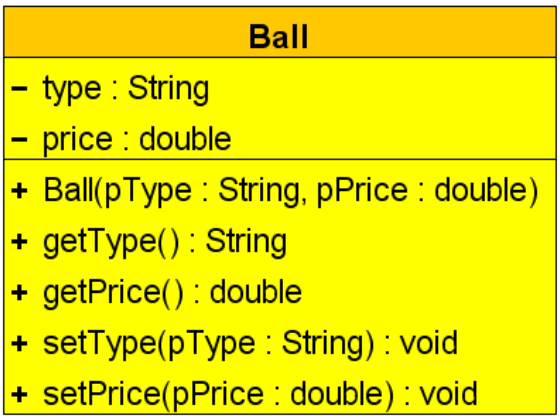
```
public class Theatre
{
	private String name = "Grand Theatre";
	private int capacity = 500;

	public String getName()
	{
		return name;
	}

	public int getCapacity()
	{
		return capacity;
	}

	public void setName(String pName)
	{
		name = pName;
	}

	public void setCapacity(int pCapacity)
	{
		capacity = pCapacity;
	}
}
```

C.5) BALL (CLASSE AVEC CONSTRUCTEUR)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Ouvrez votre projet ***Ball (sans constructeur)*** et renommez celui-ci en ***Ball (avec constructeur).***

Ajoutez un constructeur, tel qu'indiqué dans l'UML ci-dessous.

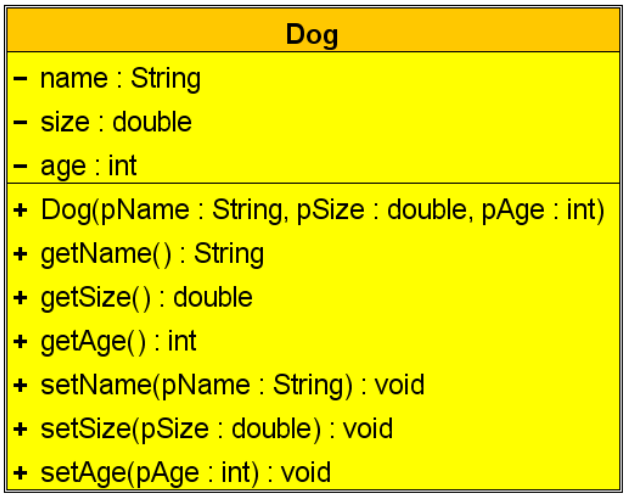


C.6) Dog (avec constructeur)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Ouvrez votre projet ***Dog (sans constructeur)*** et renommez celui-ci en ***Dog (avec constructeur).***

Ajoutez un constructeur, tel qu'indiqué dans l'UML ci-dessous.

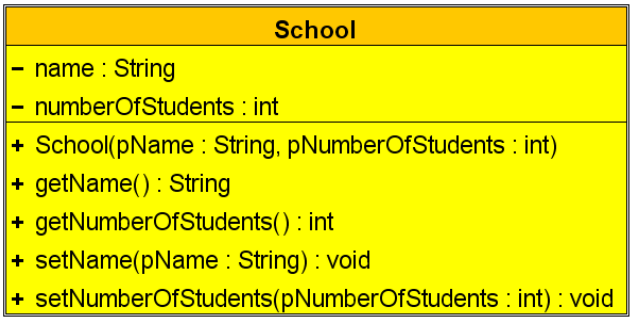


C.7) SCHOOL (CLASSE AVEC CONSTRUCTEUR)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Ouvrez votre projet ***School (sans constructeur)*** et renommez celui-ci en ***School (avec constructeur).***

Ajoutez un constructeur, tel qu'indiqué dans l'UML ci-dessous.



C.8) THEATRE2 (CLASSE AVEC CONSTRUCTEUR)

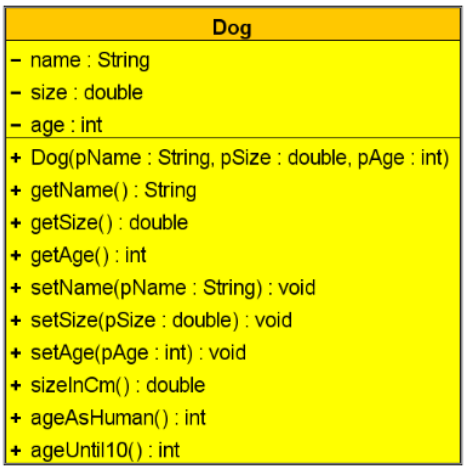
X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Voici le code source de la classe ***Theatre2***. Créez l'UML correspondant au code source ci-dessous.

```
public class Theatre2
{
	private String name;
	private int capacity;

	public Theatre2(String pName, int pCapacity)
	{
		name = pName;
		capacity = pCapacity;
	}

	public String getName()
	{
		return name;
	}

	public int getCapacity()
	{
		return capacity;
	}

	public void setName(String pName)
	{
		name = pName;
	}

	public void setCapacity(int pCapacity)
	{
		capacity = pCapacity;
	}
}
```

C.9) DOG (AVEC METHODES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

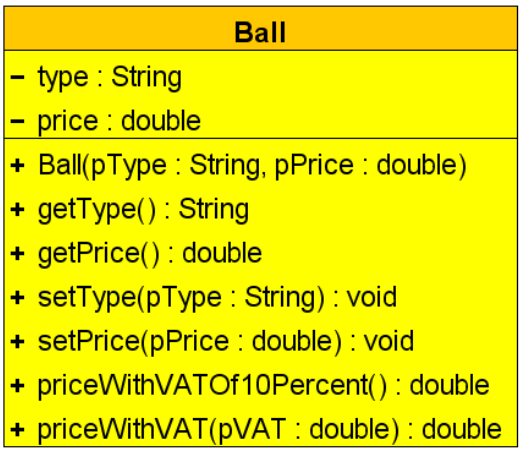


Ouvrez votre projet ***Dog (avec constructeur)*** et renommez celui-ci en ***Dog (avec méthodes)***.

Partie	Description
Attribut : name	Cet attribut représente le nom du chien.
Attribut : size	Cet attribut représente la taille du chien (en m).
Attribut : age	Cet attribut représente l'âge du chien (en années).
Méthode : sizeInCm	Cette méthode retourne la taille du chien en cm.
Méthode : ageAsHuman	Cette méthode retourne l'âge du chien s'il était un être humain. Pour ceci on multiplie l'âge du chien par 7.
Méthode : ageUntil10	Cette méthode retourne le nombre d'années dans lequel le chien aura 10 ans. Si le chien a déjà plus de 10 ans, la méthode retourne un nombre négatif. Exemples : Si age = 3 alors le résultat est 7, car 10 - 3 = 7 Si age = 8 alors le résultat est 2, car 10 - 8 = 2 Si age = 12 alors le résultat est -2, car 10 - 12 = -2

C.10) BALL (AVEC METHODES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



Ouvrez votre projet ***Ball (avec constructeur)*** et renommez celui-ci en ***Ball (avec méthodes)***.

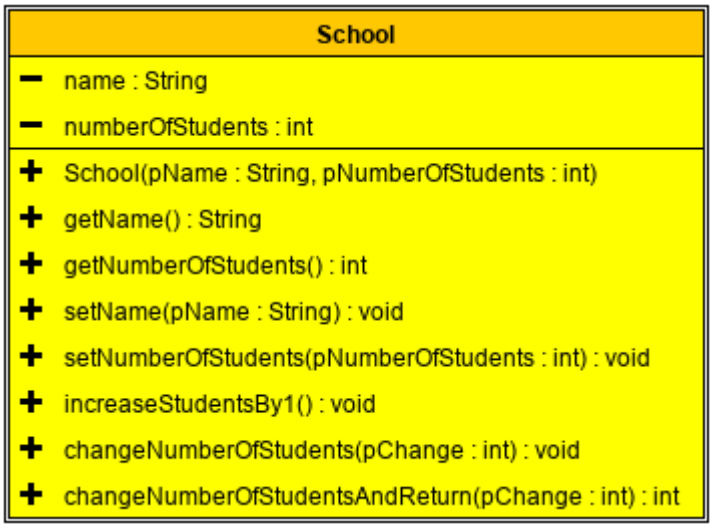
Ajoutez les méthodes (non encore existantes) décrites ci-dessous.

Partie	Description
Attribut : **type**	Cet attribut représente le type de la balle. Exemples: football, tennis, basketball, ping pong, baseball, …
Attribut : **price**	Cet attribut représente le prix de la balle sans TVA.
Méthode : **priceWithVATOf10Percent**	Cette méthode retourne le prix de la balle avec une TVA de 10%.
Méthode : **priceWithVAT**	Cette méthode retourne le prix de la balle avec une TVA de pVAT. On admet que pVAT est un nombre compris dans l'intervalle [0 ;1]. Si pVat = 0,1 ceci représente 10%. Si pVat = 0,05 ceci représente 5%. Si pVat = 0,45 ceci représente 45%.

C.11) SCHOOL (AVEC METHODES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

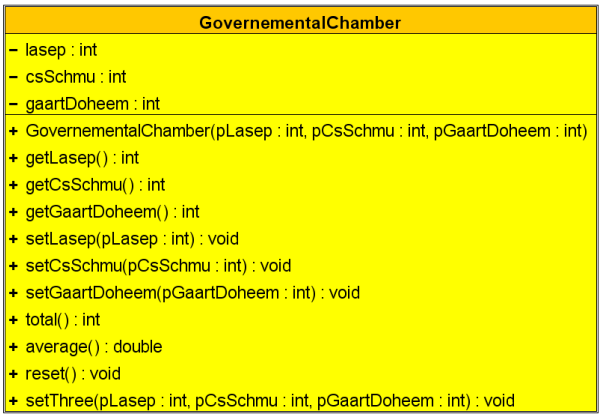
Ouvrez votre projet ***School (avec constructeur)*** et renommez celui-ci en ***School (avec méthodes)***.



Partie	Description
Attribut : name	Cet attribut représente le nom de l'école. Exemples : Atert-Lycée Redange, Lycée Guillaume Kroll, Lycée Aline Mayrisch, …
Attribut : numberOfStudents	Cet attribut représente le nombre d'élèves inscrits à l'école.
Méthode : increaseStudentsBy1	Cette méthode ajoute un élève à ***numberOfStudents.***
Méthode : changeNumberOfStudents	Cette méthode ajoute ***pChange*** élève(s) à ***numberOfStudents.***
Méthode : changeNumberOfStudentsAndReturn	Cette méthode ajoute ***pChange*** élève(s) à ***numberOfStudents.*** Le nouveau nombre d'élèves est retourné par cette méthode.

C.12) GOVERNMENTALCHAMBER (DOUBLE/INT)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

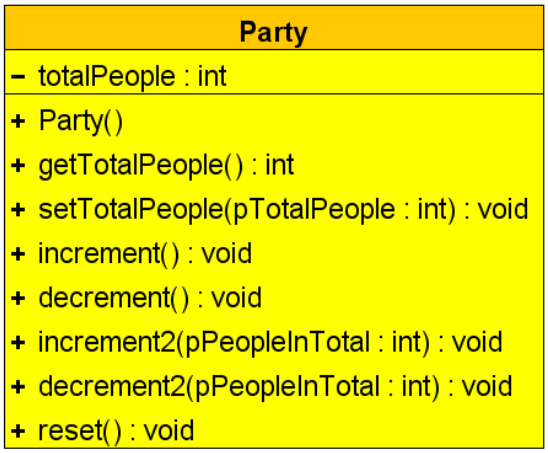


Créez la classe ***GovernmentalChamber*** qui représente la chambre avec les répartitions des sièges de différents partis politiques au Luxembourg. Ajoutez les méthodes et attributs de l'UML ci-dessous.

Partie	Description
Attribut : lasep	Cet attribut représente le nombre de sièges du parti politique Lasep.
Attribut : csSchmu	Cet attribut représente le nombre de sièges du parti politique CsSchmu.
Attribut : gaartDoheem	Cet attribut représente le nombre de sièges du parti politique GaartDoheem.
Constructeur	Ajoutez un constructeur qui initialise les attributs avec les valeurs des paramètres.
Accesseurs & Manipulateurs	Créer les accesseurs et manipulateurs pour tous les attributs.
Méthode : total	Cette méthode retourne le nombre total de sièges des 3 partis politiques.
Méthode : average	Cette méthode retourne la moyenne du nombre de sièges des 3 partis politiques.
Méthode : reset	Cette méthode réinitialise les valeurs des 3 attributs à ***0***.
Méthode : setThree	Cette méthode réinitialise les valeurs des 3 attributs aux paramètres qui lui sont fournis.

C.13) PARTY (AVEC CONSTRUCTEUR ET METHODES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



Créez un nouveau projet avec le nom ***Party***. Cette classe, décrite ci-dessous, représente un simple compteur qui indique le nombre de personnes qui se situent à des fêtes différentes. Les explications pour les attributs et méthodes se trouvent dans le tableau plus bas.

Exemples :

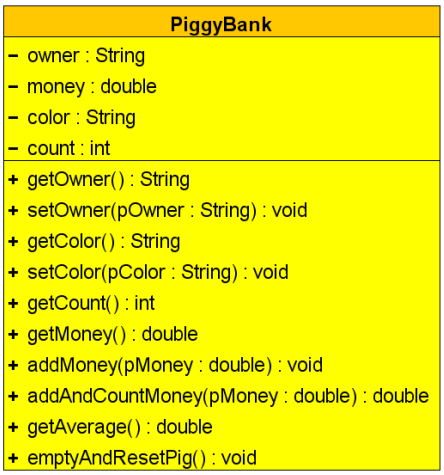
Melusina : 154 personnes,

M Club : 129 personnes

Partie	Description
Attribut : totalPeople	Cet attribut indique le nombre de personnes à une fête.
Constructeur	Ajoutez un constructeur qui initialise l'attribut ***totalPeople*** à 0.
Accesseurs & Manipulateurs	La méthode ***getTotalPeople*** est un accesseur pour l'attribut respectif. La méthode ***setTotalPeople*** est un Manipulateur pour l'attribut respectif.
Méthode : increment	Cette méthode incrémente l'attribut ***totalPeople*** de 1.
Méthode : decrement	Cette méthode décrémente l'attribut ***totalPeople*** de 1.
Méthode : increment2	Cette méthode incrémente l'attribut ***totalPeople*** de ***pPeopleInTotal***.
Méthode : decrement2	Cette méthode décrémente l'attribut ***totalPeople*** de ***pPeopleInTotal***.
Méthode : reset	Cette méthode réinitialise l'attribut ***totalPeople*** sur ***0***.

C.14) TIRELIRE – PHASE 1

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

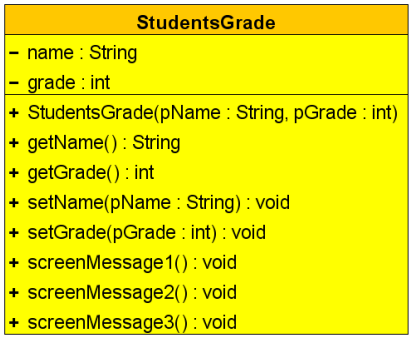


Créez la classe ***PiggyBank*** telle que décrite ci-dessous. Cette classe représente une tirelire. Les explications pour les attributs et méthodes se trouvent dans le tableau.

Partie	Description
Attribut : owner	Cet attribut représente le nom du propriétaire de la tirelire. Initialisez cet attribut à ***Pitti***.
Attribut : money	Cet attribut représente l'argent déjà inséré dans la tirelire. Initialisez cet attribut à ***0***.
Attribut : color	Cet attribut représente la couleur de la tirelire. Initialisez cet attribut à ***rouge***.
Attribut : count	Cet attribut représente combien de fois on a inséré de l'argent dans la tirelire. Initialisez cet attribut à ***0***.
Accesseurs & Manipulateurs	Les méthodes ***getOwner***, ***getColor***, ***getCount*** et ***getMoney*** sont des accesseurs pour les attributs respectifs. Les méthodes ***setOwner*** et ***setColor*** sont des manipulateurs pour les attributs respectifs.
Méthode : addMoney	Cette méthode ajoute de l'argent à la tirelire. N'oubliez pas de modifier l'attribut ***count***.
Méthode : addAndCountMoney	La méthode ***addAndCountMoney*** est identique à la méthode ***addMoney***, mais retourne aussi combien d'argent il y a dans la tirelire.
Méthode : getAverage	Cette méthode retourne la somme moyenne insérée dans la tirelire à chaque dépôt.
Méthode : emptyAndResetPig	Cette méthode réinitialise la tirelire à l'état initial. Sans modifier la couleur ou le propriétaire.

C.15) StudentsGrade (affichage en console)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



Créez un nouveau projet avec le nom ***StudentsGrade***. Cette classe, décrite ci-dessous, représente un élève et sa note en informatique. Les explications pour les attributs et méthodes se trouvent en dessous.

Partie	Description
Attribut : name	Cet attribut indique le nom de l'élève.
Attribut : grade	Cet attribut indique la note de l'élève.
Constructeur	Ajoutez un constructeur qui initialise les deux attributs avec les valeurs des paramètres.
Accesseurs & Manipulateurs	Créez les accesseurs et manipulateurs pour les attributs ***name*** et ***grade***.
Méthode : screenMessage1	Cette méthode affiche en console le nom et la note de l'élève sous le format suivant : <nom de l'élève>: <note de l'élève>! Exemple avec ***name*** = Pit et ***grade*** = 45 `Pit: 45!`

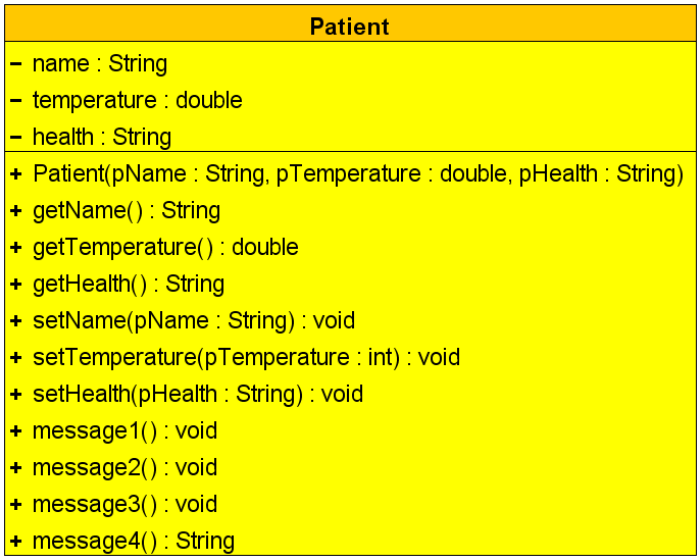
Méthode : screenMessage2	Cette méthode affiche en console le nom et la note de l'élève sous le format suivant :
	<nom de l'élève>
	--> <note de l'élève>!
	Exemple pour ***name*** = Pit et ***grade*** = 45
	`Pit`
	`--> 45!`

Méthode : screenMessage3	Cette méthode affiche en console le nom et la note de l'élève sous le format suivant : <nom de l'élève> a obtenu <note de l'élève> points dans le test en informatique! Exemple pour ***name*** = Pit et ***grade*** = 45 `Pit a obtenu 45 points dans le test en informatique!`

C.16) PATIENT (AFFICHAGE EN CONSOLE)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Créez un nouveau projet avec le nom ***Patient*** décrite par le diagramme UML. Cette classe représente un patient avec son nom, son statut de santé (malade/sain) et sa température corporelle. Les explications pour les attributs et méthodes se trouvent en dessous.



Partie	Description
Attribut : name	Cet attribut indique le nom du patient.
Attribut : temperature	Cet attribut indique la température du patient.
Attribut : health	Cet attribut indique le statut de santé du patient (malade/sain).
Constructeur	Ajoutez un constructeur qui initialise les trois attributs sur les valeurs des paramètres.
Accesseurs & Manipulateurs	Créez les accesseurs et manipulateurs pour les attributs ***name***, ***temperature*** et ***health***.
Méthode : message1	Cette méthode affiche en console le nom, la température et l'état de santé du patient sous le format suivant : <nom>: <température> --> <état de santé> Exemple pour ***name*** = Ana, ***temperature*** = 36,3 et ***health*** = sain `Ana: 36.3 --> sain`

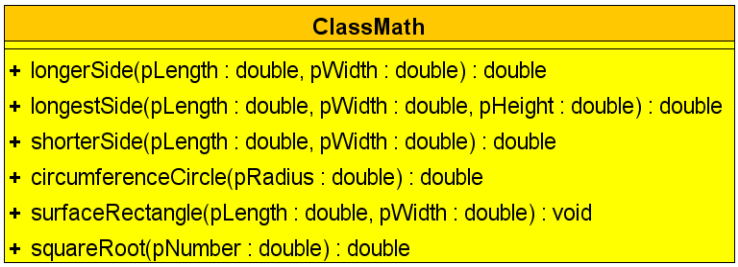
Partie	Description

Méthode : message2	Cette méthode affiche en console le nom, la température et l'état de santé du patient sous le format suivant :
	<nom>
	--> <température>
	(<statut de santé>)
	Exemple pour ***name*** = Jean, ***temperature*** = 39 et ***health*** = malade
	`Jean`
	`--> 39.0`
	`(malade)`

Méthode : message3	Cette méthode affiche en console le nom, la température et l'état de santé du patient sous le format suivant : Le patient <nom> a <température> degrés de fièvre. Il est <statut de santé>! Exemple pour ***name*** = Jean, ***temperature*** = 39 et ***health*** = malade `Le patient Jean a 39.0 degrés de température. Il est malade!`
Méthode : message4	Cette méthode retourne un objet de type String contenant le nom, la température et l'état de santé du patient, sous le format suivant : Le patient <nom> a <température> degrés de fièvre. Il est <statut de santé >! Exemple pour ***name*** = Jean, ***temperature*** = 39 et ***health*** = malade `Le patient Jean a 39.0 degrés de température. Il est malade!`

C.17) CLASSMATH

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



Créez un nouveau projet avec le nom ***ClassMath***. Les explications pour les attributs et méthodes de cette classe se trouvent en dessous.

Le but de cet exercice est d'utiliser le plus possible les méthodes prédéfinies par la classe Math : Math.PI, Math.abs(…), Math.max(…, …), Math.min(…, …), Math.round(…), Math.sqrt(…), Math.pow(…, …).

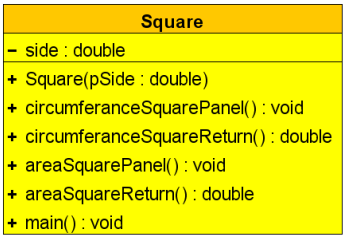
Partie	Description
Méthode : longerSide	Cette méthode retourne comme résultat la longueur du plus grand côté d'un rectangle de dimensions ***pLength*** x ***pWidth***.
Méthode : longestSide	Cette méthode retourne comme résultat la longueur du plus grand côté d'un pavé droit (parallélépipède rectangle) de dimensions ***pLength*** x ***pWidth*** x ***pHeight***.
Méthode : shorterSide	Cette méthode retourne comme résultat la longueur du plus petit côté d'un rectangle de dimensions ***pLength*** x ***pWidth***.
Méthode : circumferenceCircle	Cette méthode retourne la circonférence d'un cercle de rayon ***pRadius***.

Méthode : surfaceRectangle	Cette méthode affiche en console :
	• l'aire d'un rectangle
	• l'aire d'un rectangle arrondie à la valeur de l'entier inférieur.
	Les côtés du rectangle sont ***pLength*** et ***pWidth***.
	Pour l'affichage, basez-vous sur l'exemple ci-dessous pour un rectangle de côtés ***pLength*** = 3,69 et ***pWidth*** = 7,41 :
	`surface of rectangle: 27.3429`
	`rounded surface of rectangle: 27`

Méthode : squareRoot	Cette méthode retourne la racine carrée du nombre ***pNumber*** ou de sa valeur absolue s'il s'agit d'un nombre négatif. Pour ***pNumber*** = 9 le résultat est 3, car $\sqrt{9} = 3$ Pour ***pNumber*** = -9 le résultat est 3, car |-9| = 9 et $\sqrt{9} = 3$

C.18) SQUARE (APPEL DES METHODES - SANS PARAMETRES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



Demandez à l'enseignant de vous fournir le projet ***Square (appel des méthodes - sans parametres - ELEVES)***. Cette classe représente un carré. Complétez uniquement la méthode ***main*** sans faire de changements aux autres parties.

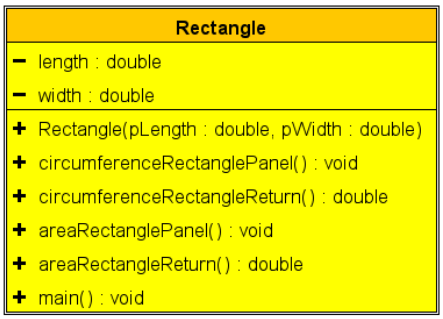
Les explications pour les attributs et méthodes se trouvent ci-dessous. Respectez l'UML ci-dessus.

Partie	Description
Attribut : side	**(à lire)** Cet attribut indique la longueur du côté du carré.
Constructeur	**(à lire)** Le constructeur initialise les attributs.
Méthode : circumferenceSquarePanel	**(à lire)** Cette méthode affiche dans la console l'aire du carré.
Méthode : circumferenceSquareReturn	**(à lire)** Cette méthode retourne le périmètre du carré.
Méthode : areaSquarePanel	**(à lire)** Cette méthode affiche dans la console l'aire du carré.
Méthode : areaSquareReturn	**(à lire)** Cette méthode retourne l'aire du carré.

Méthode : main	Faites appel aux méthodes suivantes :
	• ***circumferenceSquarePanel*** pour afficher le périmètre du carré
	• ***circumference SquareReturn*** pour afficher le périmètre du carré
	• ***areaSquarePanel*** pour afficher l'aire du carré
	• ***areaSquareReturn*** pour afficher l'aire du carré
	Affichez ensuite la somme du périmètre et de l'aire du carré.
	Tous les affichages sont à faire en console. Respectez l'affichage de l'exemple ci-dessous.
	`length: 5.0 width: 6.0`
	`**********`
	`circumference of rectangle: 22.0`
	`circumferance of rectangle: 22.0`
	`area of rectangle: 30.0`
	`area of rectangle: 30.0`
	`sum of circumferance and area of rectangle: 52.0`

C.19) RECTANGLE (APPEL DES METHODES - SANS PARAMETRES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



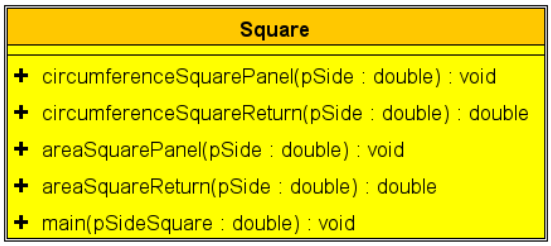
Demandez à l'enseignant de vous fournir le projet ***Rectangle (appel des méthodes - sans parametres - ELEVES)***. Cette classe représente un rectangle. Complétez uniquement la méthode ***main*** sans faire de changements aux autres parties.
Les explications pour les attributs et méthodes se trouvent ci-dessous. Respectez l'UML ci-dessus.

Partie	Description
Attributs : **length et width**	**(à lire)** Ces attributs indiquent la longueur et la largeur du rectangle.
Constructeur	**(à lire)** Le constructeur initialise les attributs.
Méthode : **circumferenceRectangle Panel**	**(à lire)** Cette méthode affiche dans la console le périmètre du rectangle.
Méthode : **circumferenceRectangle Return**	**(à lire)** Cette méthode affiche l'aire de la figure en console.**(à lire)** Cette méthode retourne le périmètre du rectangle.
Méthode : **areaRectanglePanel**	**(à lire)** Cette méthode retourne le **périmètre** de la figure en.**(à lire)** Cette méthode affiche dans la console l'aire du rectangle.
Méthode : **areaRectangleReturn**	**(à lire)** Cette méthode retourne l'aire du rectangle.
Méthode : **main**	Faites appel aux méthodes : • ***circumferenceRectanglePanel*** pour afficher le périmètre du rectangle • ***circumferenceRectangleReturn*** pour afficher le périmètre du rectangle • ***areaRectanglePanel*** pour afficher l'aire du rectangle • ***areaRectangleReturn*** pour afficher l'aire du rectangle Affichez ensuite la somme du périmètre et de l'aire du rectangle. Tous les affichages sont à faire en console.

	Respectez l'affichage de l'exemple ci-dessous.
	`length: 5.0 width: 6.0`
	`**********`
	
	`circumference of rectangle: 22.0`
	`circumferance of rectangle: 22.0`
	
	`area of rectangle: 30.0`
	`area of rectangle: 30.0`
	
	`sum of circumferance and area of rectangle: 52.0`

C.20) SQUARE (APPEL DES METHODES - AVEC PARAMETRES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



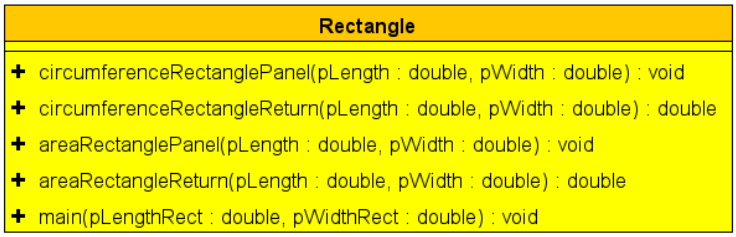
Demander à l'enseignant de vous fournir le projet ***Square (appel des methodes - avec parametres - ELEVES)***. Cette classe représente un carré. Complétez uniquement la méthode ***main*** sans réaliser des changements des autres parties.

Partie	Description
Méthode : circumferenceSquarePanel	**(à lire)** Cette méthode affiche le périmètre de la figure en console. ***pSide*** indique la longueur du côté du carré.
Méthode : circumferenceSquareReturn	**(à lire)** Cette méthode retourne le périmètre du carré. ***pSide*** indique la longueur du côté du carré.
Méthode : areaSquarePanel	**(à lire)** Cette méthode affiche dans la console l'aire du carré. ***pSide*** indique la longueur du côté du carré.
Méthode : areaSquareReturn	**(à lire)** Cette méthode retourne l'aire du carré. ***pSide*** indique la longueur du côté du carré.

Méthode : main	***pSide*** indique la longueur du côté du carré.
	Faites appel aux méthodes :
	• ***circumferenceSquarePanel*** pour afficher le périmètre du carré
	• ***circumferenceSquareReturn*** pour afficher le périmètre du carré
	• ***areaSquarePanel*** afficher l'aire du carré
	• ***areaSquareReturn*** pour afficher l'aire du carré
	Affichez ensuite la somme du périmètre et de l'aire du carré.
	Tous les affichages sont à faire en console.
	Respectez l'affichage de l'exemple ci-dessous.
	`side: 5.0`
	`**********`
	`circumference of square: 20.0`
	`circumference of square: 20.0`
	`area of square: 25.0`
	`area of square: 25.0`
	`sum of circumference and area of square: 45.0`

C.21) RECTANGLE (APPEL DES METHODES - AVEC PARAMETRES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



Demander à l'enseignant de vous fournir le projet ***Rectangle (appel des methodes - avec parametres - ELEVES)***. Cette classe représente un rectangle. Complétez uniquement la méthode ***main*** sans réaliser des changements des autres parties.

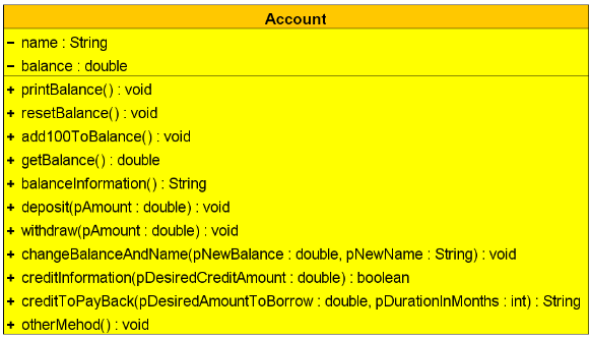
Les explications pour les attributs et méthodes se trouvent en dessous. Respectez l'UML au-dessus.

Partie	Description
Méthode : circumferenceRectangle Panel	**(à lire)** Cette méthode affiche le périmètre de la figure en console. ***pLength*** et ***pWidth*** indiquent la longueur et largeur du rectangle.
Méthode : circumferenceRectangle Return	**(à lire)** Cette méthode retourne le périmètre du rectangle. Les paramètres ***pLength*** et ***pWidth*** indiquent la longueur et la largeur du rectangle.
Méthode : areaRectanglePanel	**(à lire)** Cette méthode affiche dans la console l'aire du rectangle. Les paramètres ***pLength*** et ***pWidth*** indiquent la longueur et la largeur du rectangle.
Méthode : areaRectangleReturn	**(à lire) (à lire)** Cette méthode retourne l'aire du rectangle. Les paramètres ***pLength*** et ***pWidth*** indiquent la longueur et la largeur du rectangle.
Méthode : main	Les paramètres ***pLengthRect et pWidthRect*** indiquent la longueur et la largeur du rectangle. Faites appel aux méthodes : • ***circumferenceRectanglePanel*** pour afficher le périmètre du rectangle • ***circumferenceRectangleReturn*** pour afficher le périmètre du rectangle • ***areaRectanglePanel*** pour afficher l'aire du rectangle • ***areaRectangleReturn*** pour afficher l'aire du rectangle

	Affichez ensuite la somme du périmètre et de l'aire du rectangle.
	Tous les affichages sont à faire en console.
	Respectez l'affichage de l'exemple ci-dessous.
	`length: 5.0 width: 6.0`
	`**********`
	
	`circumferance of rectangle: 22.0`
	`circumference of rectangle: 22.0`
	
	`area of rectangle: 30.0`
	`area of rectangle: 30.0`
	
	`sum of circumferance and area of rectangle: 52.0`

C.22) BANK ACCOUNT (APPEL DES METHODES)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



L'enseignant doit vous fournir le projet *Bank account (élèves)*. Ajoutez uniquement du code source dans la méthode *otherMethod*. Analysez le code source des méthodes existantes, afin de comprendre leur fonctionnement.

Partie	Description
Méthode : otherMehod	Cette méthode fait appel aux autres méthodes de la classe. Faites appel à ces méthodes dans l'ordre indiqué ci-dessous : • ***printBalance*** • ***resetBalance*** • ***printBalance*** • ***add100ToBalance*** • ***printBalance*** • ***getBalance*** (et affichez la balance actuelle en console) • ***balanceInformation*** (et affichez le résultat retourné par cette méthode en console) • ***deposit*** avec ***125,5€*** à ajouter au compte bancaire • ***withdraw*** avec ***25€*** à retirer du compte bancaire • ***changeBalanceAndName*** avec une balance de ***1000€*** et un propriétaire de compte bancaire nommé ***Jempi*** • ***creditInformation*** avec un montant de ***802,5€*** à passer en paramètre (et affichez le résultat retourné par cette méthode en console) • ***creditInformation*** avec un montant de ***1500€*** à passer en paramètre (et affichez le résultat retourné par cette méthode en console) • ***creditToPayBack*** avec ***100€*** et ***3*** à passer en paramètres (et affichez le résultat retourné par cette méthode en console).

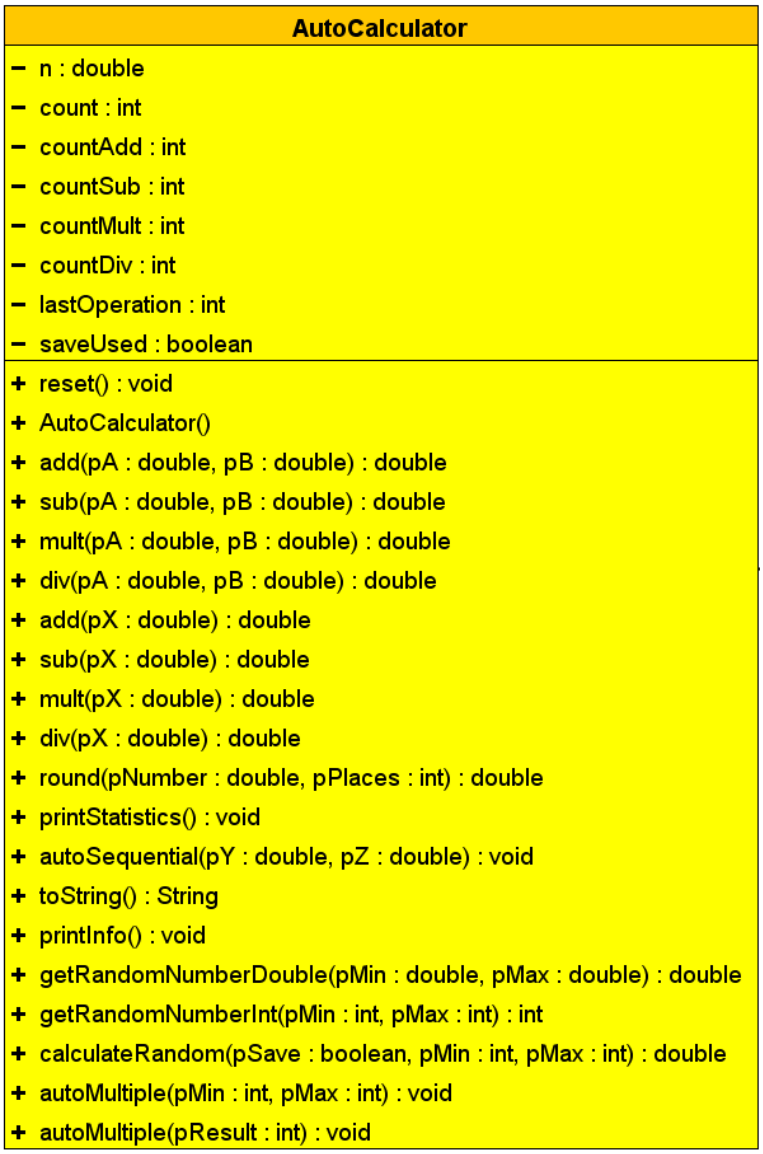
C.23) AUTOCALCULATOR – PHASE 1

X	Concepts de base		Boucles (FOR / WHILE)	X	Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

C.23.1) CLASSE « AUTOCALCULATOR »

La classe ***AutoCalculator*** représente une calculatrice avancée.

Ajoutez les attributs et méthodes de l'UML ci-dessous.



Partie	Description
Attribut : n	Cet attribut représente le résultat sauvegardé d'un calcul.
Attribut : count	Cet attribut représente le nombre total d'opérations réalisées.
Attribut : countAdd	Cet attribut représente le nombre d'additions faites.
Attribut : countSub	Cet attribut représente le nombre de soustractions faites.
Attribut : countMult	Cet attribut représente le nombre de multiplications faites.
Attribut : countDiv	Cet attribut représente le nombre de divisions faites.
Attribut : lastOperation	Cet attribut sauvegarde la dernière opération faite. • Cet attribut contient la valeur ***-1*** si aucune opération n'a été faite. • Dans les autres cas : ***0*** pour une addition, ***1*** pour une soustraction, ***2*** pour une multiplication, ***3*** pour une division.
Attribut : saveUsed	L'attribut ***saveUsed*** contient ***true*** si le résultat de la dernière opération a été sauvegardé et ***false*** dans le cas contraire.
Constructeur	Le constructeur appelle la méthode ***reset***.
Méthode : reset	La méthode ***reset*** réinitialise la calculatrice. Modifiez les attributs en affectant les valeurs appropriées.
Méthodes : add, sub, mult, div **avec 2 paramètres**	Les méthodes ***add***, ***sub***, ***mult*** et ***div*** réalisent respectivement les opérations suivantes : addition, soustraction, multiplication et division.Ces méthodes calculent et retournent un résultat. • Elles ne sauvegardent pas ce résultat dans l'attribut ***n***. • Les calculs sont réalisés avec les paramètres ***pA*** et ***pB***. • Le nombre total d'opérations faites est incrémenté. • Chaque méthode incrémente également son propre compteur d'opérations (l'attribut ***countAdd*** pour la méthode ***add***, etc). • L'attribut sauvegardant la dernière opération faite est actualisé. • L'attribut ***saveUsed*** est actualiser

Partie	Description
Méthodes : add, sub, mult, div **<u>avec 1 paramètre</u>**	Les méthodes ***add***, ***sub***, ***mult*** et ***div*** réalisent respectivement les opérations suivantes : addition, soustraction, multiplication et division. • Ces méthodes utilisent l'attribut ***n*** et le paramètre ***pX*** pour réaliser leur calcul. Par exemple, l'addition ajoute la valeur du paramètre ***pX*** à la valeur de ***n*** et sauvegarde le résultat dans ***n***. • N'oubliez pas d'actualiser l'attribut ***saveUsed***. Les autres attributs sont actualisés lors de l'appel des différentes méthodes. **<u>Pour chaque calcul vous devez faire appel aux méthodes, déjà programmées, avec les deux paramètres !!! Il est défendu d'utiliser des opérateurs arithmétiques (sauf pour la vérification).</u>**
Méthode : round	La méthode « round » fait un arrondi du nombre ***pNumber*** à ***pPlaces*** chiffres après la virgule. Utilisez la méthode ***Math.round*** qui arrondit à l'entier le plus proche, ainsi que la méthode ***Math.pow*** pour obtenir le nombre de décimales demandé.

Méthode : printStatistics	La méthode ***printStatistics*** utilise les différents attributs, ainsi que la méthode ***round***, pour afficher des statistiques dans la console. Voici une capture d'écran :
	`Opérations totales: 12`
	`Addition: 3 en pourcent: 25.0 %`
	`Soustraction: 1 en pourcent: 8.33 %`
	`Multiplication: 5 en pourcent: 41.67 %`
	`Division: 3 en pourcent: 25.0 %`

Méthode : auto Sequential

La méthode « autoSequential » réalise différents calculs.
<u>Cette méthode doit utiliser les méthodes que vous avez déjà programmées. Il est défendu d'utiliser des opérateurs arithmétiques (sauf pour la vérification).</u>

Avant de commencer :
Faites appel à la méthode « reset » et affichez en premier lieu une ligne indiquant les valeurs de pY et pZ.

Programmez les calculs suivants et faites attention aux types de données (int, double, …) pour obtenir un résultat adéquat, implémentez uniquement les méthodes avec <u>deux paramètres</u> (sans sauvegarde).

- 5 + 3 * 6
- 10 * 8 / 6
- (9 + 6) * 3
- 9 * pY + pZ
- ((10 / pZ) * (39 - pY)) / 10

<u>Exemple :</u>
5 + 3 * 6 add(5, mult(3,6))

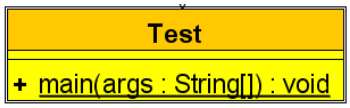
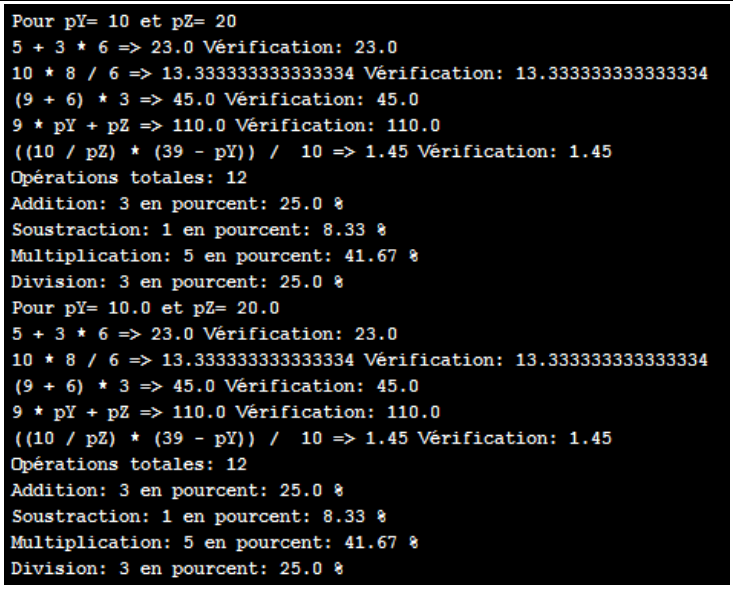
Après chaque opération, affichez le résultat en console.

- Pour vérifier votre résultat plus facilement, affichez-le aussi en utilisant des opérateurs mathématiques.
- Le format d'affichage est le suivant:
  <calcul> => **<résultat avec appel des méthodes avec 2 paramètres>** Vérification: **<résultat avec les opérateurs arithmétiques>**
  Les parties en gras sont à remplacer par leurs valeurs.

Faites appel à la méthode « printStatistics » à la fin.

```
Pour pY= 10.0 et pZ= 20.0
5 + 3 * 6 => 23.0 Vérification: 23.0
10 * 8 / 6 => 13.333333333333334 Vérification: 13.333333333333334
(9 + 6) * 3 => 45.0 Vérification: 45.0
9 * pY + pZ => 110.0 Vérification: 110.0
((10 / pZ) * (39 - pY)) /  10 => 1.45 Vérification: 1.45
Opérations totales: 12
Addition: 3 en pourcent: 25.0 %
Soustraction: 1 en pourcent: 8.33 %
Multiplication: 5 en pourcent: 41.67 %
Division: 3 en pourcent: 25.0 %
```

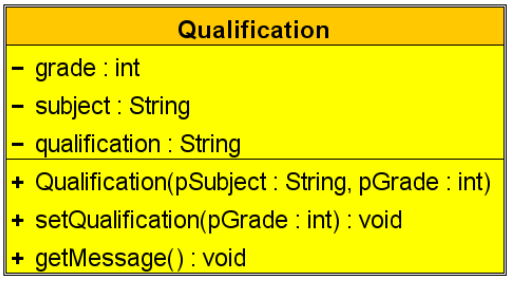
C.23.2) Classe « Test »

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un objet du type ***AutoCalculator***.
Méthodes à appeler / Tests à réaliser	Copiez le code de la méthode ***autoSequential*** de la classe ***AutoCalculator*** vers la méthode ***main***. Modifiez le code pour qu'il fonctionne depuis la classe ***Test***. Utilisez les valeurs 10 et 20 pour les valeurs de ***pY*** et ***pZ***. À la fin de la méthode, faites appel à la méthode ***autoSequential*** avec les valeurs 10 et 20.
Aperçu du résultat	

C.24) QUALIFICATION

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Développez une classe ***Qualification*** telle que décrite ci-dessous. Cette classe indique le résultat d'un élève dans un examen.



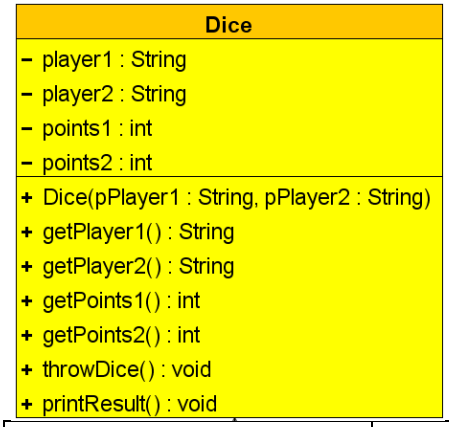
Partie	Description
Attribut : grade	Cet attribut indique la note d'examen de l'élève.
Attribut : subject	Cet attribut indique la matière de l'examen. Exemples : Informatique, Allemand, …
Attribut : qualification	Cet attribut contient de l'information textuelle indiquant si l'élève a passé ou échoué à l'examen.
Constructeur	Ajoutez un constructeur qui initialise les attributs ***grade*** et ***subject*** avec les valeurs des paramètres. L'attribut ***qualification*** est initialisé par la méthode ***setQualification***. Faites appel à cette méthode ***setQualification***.
Méthode : setQualification	Cette méthode actualise l'attribut ***qualification*** avec le texte **passed** ou **not passed**, selon que la note est inférieure à 30, ou supérieure ou égale à 30.

Méthode : getMessage	Cette méthode affiche en console la matière de l'examen, la note obtenue et l'information de passage/échec. Respectez l'affichage textuel des exemples ci-dessous.
	
	Exemple pour ***grade*** = 45 et ***subject*** = informatics
	`informatics --> 45 (passed)`
	Exemple pour ***grade*** = 25 et ***subject*** = informatics
	`informatics --> 25 (not passed)`

C.25) DICE

X	Concepts de base		Boucles (FOR / WHILE)	X	Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

C.25.1) CLASSE « DICE »

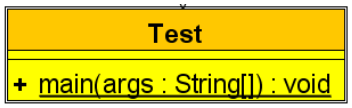
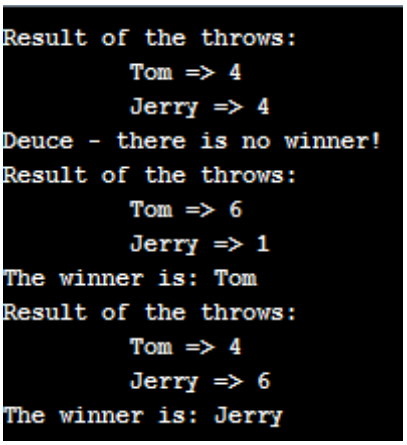


Développez une classe ***Dice***, qui simule un jeu où deux joueurs jouent un contre l'autre, en lançant chacun un dé à six faces. Le joueur avec le plus de points gagne le jeu.

Partie	Description
Attribut : player1	Cet attribut contient le nom du joueur 1.
Attribut : player2	Cet attribut contient le nom du joueur 2.
Attribut : points1	Cet attribut contient le résultat du jet (anglais : *throw*, allemand : *Wurf*) du joueur 1.
Attribut : points2	Cet attribut contient le résultat du jet du joueur 2.
Constructeur	Le constructeur initialise les attributs et fait appel à la méthode ***throwDice***.
Accesseurs	Créez les accesseurs indiqués.
Méthode : throwDice	Cette méthode simule un lancement de deux dés à six faces. ***points1*** et ***points2*** sont à initialiser sur des valeurs aléatoires entre les bornes [1 ; 6].

Méthode : printResult	Cette méthode affiche le résultat comme indiqué ci-dessous. Voici deux exemples ou player1 = Tom et player2 = Jerry
	`Result of the throws:`
	`Tom => 4`
	`Jerry => 4`
	`Result of the throws:`
	`Tom => 6`
	`Jerry => 1`
	`The winner is: Tom`

C.25.2) Classe « Test »

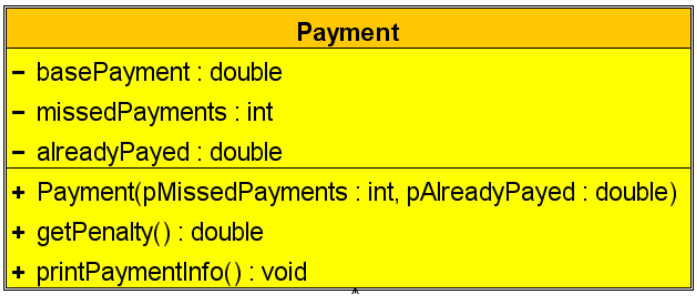
Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un nouveau jeu avec les joueurs ***Tom*** et ***Jerry***.
Méthodes à appeler / Tests à réaliser	Appelez trois fois les méthodes ***throwDice*** et ***printResult***.
Aperçu du résultat	 La capture est seulement un exemple, car des valeurs aléatoires sont utilisées dans l'exercice.

C.26) PAYMENT

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

C.26.1) CLASSE « PAYMENT »

Développez une classe ***Payment*** telle que décrite ci-dessous. Programmer des méthodes qui aident au calcul de payements d'une cotisation. Dans le cas, où le client n'aurait pas payé sa facture à la fin du mois, une amande est à payer.

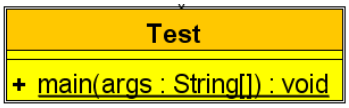


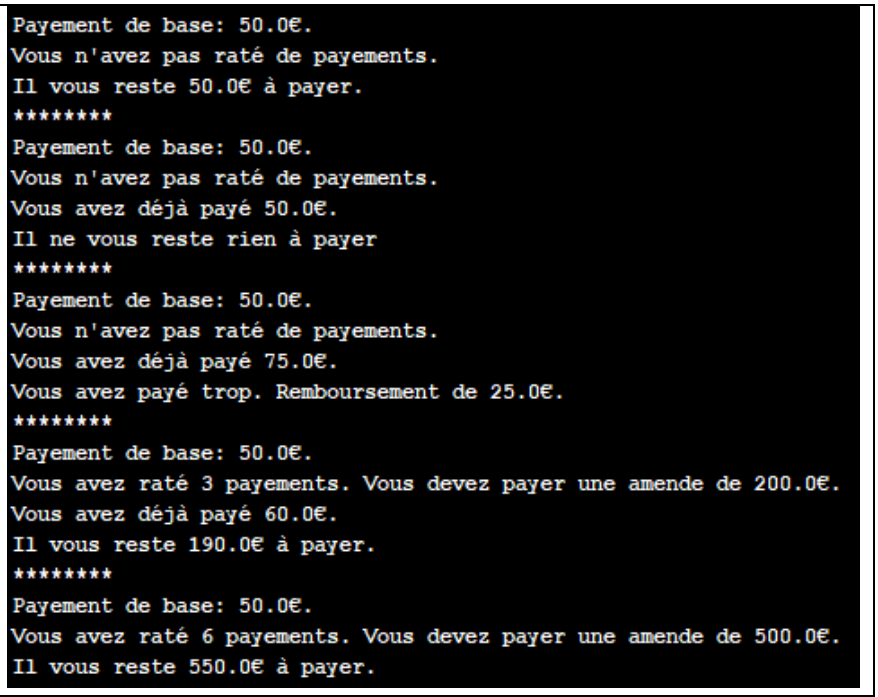
Partie	Description
Attribut : basePayment	Cet attribut indique le montant de la cotisation mensuel. Exemple : facture Tango de 50€ par mois.
Attribut : missedPayments	Cet attribut indique combien de payements ont été manqués dans le passé.
Attribut : alreadyPayed	Cet attribut indique combien d'euros ont déjà été payés pour la facture de ce mois ci et les amendes. (Il est possible, de payer une facture en plusieurs fois.)
Constructeur	Ajoutez un constructeur qui initialise les attributs avec les valeurs des paramètres. L'attribut ***basePayment*** est à initialiser à 50.
Méthode : getPenalty	Cette méthode calcule et retourne le montant d'une amende éventuelle à payer selon les payements manqués dans le passé (le mois actuel n'est pas pris en considération). • 1 payement manqué : la moitié de la cotisation • [2 ; 5] payements manqués : quatre fois la cotisation • 6 payements manqués ou plus : dix fois la cotisation • Il n'y a pas d'amende si aucun payement n'a été manqué
Méthode : printPaymentInfo	Cette méthode affiche en console toutes les informations nécessaires pour identifier le montant à payer. Le montant à payer se calcule en ajoutant la cotisation actuelle **et** l'amende éventuelle, en retranchant le montant déjà payé.

Partie	Description

	Diverses informations sont affichées en console. Basez-vous sur la capture d'écran et respectez l'affichage.
	
	Voici deux exemples :
	`Payement de base: 50.0€.`
	`Vous n'avez pas raté de payements.`
	`Il vous reste 50.0€ à payer.`
	
	`Payement de base: 50.0€.`
	`Vous avez raté 3 payements. Vous devez payer une amende de 200.0€.`
	`Vous avez déjà payé 60.0€.`
	`Il vous reste 190.0€ à payer.`

C.26.2) Classe « Test »

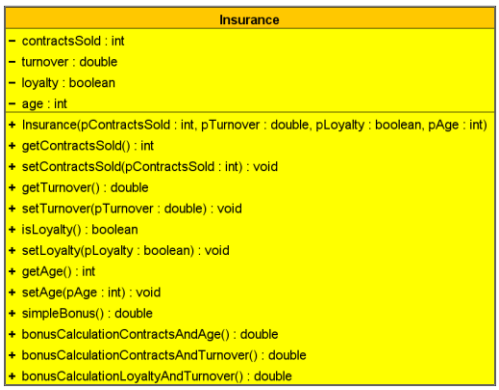
Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 6 objets de type ***Payment*** en utilisant les valeurs : • pMissedPayments = 0, pAlreadyPayed = 0 • pMissedPayments = 0, pAlreadyPayed = 50 • pMissedPayments = 0, pAlreadyPayed = 75 • pMissedPayments = 3, pAlreadyPayed = 60 • pMissedPayments = 6, pAlreadyPayed = 0
Méthodes à appeler / Tests à réaliser	Appelez la méthode ***printPaymentInfo*** pour chaque objet. Séparez les informations en console par le texte « ******** » pour chaque objet.

Aperçu du résultat	

C.27) INSURANCE (STRUCTURES IF+IF, IF+ELSE IF, ...)

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
X	IF imbriqué		Opérateurs logiques		

Développez une classe ***Insurance*** telle que décrite ci-dessous. Programmez des méthodes qui aident l'assurance à calculer, à la fin de l'année, la prime que les employés reçoivent.



Partie	Description
Attribut : contractsSold	Cet attribut indique le nombre de contrats que l'employé a vendus pendant une année.
Attribut : turnover	Cet attribut indique le chiffre d'affaires de l'employé.
Attribut : loyalty	Cet attribut indique si l'employé est un employé loyal envers l'entreprise (l'attribut prend la valeur ***true*** si l'employé est loyal).
Attribut : age	Cet attribut indique l'âge de l'employé.
Constructeur	Ajoutez un constructeur qui initialise les attributs avec les valeurs des paramètres.
Accesseurs & Manipulateurs	Créez les accesseurs et manipulateurs pour les attributs.
Méthode : simpleBonus	Cette méthode retourne le bonus que chaque employé reçoit à la fin de l'année. Pour le calcul du bonus, respectez les règles suivantes. Si l'employé : • a vendu plus de 100 contrats, son bonus est majoré de 500€. • a un chiffre d'affaires supérieur à 100000€, son bonus est majoré de 500€. • est un employé loyal envers l'entreprise, son bonus est majoré de 500€.

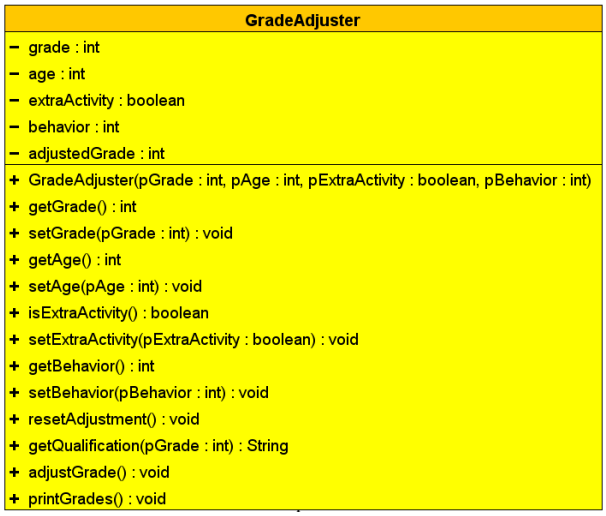
Partie	Description
Méthode : bonusCalculationContracts AndAge	Cette méthode retourne le bonus des employés, selon leurs contrats vendus et leur âge. La prime pour les contrats vendus et la prime d'âge sont additionnées. La prime pour les contrats vendus dépend du nombre de contrats vendus. Si l'employé a : • vendu au moins 10 contrats, son bonus est de 250€. • vendu au moins 50 contrats, son bonus est de 1000€. • vendu au moins 100 contrats, son bonus est de 5000€. La prime d'âge dépend de l'âge de l'employé. Si l'employé a : • **au** moins 40 ans, son bonus est de 250€. • au moins 50 ans, son bonus est de 500€.
Méthode : bonusCalculationContracts AndTurnover	Cette méthode retourne le bonus des employés, selon leurs contrats vendus et leur chiffre d'affaires réalisé. La prime pour les contrats vendus et la prime pour le chiffre d'affaires réalisé sont additionnées. Si l'employé a vendu au moins 100 contrats, son bonus est de 500€. Si l'employé a un chiffre d'affaires d'au moins 100000€, sa prime est de 1500€. Sinon sa prime est de 500€.
Méthode : bonusCalculationLoyalty AndTurnover	Cette méthode retourne le bonus des employés, selon leur loyauté et leur chiffre d'affaires réalisé. Si l'employé est loyal sa prime est de 500€. Et si l'employé a un chiffre d'affaires d'au moins 100000€, sa prime est majorée de 2500€. Sinon, si le chiffre d'affaires est inférieur à 100000€, sa prime est majorée de 500€ seulement.

C.28) GRADEADJUSTER

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
X	IF imbriqué		Opérateurs logiques		

C.28.1) CLASSE « GRADEADJUSTER »

Développez une classe ***GradeAdjuster*** telle que décrite ci-dessous. Programmez des méthodes qui aident un professeur à ajuster la note de fin d'année.

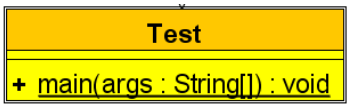
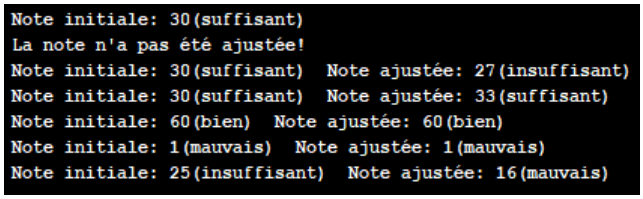


Partie	Description
Attribut : grade	Cet attribut indique la note initiale de l'élève.
Attribut : age	Cet attribut indique l'âge de l'élève.
Attribut : extraActivity	Cet attribut indique si l'élève a réalisé des activités supplémentaires dans l'enceinte de l'école (si c'est le cas, l'attribut a la valeur ***true***).
Attribut : behavior	Cet attribut indique le comportement (la conduite) de l'élève pendant l'année. Les valeurs ci-après sont utilisées pour le comportement : • 2 : très bon • 1 : bon • -1 : insuffisant • -2 : mauvais Toutes les autres valeurs sont à interpréter comme « rien à signaler ».
Attribut : adjustedGrade	Cet attribut indique la note ajustée de l'élève. Une valeur inférieure ou égale à 0 indique que la note n'a pas encore été ajustée.
Constructeur	Ajoutez un constructeur qui initialise les attributs avec les valeurs des paramètres. Appelez la méthode ***resetAdjustment*** pour initialiser l'attribut ***adjustedGrade***.

Accesseurs & Manipulateurs	Créez les accesseurs et manipulateurs pour les attributs.
Méthode : resetAdjustment	Cette méthode réinitialise l'attribut ***adjustedGrade*** à 0.
Méthode : getQualification	Cette méthode retourne l'appréciation de la note passée en paramètre : • Une note inférieure à 20 est considérée comme mauvaise. • Une note dans l'intervalle [20 ; 30[est considérée comme insuffisante. • Une note dans l'intervalle [30 ; 40[est considérée comme satisfaisante. • Une note supérieure à 40 est considérée comme bonne.
Méthode : adjustGrade	Si l'ajustement a déjà été effectué, cette méthode ne fait rien. L'ajustement peut se faire en fonction d'un pourcentage qui dépend de la note. Si la note est inférieure à 30, on se base sur un pourcentage de 25%. Sinon on utilise un pourcentage de 10%. • Si l'élève a réalisé des activités supplémentaires, on ajoute 5 points à la note. • Si le comportement est très bon, l'élève reçoit 2 points supplémentaires et la note est ajustée vers le haut d'après le pourcentage adéquat. • Si le comportement est bon, la note est ajustée vers le haut d'après le pourcentage adéquat. • Si le comportement est insuffisant, la note est ajustée vers le bas d'après le pourcentage adéquat. • Si le comportement est mauvais, l'élève perd 2 points (et il perd un point de plus s'il est déjà majeur), et la note est ajustée vers le bas d'après le pourcentage adéquat. • Finalement, si la note ajustée dépasse 60, elle est mise à 60, et si elle tombe en-dessous de 1, elle est mise à 1.

Méthode : printGrades	Cette méthode affiche la note initiale suivie de l'appréciation puis la note ajustée suivie de l'appréciation, le tout sur une seule ligne. Cependant si l'ajustement n'a pas encore été effectué, seule la première partie est affichée. Dans une deuxième ligne le texte « La note n'a pas été ajustée ! » apparaît.
	Voici deux exemples :
	`Note initiale: 30(suffisant)  Note ajustée: 27(insuffisant)`
	`Note initiale: 30(suffisant)`
	`La note n'a pas été ajustée!`

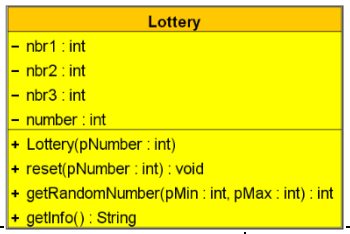
C.28.2) Classe « Test »

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez cinq objets du type ***GradeAdjuster*** : • Objet 1 - paramètres 30, 15, false, -1 • Objet 2 - paramètres 30, 15, false, 1 • Objet 3 - paramètres 60, 15, true, 2 • Objet 4 – paramètres 1, 15, false, -2 • Objet 5- paramètres 25, 18, false, -2
Méthodes à appeler / Tests à réaliser	Pour chaque objet faites appel aux méthodes ***printGrades, adjustGrade*** et de nouveau ***printGrades***.
Aperçu du résultat	

C.29) LOTERIE

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué	**X**	Opérateurs logiques		

Développez une classe ***Lottery*** telle que décrite ci-dessous.



Partie	Description
Attribut : nbr1	Cet attribut représente le premier nombre tiré au hasard.
Attribut : nbr2	Cet attribut représente le deuxième nombre tiré au hasard.
Attribut : nbr3	Cet attribut représente le troisième nombre tiré au hasard.
Attribut : number	Cet attribut mémorise le nombre indiqué par le joueur.
Constructeur	Le constructeur appelle la méthode ***reset***.
Méthode : getRandomNumber	Cette méthode retourne un nombre aléatoire entier compris entre les bornes indiquées comme paramètres.
Méthode : reset	Cette méthode réinitialise les attributs ***nbr1***, ***nbr2*** et ***nbr3*** avec des nombres aléatoires (compris respectivement dans les intervalles [1 ;25], [26 ;50] et [51 ;75]) en appelant la méthode ***getRandomNumber***. Trois nombres sont tirés au hasard (nombres aléatoires). Le premier ***nbr1*** contient des nombres de l'intervalle [1 ; 25], le deuxième ***nbr2*** contient des nombres de l'intervalle [26 ; 50] et le troisième ***nb3*** contient des nombres de l'intervalle [51 ; 75]. Ainsi, elle réinitialise l'attribut ***number*** avec la valeur de ***pNumber*** si ***pNumber*** est un nombre entre les bornes [1 ; 75]. Dans le cas contraire, ***number*** est initialisé sur une valeur aléatoire entre les bornes [1 ; 75].
Méthode : getInfo	Créez une méthode ***getInfo*** du type String qui retourne le texte *« Vous avez gagné ! »* si un numéro tiré au hasard (***nbr1***, ***nbr2*** ou ***nbr3***) correspond à ***number*** et *« Vous avez perdu ! »* dans le cas contraire.

C.30) ROULETTE

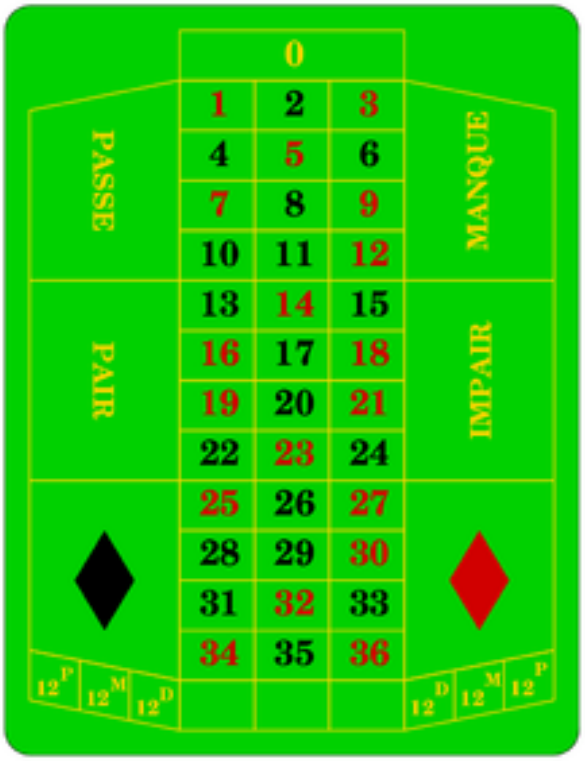
X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
X	IF imbriqué	X	Opérateurs logiques		

Développez une classe ***Roulette*** décrite par le diagramme de classes ci-dessous.

Roulette
− name : String − res : int
+ Roulette(pName : String) + spin() : void + printResult() : void

Cette classe permet de simuler la roulette d'un casino.
Dans un casino, au jeu de roulette, le croupier doit toujours annoncer les résultats aux joueurs d'une manière définie.
Le résultat est toujours un nombre entier de l'intervalle [0 ; 36]. Les nombres sont groupés en plusieurs catégories (voir image ci-dessous) :

- Le zéro (0) – le casino gagne
- Un nombre inférieur ou égal à 18 est annoncé comme ***manque***.
- Un nombre au-dessus de 18 est annoncé comme ***passe***.
- Certains nombres sont marqués en ***rouge*** d'autres sont marqués en ***noir*** sur le tapis de jeu.

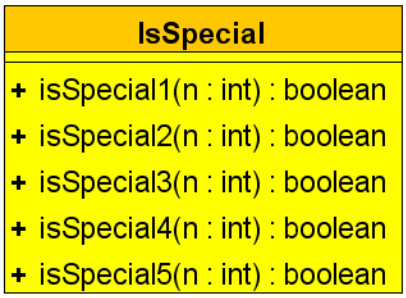


Partie	Description
Attribut : **name**	Cet attribut contient le nom de la roulette. Exemple : table roulette 1, R22, …
Attribut : **res**	Cet attribut contient le résultat d'un tour de roulette. Le résultat est généré aléatoirement par la méthode ***spin***.
Constructeur	Le constructeur initialise le nom.
Méthode : **spin**	Cette méthode génère un nombre aléatoire entier entre [0 ; 36] et le sauvegarde dans l'attribut ***res***.
Méthode : **printResult**	L'annonce du croupier (voir explications sur la page précédente) se fait avec cette méthode. Voici des exemples d'annonces de résultats, où le nom de la table correspond à R22 : • si le nombre est le 5, l'annonce est « R22 – le 5, rouge, impair et manque » • si le nombre est le 23, l'annonce est « R22 – le 23, rouge, impair et passe » • si le nombre est le 28, l'annonce est « R22 – le 28, noir, pair et passe » • si le nombre est le 0, l'annonce est « R22 – le 0, le casino gagne! » Le résultat est à afficher en console. Astuce : Pour déterminer si un nombre est rouge ou noir plusieurs conditions sont nécessaires. Pour déterminer ceci, vous devez regarder l'image sur la page précédente.

C.31) IsSpecial1

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué	X	Opérateurs logiques		

Développez une classe ***IsSpecial*** décrite par le diagramme de classes ci-dessous.



Partie	Description
Méthode : isSpecial1	Cette méthode vérifie si le nombre ***n*** est spécial ou pas. Elle considère qu'un nombre est spécial quand il est divisible par 8, sans être en même temps plus petit ou égal à 45 et sans avoir comme dernier chiffre le 7. Exemples de nombres spéciaux : -200, 192, 200. Exemples de nombres non spéciaux : -199, -1, 1, 2, 199, 0, 8. Cette méthode retourne ***true*** si ***n*** est spécial, et ***false*** dans le cas contraire.
Méthode : isSpecial2	Cette méthode vérifie si le nombre ***n*** est spécial ou pas. Elle considère qu'un nombre est spécial s'il est plus petit ou égal à -120, ou s'il a comme dernier chiffre le 0, ou s'il est divisible par 9. Exemples de nombres spéciaux : -200, -199, 0, 198, 200. Exemples de nombres non spéciaux : -2, -1, 1, 2, 199. Cette méthode retourne ***true*** si ***n*** es spécial, et ***false*** dans le cas contraire.
Méthode : isSpecial3	Cette méthode vérifie si le nombre ***n*** est spécial ou pas. Elle considère qu'un nombre est spécial s'il est plus petit que -90, ou s'il est positif, ou s'il a comme dernier chiffre le 8. Exemples de nombres spéciaux : -200, 0, 1, 199, 200. Exemples de nombres non spéciaux : -5, -4, -3, -2, -1. Cette méthode retourne ***true*** si ***n*** es spécial, et ***false*** dans le cas contraire.

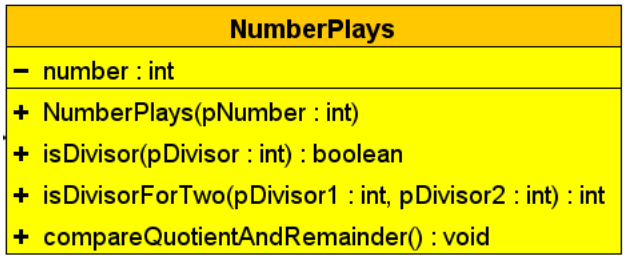
[1] Auteur : Reuland Roby

Partie	Description
Méthode : isSpecial4	Cette méthode vérifie si le nombre ***n*** est spécial ou pas. Elle considère qu'un nombre est spécial quand il est divisible par 3, sauf s'il est plus petit que -15 ou s'il a comme dernier chiffre le 6. Exemples de nombres spéciaux : -3, 0, 3, 195, 198. Exemples de nombres non spéciaux : -200, -1, 1, 199, 200. Cette méthode retourne ***true*** si ***n*** es spécial, et ***false*** dans le cas contraire.
Méthode : isSpecial5	Cette méthode vérifie si le nombre ***n*** est spécial ou pas. Elle considère qu'un nombre est spécial quand il est pair, ou s'il est plus petit ou égal à 90 sans avoir comme dernier chiffre le 0. Exemples de nombres spéciaux : -200, -1, 1, 200. Exemples de nombres non spéciaux : 191, 193, 195, 197, 199. Cette méthode retourne ***true*** si ***n*** es spécial, et ***false*** dans le cas contraire.

C.32) NUMBERPLAYS

X	Concepts de base		Boucles (FOR / WHILE)	X	Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué	X	Opérateurs logiques		

C.32.1) CLASSE « NUMBERPLAYS »



On appelle

- le résultat d'une division entière

a / b : le quotient (anglais : *quotient*)

- le reste d'une division entière

a / b : le reste (anglais : *remainder*)

C'est-à-dire : a = b * quotient + remainder

Par exemple avec a = 5 et b = 3 :

5 = 3 * 1 + 2

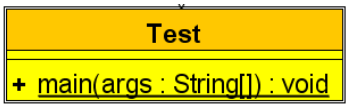
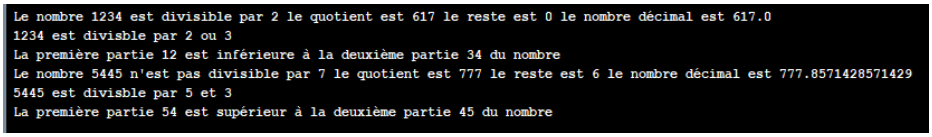
Voici la classe ***NumberPlays :***

Partie	Description
Attribut : number	Cet attribut contient un nombre indiqué par l'utilisateur.
Constructeur	Le constructeur initialise l'attribut ***number*** avec la valeur de ***pNumber***, mais seulement si cette valeur se situe dans l'intervalle [1000 ; 9999]. Sinon, l'attribut ***number*** est initialisé avec une valeur aléatoire de l'intervalle [1000 ; 9999].
Méthode : isDivisor	Cette méthode détermine si le paramètre ***pDivisor*** est un diviseur de ***number***. • Si c'est le cas, elle retourne ***true*** et affiche ce texte dans la console : Le nombre < valeur de number > est divisible par <valeur de pDivisor> le quotient est <valeur du quotient> le reste est <valeur du reste> le nombre décimal est <valeur décimale de la division> • Sinon, elle retourne ***false*** et affiche ce texte dans la console : *Le nombre < valeur de number > n'est pas divisible par <valeur de pDivisor> le quotient est <valeur du quotient> le reste est <valeur du reste> le nombre décimal est <valeur décimale de la division>*

Méthode : isDivisorForTwo	Cette méthode détermine si ***pDivisor1*** et ***pDivisor2*** sont des diviseurs de ***number***. Il y a trois cas possibles : • S'ils sont tous deux diviseurs, alors la méthode retourne ***1*** et affiche ce texte dans la console : *<valeur de number> est divisible par <valeur de pDivisor1> et <valeur de pDivisor2>* • Si l'un des deux est diviseur, alors la méthode retourne ***0*** et affiche ce texte dans la console : *<valeur de number> est divisible par <valeur de pDivisor1> ou <valeur de pDivisor2>* • Si aucun des deux n'est diviseur, alors la méthode retourne ***-1*** et affiche ce texte dans la console : *<valeur de number> n'est pas divisible par <valeur de pDivisor1> ou <valeur de pDivisor2>*

Méthode : compareQuotientAndRemainder	Utilisez une division entière et l'opérateur modulo pour "couper" le nombre en deux. Choisissez un diviseur adéquat.
	Par exemple pour 1234, on peut avoir :
	• La première partie (le quotient) est 12.
	• La deuxième partie (le reste) est 34.
	Cette méthode compare la première partie avec la deuxième partie du nombre : lui est-il inférieur, supérieur ou égal ?
	Dépendamment du résultat (inférieur, supérieur ou égal), elle affiche dans la console : *La première partie <valeur du quotient> est <inférieur/supérieur/égal> à la deuxième partie <valeur du reste> du nombre*
	Exemples d'affichage :
	`La première partie 12 est inférieure à la deuxième partie 34 du nombre`
	`La première partie 34 est supérieur à la deuxième partie 12 du nombre`
	`La première partie 12 est égale à la deuxième partie 12 du nombre`

C.32.2) Classe « Test »

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 2 objets de type ***NumberPlays*** en utilisant les valeurs : • 1234 • 5445
Méthodes à appeler / Tests à réaliser	Pour le premier objet, implémentez les étapes suivantes : • Faites appel à la méthode ***isDivisor*** avec le nombre 2. • Faites appel à la méthode ***isDivisorForTwo*** avec les nombres 2 et 3 • Faites appel à la méthode ***compareQuotientAndRemainder***. Pour le deuxième objet, implémentez les étapes suivantes : • Faites appel à la méthode ***isDivisor*** avec le nombre 7. • Faites appel à la méthode ***isDivisorForTwo*** avec les nombres 5 et 3. • Faites appel à la méthode ***compareQuotientAndRemainder***.
Aperçu du résultat	

C.33) AUTOCALCULATOR - PHASE 2

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
X	IF imbriqué		Opérateurs logiques		

Reprenez la dernière phase de l'exercice « AutoCalculator ».

Ajoutez les 5 méthodes décrites ci-dessous.

```
+ toString() : String
+ printInfo() : void
+ getRandomNumberDouble(pMin : double, pMax : double) : double
+ getRandomNumberInt(pMin : int, pMax : int) : int
+ calculateRandom(pSave : boolean, pMin : int, pMax : int) : double
```

Partie	Description
Méthode : toString	Cette méthode retourne une représentation textuelle de la calculatrice. - Si la valeur de l'attribut ***lastOperation*** est ***-1***, retournez : « Aucune opération n'a été faite ». - Sinon :la dernière opération est déterminée selon l'attribut ***lastOperation*** : ○ 0 : « addition », ○ 1 : « soustraction », ○ 2 : « multiplication », ○ 3 : « division ». • L'attribut ***saveUsed*** détermine le texte à afficher ensuite : ○ Si ***saveUsed*** est ***true***, affichez : « a été sauvegardé » ○ Si ***saveUsed*** est ***false***, affichez : « n'a pas été sauvegardé » Le résultat à retourner par la méthode ***toString*** a la forme suivante : <**Nombre d'opérations**>: Opération <**nom de l'opération**> le résultat <**texte indiquant si le résultat a été sauvegardé**>. Les parties en gras sont à remplacer par leurs valeurs.

Partie	Description

	Par exemple :
	`269: Opération division le résultat n'a pas été sauvegardé.`
	`269: Opération division le résultat a été sauvegardé.`

Méthode : printInfo	Cette méthode affiche dans la console le texte retourné par la méthode ***toString*** et appelle la méthode ***printStatistics***.
Méthode : getRandomNumberDouble	Cette méthode calcule et retourne un nombre aléatoire réel entre ***pMin*** et ***pMax***.
Méthode : getRandomNumberInt	Cette méthode calcule et retourne un nombre aléatoire entier entre ***pMin*** et ***pMax***.
Méthode : calculateRandom	Cette méthode réalise des calculs de façon aléatoire. • Elle fait une opération aléatoire (+, -, *, /). ○ Générez un nombre aléatoire entre 0 et 3 (faites appel à la méthode appropriée). Faites l'opération correspondant au nombre généré (0 : addition, 1 : soustraction, 2 : multiplication, 3 : division). • Si le paramètre ***pSave*** est ***true***, utilisez les opérations avec un paramètre, par exemple *add(double pX)*, mais si le paramètre ***pSave*** est ***false***, utilisez les opérations avec 2 paramètres, par exemple *add(double pA, double pB).* • Les nombres utilisés pour les calculs sont des nombres aléatoires entre ***pMin*** et ***pMax***. • Sauvegardez le résultat dans une variable locale et retournez-le à la fin de la méthode.

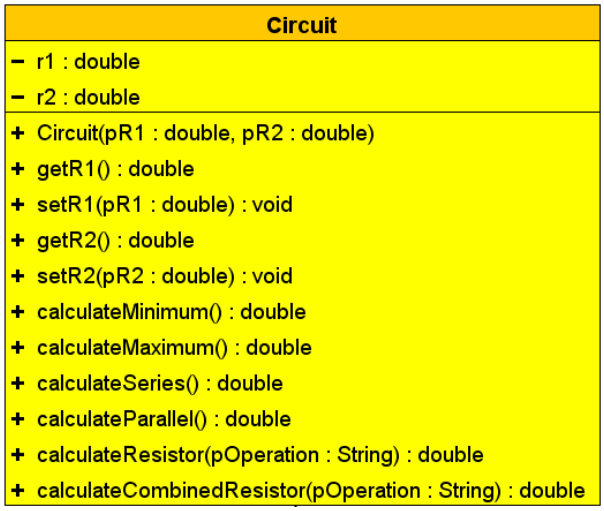
C.34) CIRCUIT

X	Concepts de base		Boucles (FOR / WHILE)	X	Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
X	IF imbriqué	X	Opérateurs logiques		

C.34.1) CLASSE « CIRCUIT »

Développez une classe ***Circuit*** telle que décrite ci-dessous.

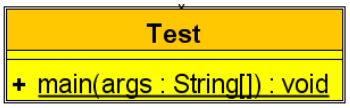
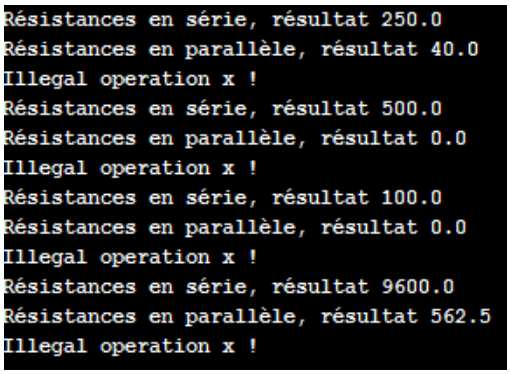
Cette classe permet de manipuler des circuits électroniques – principalement des résistances ***r1*** et ***r2***.



Partie	Description
Attribut : r1	Cet attribut contient la valeur de la résistance 1.
Attribut : r2	Cet attribut contient la valeur de la résistance 2.
Accesseurs & Manipulateurs	Créez les accesseurs indiqués. Créez les manipulateurs indiqués. Les manipulateurs veillent à ce que les valeurs de ***r1*** et ***r2*** soient toujours positives. Pour ceci, utilisez une méthode de la classe Math.
Constructeur	Le constructeur appelle les manipulateurs de ***r1*** et ***r2***.
Méthode : calculateMinimum	Cette méthode calcule le minimum de ***r1*** et ***r2***. Utilisez une méthode de la classe Math.
Méthode : calculateMaximum	Cette méthode calcule le maximum de ***r1*** et ***r2***. Utilisez une méthode de la classe Math.
Méthode : calculateSeries	Cette méthode retourne la valeur des résistances, si celles-ci sont branchées en série. La formule générale est : $R_S = R_1 + R_2 + \ldots + R_n$

Partie	Description
Méthode : calculateParallel	Cette méthode retourne la valeur des résistances, si celles-ci sont branchées en parallèle. La formule générale est : $R_p = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}+\ldots+\frac{1}{R_n}}$ Si une des résistances nulles, le résultat est à mettre à zéro.
Méthode : calculateResistor	Cette méthode permet de calculer la résistance totale d'un circuit en parallèle ou en série. Pour cela, le paramètre ***pOperation*** contient le type du branchement souhaité :« s » ou « S » pour un branchement en série, auquel cas le message suivant est affiché en console : *Résistances en série, résultat <valeur du résultat>.* Faites appel à la méthode ***calculateSeries*** • « ***p*** » ou « ***P*** » pour un branchement en parallèle, auquel cas le message suivant est affiché en console : *Résistances en parallèle, résultat <valeur du résultat>.* Faites appel à la méthode ***calculateParallel*** • Toute autre valeur (par exemple « ***z*** ») doit afficher un message d'erreur dans la console avec le texte : *Illegal operation <valeur de pOperation> !* . Dans ce cas il faut retourner la valeur ***-1***. Remarque : pour vérifier si un objet du type String contient un certain texte il faut utiliser la méthode prédéfinie ***equals***. Exemple : pOperation.equals("hello")
Méthode : calculateCombinedResistor	Cette méthode fait la même chose que la méthode ***calculateResistor***, mais ne fait plus appel aux méthodes ***calculateSeries*** et ***calculateParallel***. Il faut alors combiner ces méthodes.

C.34.2) Classe « Test »

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 4 objets de type ***Circuit*** en utilisant les noms et valeurs suivants : • c1 : 50 et 200 • c2 : 500 et 0 • c3 : 0 et 100 • c4 : -600 et -9000
Méthodes à appeler / Tests à réaliser	Pour chaque circuit, faites appel à la méthode ***calculateResistor*** trois fois, avec les valeurs « s », « p » et « x ».
Aperçu du résultat	

C.35) CONTRAVENTIONS

X	Concepts de base		Boucles (FOR / WHILE)	X	Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
X	IF imbriqué	X	Opérateurs logiques		

C.35.1) CLASSE « TRAFFICFINES »

Le tableau[2] ci-dessous reprend tous les seuils relatifs aux excès de vitesse et leur catégorisation par gravité.

simple fine

serious fine

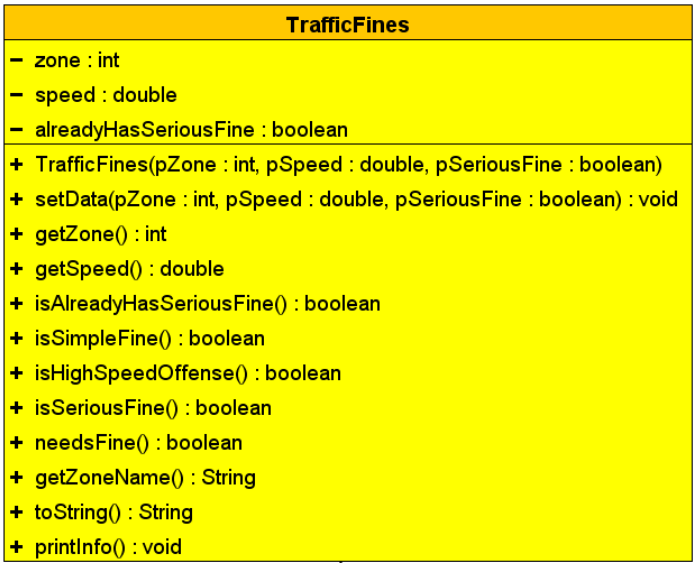
high speed offense

situation de référence	Vitesse maximale autorisée	contravention simple	contravention grave	délit de grande vitesse (seulement en cas de récidive suite à une 1ère contravention grave)	
zone piétonne / zone résidentielle	20 km/h	21 - 35 km/h	> 35 km/h	≥ 40 km/h	Zone 0
zone à 30 km/h	30 km/h	31-45 km/h	> 45 km/h	≥ 50 km/h	Zone 1
en agglomération	50 km/h	51-65 km/h	> 65 km/h	> 75 km/h	Zone 2
en rase campagne:	90 km/h	91-110 km/h	> 110 km/h	> 135 km/h	Zone 3
sur autoroute - temps sec:	130 km/h	131 - 155 km/h	> 155 km/h	> 195 km/h	Zone 4
sur autoroute - pluie:	110 km/h	111 - 135 km/h	> 135 km/h	> 165 km/h	Zone 5

[2] Les informations sont prises du site: http://www.mt.public.lu/transports/circulation/permis_points/delit/index.html

Créez la classe ***TrafficFines***.

Cette classe est une modélisation des informations relevées par un policier lors d'un contrôle de vitesse.

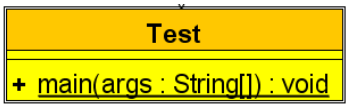


Partie	Description
Attribut : **zone**	Cet attribut décrit la situation de référence. Un nombre entier différent est utilisé pour chaque situation : • zone piétonne → 0 • zone à 30 → 1 • en agglomération → 2 • en rase campagne → 3 • sur autoroute → 4 • sur autoroute mouillée → 5
Attribut : **speed**	Cet attribut décrit la vitesse roulée en km/h.
Attribut : **alreadyHasSeriousFine**	Cet attribut est ***true*** si le conducteur avait déjà reçu une contravention grave.
Accesseurs & Manipulateur	Créez les accesseurs indiqués. Créez le manipulateur indiqué.
Constructeur	Le constructeur fait appel à la méthode ***setData*** pour initialiser les attributs.
Méthode : **isSimpleFine**	Cette méthode retourne ***true*** si le conducteur reçoit une **contravention simple**. Utilisez le tableau sur la première page pour vous guider. Vous devez tester la zone (situation de référence) **et en même temps** la limite de vitesse pour cette zone.

Partie	Description
Méthode : isHighSpeedOffense	Cette méthode retourne ***true*** si le conducteur reçoit un **délit de grande vitesse**. Utilisez le tableau sur la première page pour vous guider. Vous devez tester la zone (situation de référence) **et en même temps** la limite de vitesse pour cette zone. **Attention :** On reçoit seulement un **délit de grande vitesse** si on a déjà eu une **contravention grave**. Ce fait est mémorisé dans l'attribut ***alreadyHasSeriousFine*****.**
Méthode : isSeriousFine	Cette méthode retourne ***true*** si le conducteur reçoit une **contravention grave**. Utilisez le tableau sur la première page pour vous guider. Vous devez tester la zone (situation de référence) **et en même temps** la limite de vitesse pour cette zone. **Attention :** Si on reçoit un délit de grande vitesse, on ne reçoit pas une contravention grave.
Méthode : needsFine	Cette méthode retourne ***true*** si un délit a été constaté. Faites appel aux méthodes précédentes.
Méthode : getZoneName	Cette méthode vous retourne le nom de la zone. • 0 → « zone piétonne » • 1 → « zone à 30 » • 2 → « en agglomération » • 3 → « en rase campagne » • 4 → « sur autoroute » • 5 → « sur autoroute mouillée »

Partie	Description
Méthode : toString	Cette méthode vous retourne une représentation textuelle. Dans le cas d'un délit, le format est : Vous avez roulé à <**vitesse**> km/h dans une zone <**nom de la zone**> et vous avez reçu <**nom du délit**>. Le nom du délit peut-être : • une contravention simple • une contravention grave • un délit de grande vitesse Si vous n'avez pas roulé trop vite, le format est : Vous avez roulé à <vitesse> km/h dans une zone <nom de la zone>et vous n'avez pas dépassé les limites. Les parties en gras sont à remplacer par les valeurs correctes. Faites appel aux méthodes précédentes !!
Méthode : printInfo	Cette méthode appelle la méthode ***toString*** et affiche le texte dans la console.

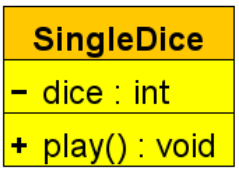
C.35.2) Classe « Test »

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez un nouvel objet du type ***TrafficFines***. Les paramètres pour le constructeur sont : zone : 0, vitesse : 30, délit : false.
Méthodes à appeler / Tests à réaliser	Faites appel à la méthode ***printInfo*** sur cet objet. Faites appel à la méthode ***setData*** et puis la méthode ***printInfo***. Les paramètres à utiliser sont : • zone : 1, vitesse : 30, délit : true • zone : 2, vitesse : 80, délit : true • zone : 3, vitesse : 110, délit : true • zone : 4, vitesse : 180, délit : false • zone : 5, vitesse : 130, délit : true
Aperçu du résultat	Vous avez roulé à 30.0 km/h dans une zone < zone piétonne > et vous avez reçu une contravention simple. Vous avez roulé à 30.0 km/h dans une zone < zone à 30 > et vous n'avez pas dépassé les limites. Vous avez roulé à 80.0 km/h dans une zone < en agglomération > et vous avez reçu un délit de grande vitesse. Vous avez roulé à 110.0 km/h dans une zone < en rase campagne > et vous avez reçu une contravention simple. Vous avez roulé à 180.0 km/h dans une zone < sur autoroute > et vous avez reçu une contravention grave. Vous avez roulé à 130.0 km/h dans une zone < sur autoroute pluie > et vous avez reçu une contravention simple.

C.36) SINGLE DICE

X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Créez la classe ***SingleDice***, qui représente un dé (allemand : Würfel, anglais : dice) de 12 faces. Chaque face présente un chiffre différent, allant de 1 à 12.



Partie	Description
Attribut : **dice**	Cet attribut représente la valeur du dé, qui se situe entre les bornes [1 ; 12].

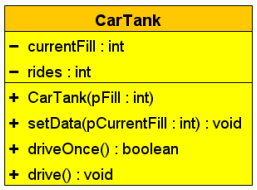
Méthode :
play
	Étape 1 :
	Cette méthode simule un lancement du dé où un nombre aléatoire compris dans l'intervalle [1 ; 12] est généré. Ce nombre correspond à la face sur laquelle le dé est tombé lors du lancement.
	Affichez le texte « Start … » avant le lancement et le texte « Jeu terminé ! » après le lancement.
	La valeur du dé est à afficher en console. Si la valeur lancée correspond à 10, le texte « gagné ! » suit la valeur du dé.
	Exemples d'affichage en console :
	`Start ...`
	`6`
	`Jeu terminé !`
	`Start ...`
	`10 gagné!`
	`Jeu terminé !`
	
	Étape 2 :
	Adaptez votre code source, afin que la méthode s'arrête si la valeur du dé correspond à 10, plusieurs lancements du dé possibles.

Partie	Description

	Exemple d'affichage en console :
	`Start ...`
	`9`
	`9`
	`8`
	`10 gagné!`
	`Jeu terminé !`
	
	Étape 3 :
	Adaptez votre code source, afin que la méthode affiche le nombre de jets réalisés jusqu'à ce que le nombre 10 ait été lancé.
	Exemple d'affichage en console et résultat retourné :
	`Start ...`
	`11`
	`9`
	`12`
	`8`
	`10 gagné!`
	`Nombre de jets: 5`
	`Jeu terminé !`

C.37) RESERVOIR D'UNE VOITURE

X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		



Créez la classe ***CarTank***.

Partie	Description
Attribut : currentFill	Cet attribut représente le volume d'essence en litres dans le réservoir de la voiture.
Attribut : rides	Cet attribut indique le nombre d'excursions qu'on a déjà faites.
Constructeur	Le constructeur appelle la méthode ***setData***.
Méthode : setData	Cette méthode initialise l'attribut ***currentFill*** à la valeur passée en paramètre. L'attribut ***rides*** est initialisé à ***0***.
Méthode : driveOnce	Cette méthode est utilisée pour organiser une excursion. • Pendant une excursion, on utilise un nombre de litres d'essence aléatoire. Le minimum est 1 litre et le maximum est le volume d'essence qui reste dans le réservoir. • Si l'attribut ***currentFill*** devient inférieur à 0, il est à mettre à 0. • La méthode affiche dans la console des informations sur l'excursion. Le format est : pendant l'excursion numéro **<nombre d'excursions>** la voiture a utilisé **<litres utilisés>** litres et il reste **<litres restants>** litres dans le réservoir. Les parties en gras sont à remplacer par des valeurs concrètes. • La méthode retourne ***true*** s'il reste encore de l'essence dans le réservoir.

Méthode : drive	Cette méthode appelle la méthode ***driveOnce*** tant qu'il reste de l'essence dans le réservoir. Quand le réservoir est vide, un message est affiché dans la console. Voici une capture d'écran de la console, pour **un** appel de la méthode ***drive***.
	`Pendant l'excursion numéro 1 la voiture a utilisé 14 litres et il reste 86 litres dans le réservoir.`
	`Pendant l'excursion numéro 2 la voiture a utilisé 49 litres et il reste 37 litres dans le réservoir.`
	`Pendant l'excursion numéro 3 la voiture a utilisé 23 litres et il reste 14 litres dans le réservoir.`
	`Pendant l'excursion numéro 4 la voiture a utilisé 2 litres et il reste 12 litres dans le réservoir.`
	`Pendant l'excursion numéro 5 la voiture a utilisé 11 litres et il reste 1 litres dans le réservoir.`
	`Pendant l'excursion numéro 6 la voiture a utilisé 1 litres et il reste 0 litres dans le réservoir.`
	`Après 6 excursions, le réservoir de la voiture est vide.`

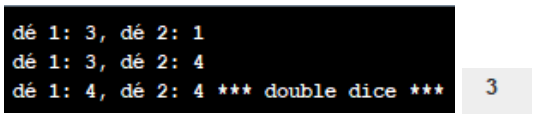
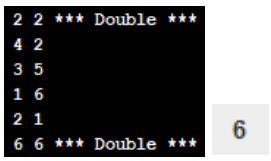
C.38) TWO DICE

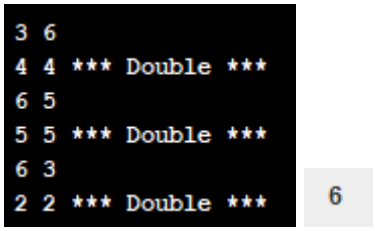
X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Créez la classe ***TwoDice***. Les deux dés (allemand : Würfel, anglais : dice) disposent chacun d'entre eux de six faces. Chaque face présente un chiffre différent, allant de 1 à 6.

TwoDice
- dice1 : int - dice2 : int
+ randomInt(pMin : int, pMax : int) : int + throwDiceUntilDouble() : int + throwDiceUntilNTimesSame(pN : int) : int

Partie	Description
Attribut : **dice1**	Cet attribut représente la valeur du premier dé.
Attribut : **dice2**	Cet attribut représente la valeur du deuxième dé.
Méthode : **randomInt**	Cette méthode génère un nombre aléatoire entier, compris dans l'intervalle [pMin ; pMax].
Méthode : **throwDiceUntilDouble**	Étape 1 : Cette méthode génère des lancements des deux dés simultanément (allemand : gleichzeitig). Chacun des deux dés (dice1 et dice2) est à initialiser avec un nombre aléatoire entier compris dans l'intervalle [1 ; 6]. La méthode arrête les lancements si la valeur des deux dés est identique. La valeur des deux dés (pour chaque lancement) est à afficher en console. La méthode retourne comme résultat le nombre de jets qui ont été effectués. Exemple d'affichage en console et résultat retourné : dé 1: 2, dé 2: 1 dé 1: 6, dé 2: 2 dé 1: 6, dé 2: 5 dé 1: 6, dé 2: 3 dé 1: 2, dé 2: 3 dé 1: 2, dé 2: 2 6

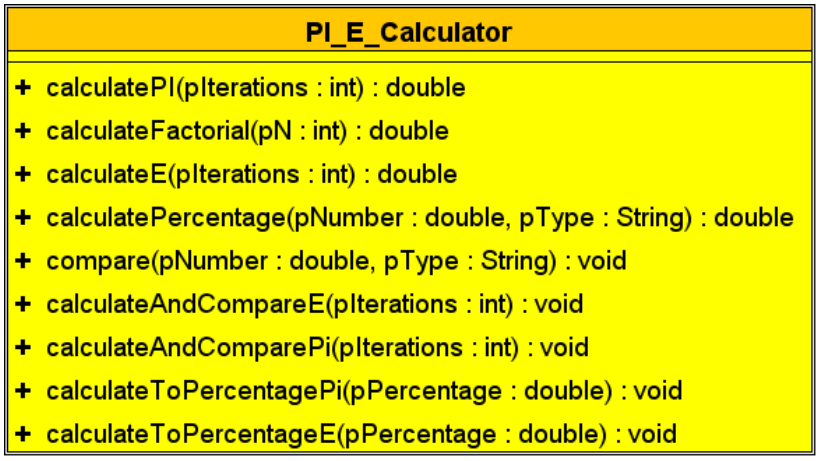
<table>
<tr><th>Partie</th><th>Description</th></tr>
<tr><td></td><td>Étape 2 :
Affichez le texte « *** double dice *** » après le lancement où les deux dés ont la même valeur.
Exemple d'affichage en console et résultat retourné :
 
</td></tr>
<tr><td>Méthode :
throwDiceUntilNTimesSame</td><td>Étape 1 : sans paramètre throwDiceUntilNTimesSame
Cette méthode génère des lancements des deux dés simultanément. Chacun des deux dés (dice1 et dice2) est à initialiser avec un nombre aléatoire entier compris dans l'intervalle [1 ; 6]. La méthode arrête les lancements si la valeur des deux dés est deux fois identique. La valeur des deux dés (pour chaque lancement) est à afficher en console.
La méthode retourne comme résultat le nombre de jets qui ont été effectués.
Affichez le texte « *** double dice *** » après les lancements où les deux dés ont la même valeur.
Exemple d'affichage en console et résultat retourné :
 

Étape 2 : avec paramètre pN
La méthode arrête les lancements si la valeur des deux dés est pN fois identique.
Exemple d'affichage en console et résultat retourné pour pN = 3 :
 
</td></tr>
</table>

C.39) CALCUL DE E ET PI

X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Créez la classe ***Pi_E_Calculator*** telle que décrite ci-dessous.



Dans le tableau suivant, le terme **itération** est utilisé. Une itération est un tour de boucle.

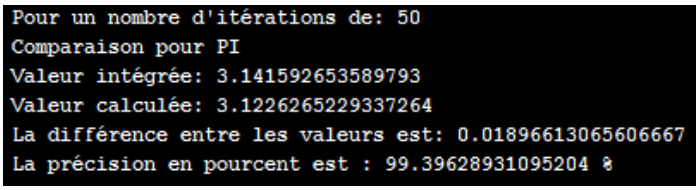
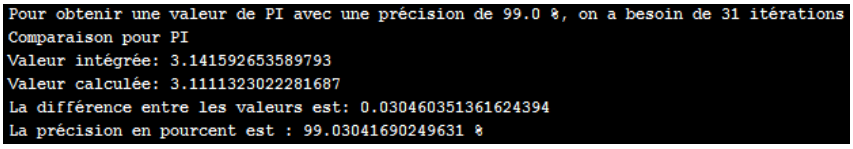
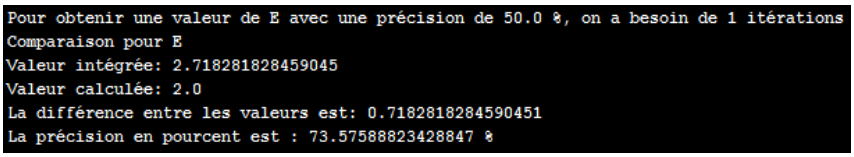
Exemple : Si on a 4 itérations, alors la boucle fait 4 tours.

Partie	Description
Méthode : calculatePI	Cette méthode calcule et retourne une valeur de PI approchée. Suivant Leonard Euler, la valeur de PI peut être calculée avec la formule : $\sqrt{6 * (\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots + \frac{1}{n^2})}$ Cette formule utilise un certain nombre d'itérations. La précision du calcul accroît avec le nombre d'itérations. Le nombre d'itérations est ***n***. <u>Par exemple :</u> • Avec n = 2 on obtient la formule : $\sqrt{6 * (\frac{1}{1^2} + \frac{1}{2^2})}$ • Avec n = 4 on obtient la formule : $\sqrt{6 * (\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2})}$

Méthode : calculateFactorial	Cette méthode calcule le produit des n premiers nombres entiers positifs (non nuls). En mathématiques, le produit des n premiers nombres est appelé factoriel de n ou de manière abrégée n! Par définition : 0! = 1. Par exemple : pour n=4, le résultat est 1 * 2 * 3 * 4 = 24.
Méthode : calculateE	Cette méthode calcule et retourne une valeur de E approchée. La valeur de E peut être calculée avec la formule : Cette formule utilise un certain nombre d'itérations. La précision du calcul s'accroît avec le nombre d'itérations. Le nombre d'itérations est ***n***. Faites appel à la méthode ***calculateFactorial***. Par exemple : • Avec n = 2 on obtient la formule : $1 + \frac{1}{1!} + \frac{1}{2!}$ • Avec n = 4 on obtient la formule : $1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!}$
Méthode : calculate Percentage	Cette méthode prend comme paramètre une valeur ***pNumber*** approchée de PI ou de E et retourne un nombre correspondant au pourcentage de précision de ***pNumber*** par rapport à la valeur contenue dans la classe « Math ». Le deuxième paramètre ***pType*** indique si la valeur de ***pNumber*** est une valeur de PI ou de E. C'est-à-dire : • si la valeur de ***pType*** est « pi », il faut comparer la valeur de ***pType*** à Math.PI • si la valeur de ***pType*** est « e », il faut comparer la valeur de ***pNumber*** à Math.E Par exemple : • La valeur approchée ***pNumber*** est : 3.1320765318091053 • La valeur de ***pType*** est : « pi » • Le pourcentage de précision (le résultat de la méthode) est: 99.69709243590782

Méthode : compare	Cette méthode prend une valeur approchée de PI ou de E comme paramètre et fait une comparaison entre cette valeur approchée et la valeur intégrée de la classe « Math ». Le paramètre ***pType*** indique si la valeur ***pNumber*** est une valeur approchée de PI ou de E. Faites appel à la méthode ***calculatePercentage*** pour le calcul de la précision.
	La méthode ***compare*** affiche la comparaison dans la console. Voici deux exemples :
	`Comparaison pour PI`
	`Valeur intégrée: 3.141592653589793`
	`Valeur calculée: 3.139684123138722`
	`La différence entre les valeurs est: 0.0019085304510713108`
	`La précision en pourcent est : 99.93924958893412 %`
	`Comparaison pour E`
	`Valeur intégrée: 2.718281828459045`
	`Valeur calculée: 2.6666666666666665`
	`La différence entre les valeurs est: 0.05161516179237857`
	`La précision en pourcent est : 98.10118431238462 %`

Méthode : calculateAndCompareE	Cette méthode calcule la valeur de E en utilisant le nombre d'itérations passé en paramètre, puis elle compare cette valeur calculée avec la valeur intégrée.
	Faites appel aux méthodes ***calculateE*** et ***compare***.
	Avant le texte de la méthode ***compare***, affichez les deux lignes suivantes :
	Pour un nombre d'itérations de : <**nombre d'itérations**>
	Comparaison de E
	Voici un exemple pour 3 itérations :
	`Pour un nombre d'itérations de: 3`
	`Comparaison pour E`
	`Valeur intégrée: 2.718281828459045`
	`Valeur calculée: 2.6666666666666665`
	`La différence entre les valeurs est: 0.05161516179237857`
	`La précision en pourcent est : 98.10118431238462 %`

<table>
<tr><td>Méthode : calculateAndComparePi</td><td>Cette méthode calcule et compare la valeur de PI pour un certain nombre d'itérations.
Faites appel aux méthodes calculatePi et compare.
Avant le texte de la méthode compare, affichez les deux lignes suivantes :
Pour un nombre d'itérations de : <nombre d'itérations>.
Comparaison de PI.
Voici un exemple pour 50 itérations :
 
</td></tr>
<tr><td>Méthode : calculateToPercentagePi</td><td>Cette méthode utilise un pourcentage comme paramètre et essaye de calculer le nombre d'itérations requises pour obtenir une telle précision.
<u>Aide :</u>
Faites appel à la méthode calculatePercentage, commencez par 0 itération et incrémentez le nombre d'itérations à chaque tour. À la fin, appelez-la méthode compare pour afficher la comparaison.
<u>Par exemple :</u>
Pour obtenir une précision de 99%, on a besoin de 31 itérations.
La méthode affiche son résultat dans la console.
 
</td></tr>
<tr><td>Méthode : calculateToPercentage</td><td>Cette méthode prend un pourcentage en paramètre et calcule le nombre d'itérations requises pour obtenir une telle précision.
<u>Aide :</u>
Faites appel à la méthode calculatePI, en commençant par 0 itération, et incrémentez le nombre d'itérations à chaque tour. Après chaque appel de calculatePI, faites appel à la méthode calculatePercentage.
<u>Par exemple :</u>
Pour obtenir une précision de 99%, on a besoin de 31 itérations.
La méthode affiche cette information dans la console, puis elle
 
</td></tr>
</table>

C.40) AUTOCALCULATOR - PHASE 3

X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Reprenez la dernière phase de l'exercice « AutoCalculator – phase 2 ».

Reprenez la dernière phase de l'exercice ***AutoCalculator***.

Ajoutez les 2 méthodes décrites ci-dessous.

```
+ autoMultiple(pMin : int, pMax : int) : void
+ autoMultiple(pResult : int) : void
```

Partie	Description

Méthode : autoMultiple
 	
2 paramètres
	Cette méthode génère un nombre aléatoire d'opérations.
	• Avant de commencer, faites appel à la méthode ***reset***.
	• Le nombre d'opérations à réaliser est un nombre aléatoire entre ***pMin*** et ***pMax***.
	• On utilise des opérations aléatoires non sauvegardées, avec des nombres aléatoires entre 1 et 100. Faites appel à la méthode ***calculateRandom*** avec des paramètres appropriés.
	• Sauvegardez le résultat de chaque opération dans une variable locale et affichez un texte dans la console. Pour le texte, faites appel à la méthode ***toString***.
	• À la fin de la méthode, faites appel à la méthode ***printStatistics***.
	Exemple :
	`1: Opération soustraction le résultat n'a pas été sauvegardé. --> 43.751828538389894`
	`2: Opération multiplication le résultat n'a pas été sauvegardé. --> 423.6741974931498`
	`3: Opération soustraction le résultat n'a pas été sauvegardé. --> -61.367717425871504`
	`4: Opération addition le résultat n'a pas été sauvegardé. --> 79.02691454419893`
	`5: Opération soustraction le résultat n'a pas été sauvegardé. --> 28.603627547310847`
	`6: Opération multiplication le résultat n'a pas été sauvegardé. --> 1868.324401916845`
	`7: Opération soustraction le résultat n'a pas été sauvegardé. --> 11.11774607956923`

Partie	Description

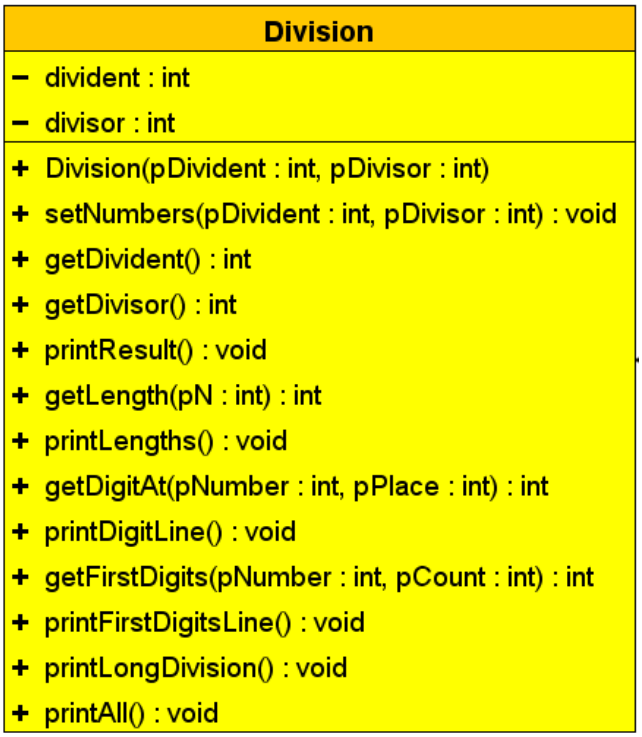
Méthode : autoMultiple
 	
1 paramètre
	Cette méthode génère un nombre aléatoire d'opérations. Le paramètre ***pResult*** indique la valeur minimale du résultat. C'est-à-dire tant que le résultat (sauvegardé dans l'attribut ***n***) est inférieur à cette limite, il faut continuer à calculer.
	• Avant de commencer, faites appel à la méthode ***reset***.
	• Affichez le texte « Résultat désiré : » suivi de la valeur ***pResult***.
	• On utilise des opérations aléatoires sauvegardées, avec des nombres aléatoires de l'intervalle [1 ; 100]. Faites appel à la méthode ***calculateRandom*** avec les paramètres appropriés.
	• Pour chaque opération, affichez un texte dans la console. Pour le texte, faites appel à la méthode ***toString***. Le nombre à la fin est la valeur de ***n***.
	• À la fin de la méthode, faites appel à la méthode ***printStatistics***.
	Exemple :
	`Résultat désiré: 100`
	`1: Opération multiplication le résultat a été sauvegardé. --> 0.0`
	`2: Opération multiplication le résultat a été sauvegardé. --> 0.0`
	`3: Opération addition le résultat a été sauvegardé. --> 64.47826868206963`
	`4: Opération division le résultat a été sauvegardé. --> 1.1253705600581616`

C.41) Division

X	Concepts de base	X	Boucles (FOR / WHILE)	X	Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué	X	Opérateurs logiques		

C.41.1) Classe « Division »

Développez une classe ***Division*** telle que décrite ci-dessous :



Les opérateurs « / » et « % » sont utilisés dans la plupart des méthodes. Révisez leur fonctionnement dans le support de cours. Ces opérateurs « / » et « % » possèdent des propriétés intéressantes.

Prenons le nombre12345 comme exemple.

- La division entière par « 10 » vous « coupe » le dernier chiffre du nombre.

  12345 / 10 = 1234

- Le reste de la division entière par « 10 » vous « retourne » le dernier chiffre du nombre. 12345 % 10 = 5

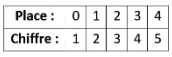
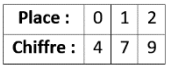
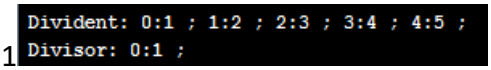
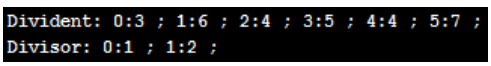
Utilisez ces propriétés dans les méthodes à écrire dans cet exercice.

Partie	Description
Attribut : dividend	Cet attribut contient le nombre à diviser.
Attribut : divisor	Cet attribut contient le diviseur.
Accesseurs & Manipulateur	Créez les accesseurs indiqués. Le manipulateur initialise les attributs. Si le diviseur indiqué (***pDivisor***) est égal à 0, le diviseur est à initialiser à ***1***.
Constructeur	Le constructeur appelle la méthode ***setNumbers***.

Méthode : printResult	Cette méthode vous affiche le résultat décimal et le résultat entier de la division des deux attributs. Voici une capture d'écran :
	`Calculation: 1203 / 12`
	`Decimal: 100.25 and 100.25 * 12 = 1203.0`
	`Integer: 100 Remainder 3 and 100 * 12 + 3 = 1203`

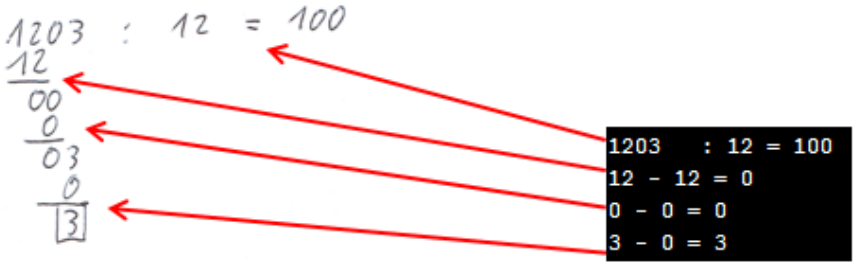
Méthode : getLength	Cette méthode vous retourne la longueur d'un nombre entier. Par exemple : • Le nombre 12345 possède une longueur de 5. • Le nombre 1 possède une longueur de 1.

Méthode : printLengths	Cette méthode vous affiche les longueurs des attributs. Voici une capture d'écran.
	`Divident length: 5 digit(s)`
	`Divisor length: 1 digit(s)`
	Faites appel à la méthode ***getLength***.

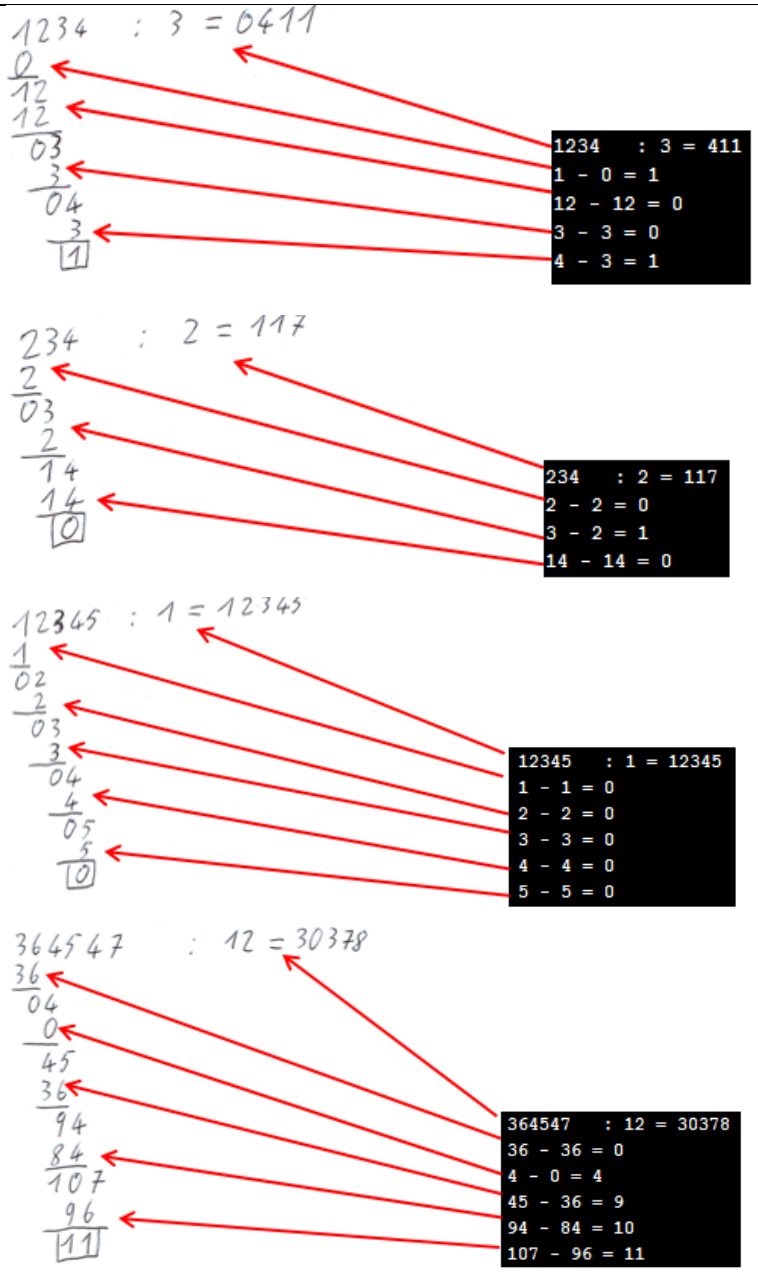
Méthode : getDigitAt	Cette méthode vous retourne le chiffre à la position demandée. La numérotation commence avec la position 0 tout à gauche. Si on spécifie une position qui n'est pas dans l'intervalle [0 ; ***getLength()***[, la méthode retourne ***-1***. Voici deux exemples : • Pour le nombre 12345, le chiffre à la position 1 est 2. Le chiffre à la position 4 est 5.  • Pour le nombre 479, le chiffre à la position 0 est 4. Le chiffre à la position 2 est 9. 
Méthode : printDigitLine	Cette méthode vous affiche les chiffres des attributs dans une ligne avec leur position à côté. Voici deux exemples : • Pour 12345 et 1  • Pour 36457 et 12 

<table>
<tr>
<td>Méthode :
getFirstDigits</td>
<td>Cette méthode retourne les pCount premiers chiffres du nombre pNumber.
Par exemple, pour les nombres 12345 et 479, on aurait respectivement :
<table>
<tr><th>pCount</th><th>Résultat</th></tr>
<tr><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td></tr>
<tr><td>2</td><td>12</td></tr>
<tr><td>3</td><td>123</td></tr>
<tr><td>4</td><td>1234</td></tr>
<tr><td>5</td><td>12345</td></tr>
</table>
<table>
<tr><th>pCount</th><th>Résultat</th></tr>
<tr><td>0</td><td>0</td></tr>
<tr><td>1</td><td>4</td></tr>
<tr><td>2</td><td>47</td></tr>
<tr><td>3</td><td>479</td></tr>
</table>
</td>
</tr>
<tr>
<td>Méthode :
printFirst
DigitsLine</td>
<td>Cette méthode vous affiche les premiers chiffres possibles des attributs.
Voici une capture d’écran pour les attributs 12345 et 1.
<pre>Divident: 0 first: 0 ; 1 first: 1 ; 2 first: 12 ; 3 first: 123 ; 4 first: 1234 ; 5 first: 12345 ;
Divisor: 0 first: 0 ; 1 first: 1 ;</pre>
</td>
</tr>
<tr>
<td>Méthode :
printLong
Division</td>
<td>Cette méthode vous affiche toutes les étapes d’une division entière. Réalisez les calculs l’un après l’autre. C’est-à-dire n’utilisez pas les opérateurs « / » et « % » pour obtenir directement le résultat final. Vous pouvez utiliser ces opérateurs pour les étapes intermédiaires.
Étapes à faire :

Faites les calculs sur papier et essayez de créer votre propre algorithme.
Créez un structogramme et vérifiez le fonctionnement avec un tableau de variables.
Programmez la méthode.

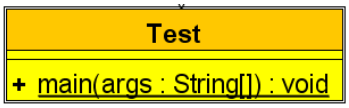
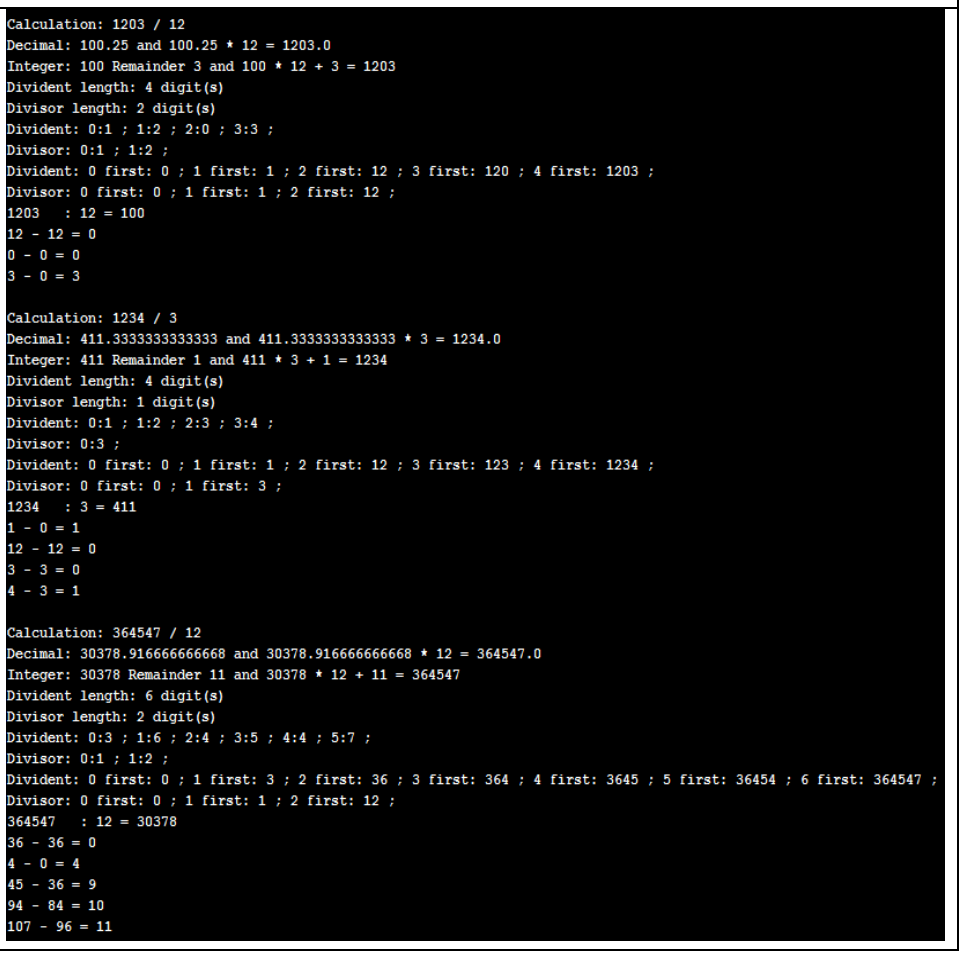
<u>Aide :</u>
N’utilisez qu’un seul System.out.println à la fin. Concaténez le texte à afficher pendant les calculs. Pour obtenir une nouvelle ligne, utilisez « \n’ ».
Voici quelques captures pour vous aider :


</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>


</td>
</tr>
<tr>
<td>Méthode :
printAll</td>
<td>Cette méthode appelle toutes les méthodes print. Par exemple pour le nombre 1203 et 12.
<pre>
Calculation: 1203 / 12
Decimal: 100.25 and 100.25 * 12 = 1203.0
Integer: 100 Remainder 3 and 100 * 12 + 3 = 1203
Divident length: 4 digit(s)
Divisor length: 2 digit(s)
Divident: 0:1 ; 1:2 ; 2:0 ; 3:3 ;
Divisor: 0:1 ; 1:2 ;
Divident: 0 first: 0 ; 1 first: 1 ; 2 first: 12 ; 3 first: 120 ; 4 first: 1203 ;
Divisor: 0 first: 0 ; 1 first: 1 ; 2 first: 12 ;
1203 : 12 = 100
12 - 12 = 0
0 - 0 = 0
3 - 0 = 3
</pre>
</td>
</tr>
</table>

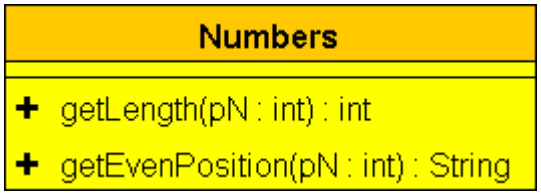
C.41.2) Classe « Test »

Classe « Test » à créer	Créez la classe ***Test***. 
Objets à créer	Créez 3 objets de type ***Division*** en utilisant les valeurs : 1203 et12 1234 et 3 364547 et 12
Méthodes à appeler / Tests à réaliser	Faites appel à la méthode ***printAll*** sur chaque objet.
Aperçu du résultat	

C.42) NUMBERS

X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Développez une classe ***Numbers*** telle que décrite ci-dessous.



Partie	Description
Méthode : getLength	Cette méthode calcule et retourne la longueur du nombre ***pN***. Astuce : Comptez le nombre de divisions jusqu'à ***pN*** est 0.
Méthode : getEvenPosition	Cette méthode retourne, sous forme de texte, les chiffres qui se situent sur les positions paires (à partir de la gauche) du nombre ***pN***. Exemple pour ***pN*** = 123456, résultat = 246 Exemple pour ***pN*** = 123456789, résultat = 2468 Astuce : vérifiez si la longueur de ***pN*** est un nombre pair ou impair et générez le résultat de droite à gauche.

C.43) Dessin

X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple	X	Boucles imbriquées		Classe comme paramètre
	IF imbriqué	X	Opérateurs logiques		

Il s'agit de développer une classe permettant de dessiner des figures géométriques d'une certaine taille (hauteur et largeur) dans la console.
Il existe 5 figures au total (carré, rectangle, triangle, sapin et losange).
Les figures possèdent une bordure et il est possible de spécifier les symboles utilisés pour les figures et les bordures.

```
Carré
Valeurs utilisées -> Largeur: 8 & Hauteur: 8
Valeurs sauvegardées -> Largeur: 8 & Hauteur: 6
----------
|********|
|********|
|********|
|********|
|********|
|********|
|********|
|********|
----------
```

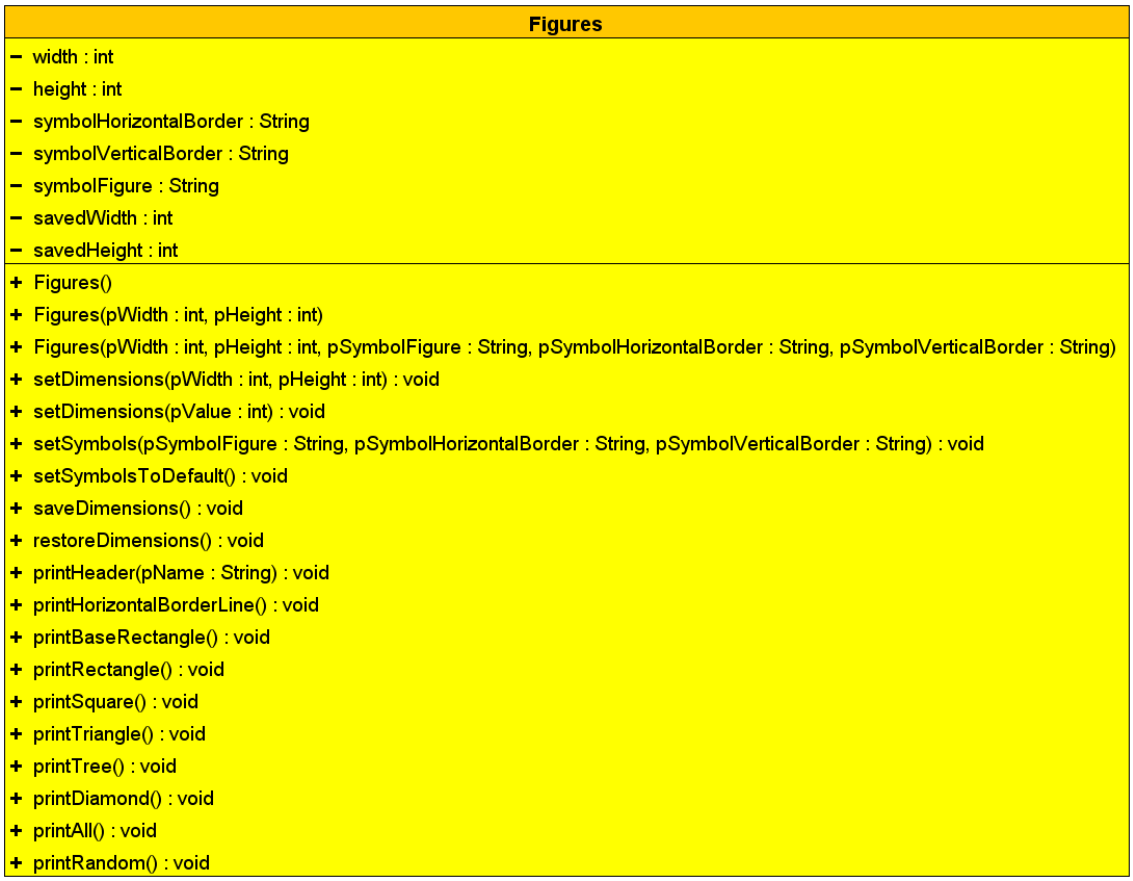
Voici des exemples pour chaque figure, les valeurs utilisées sont :

- Largeur → 8
- Hauteur → 6
- Symbole pour la figure → *
- Symbole pour la bordure verticale → |
- Symbole pour la bordure horizontale → -

```
Rectangle
Valeurs utilisées -> Largeur: 8 & Hauteur: 6
Les valeurs originales ont été utilisées.
----------
|********|
|********|
|********|
|********|
|********|
Triangle
Valeurs utilisées -> Largeur: 8 & Hauteur: 8
Valeurs sauvegardées -> Largeur: 8 & Hauteur: 6
----------
|*       |
|**      |
|***     |
|****    |
|*****   |
|******  |
|******* |
|********|
----------
```

```
Sapin
Valeurs utilisées -> Largeur: 11 & Hauteur: 6
Valeurs sauvegardées -> Largeur: 8 & Hauteur: 6
-------------
|     *     |
|    ***    |
|   *****   |
|  *******  |
| ********* |
|***********|
-------------
```

```
Losange
Valeurs utilisées -> Largeur: 9 & Hauteur: 9
Valeurs sauvegardées -> Largeur: 8 & Hauteur: 6
-----------
|    *    |
|   ***   |
|  *****  |
| ******* |
|*********|
| ******* |
|  *****  |
|   ***   |
|    *    |
-----------
```



Partie	Description
Attribut : width	Cet attribut représente la largeur utilisée par la figure.
Attribut : height	Cet attribut représente la hauteur utilisée par la figure
Attribut : symbolHorizontalBorder	Cet attribut représente le symbole utilisé pour la bordure horizontale.
Attribut : symbolVerticalBorder	Cet attribut représente le symbole utilisé pour la bordure verticale.
Attribut : symbolFigure	Cet attribut représente le symbole utilisé pour dessiner la figure.
Attribut : savedWidth	Pour certaines figures, il faut modifier les dimensions. La largeur originale est sauvegardée dans cet attribut dans ces cas.
Attribut : savedHeight	Pour certaines figures, il faut modifier les dimensions. La hauteur originale est sauvegardée dans cet attribut dans ces cas.

Partie	Description
Manipulateurs	La méthode ***setDimensions*** avec deux paramètres est lemanipulateur pour les dimensions ***width*** et ***height***. Si l'une de ces dimensions est inférieure à 1, elle est à initialiser à 1. La méthode ***setDimensions*** avec un paramètre appelle la méthode ***setDimensions*** avec deux paramètres en passant le paramètre ***pValue*** à la fois pour la hauteur et la largeur. La méthode ***setSymbols*** est le manipulateur pour les symboles.
Constructeurs	**Les constructeurs appellent les manipulateurs !** Le constructeur sans paramètres initialise les dimensions à 8 pour la largeur et à 6 pour la hauteur, et appelle la méthode ***setSymbolsToDefault***. Le constructeur avec 2 paramètres initialise les dimensions en utilisant les valeurs passées comme paramètres, et appelle la méthode ***setSymbolsToDefault***. Le constructeur avec 5 paramètres initialise tous les attributs.
Méthode : setSymbolsToDefault	Cette méthode appelle la méthode ***setSymbols*** en utilisant les valeurs « * », « - » et « | ».
Méthode : saveDimensions	Cette méthode sauvegarde les valeurs de ***width*** et ***height*** respectivement dans les attributs ***savedWidth***et ***savedHeight***.
Méthode : restoreDimensions	Cette méthode affecte les valeurs de ***savedWidth*** et ***savedHeight*** à ***width*** et ***height***. Puis elle met ***savedWidth*** et ***savedHeight*** à -1.

Partie	Description

Méthode : printHeader	Cette méthode affiche l'en-tête de la figure. Par exemple pour le carré :
	`Carré`
	`Valeurs utilisées -> Largeur: 8 & Hauteur: 8`
	`Valeurs sauvegardées -> Largeur: 8 & Hauteur: 6`
	Le nom est passé comme paramètre.
	Les autres valeurs à utiliser sont les dimensions et les dimensions sauvegardées.
	Si ***savedWidth*** et ***savedHeight*** sont ***-1***, il faut afficher le texte : « Les valeurs originales ont été utilisées. »

Méthode : printHorizontalBorderLine	Cette méthode affiche une ligne de bordure horizontale. La longueur cette ligne est ***width*** + 2. Veillez à utiliser le caractère adéquat.
Méthode : printBaseRectangle	Cette méthode affiche un rectangle avec les dimensions ***width*** et ***height***. Elle affiche aussi la bordure, mais pas d'en-tête.
Méthode : printRectangle	Cette méthode affiche l'en-tête adéquat pour un rectangle et appelle la méthode ***printBaseRectangle***.
Méthode : printSquare	Cette méthode affiche l'en-tête adéquat pour un carré et appelle la méthode ***printBaseRectangle***. Puisque tous les côtés d'un carré font la même longueur, on utilise le plus grand nombre de ***width*** et ***height*** comme taille. Sauvegardez les dimensions actuelles en appelant ***saveDimensions***, modifiez les dimensions en appelant ***setDimensions*** et à la fin restaurez les dimensions en appelant ***restoreDimensions***.
Méthode : printTriangle	Cette méthode affiche un triangle avec bordure et en-tête correct. Ce triangle a la même dimension en hauteur et en largeur. De même que pour le carré, utilisez le maximum de ***width*** et de ***height***. Sauvegardez les dimensions actuelles en appelant ***saveDimensions***, modifiez les dimensions en appelant ***setDimensions*** et à la fin restaurez les dimensions en appelant ***restoreDimensions***.

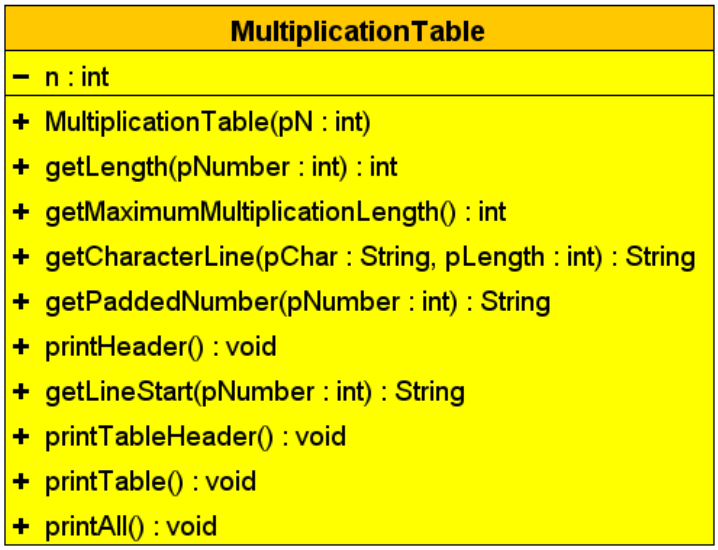
Partie	Description
Méthode : printTree	Cette méthode affiche un sapin avec bordure et en-tête correct. On doit utiliser une largeur spécifique par rapport à la hauteur. La formule pour la largeur est : 2 * ***height*** - 1. Sauvegardez les dimensions actuelles en appelant ***saveDimensions***, modifiez les dimensions en appelant ***setDimensions*** et à la fin restaurez les dimensions en appelant ***restoreDimensions***.
Méthode : printDiamond	Cette méthode affiche un losange avec bordure et en-tête correct. Un losange a la même dimension en hauteur et en largeur. Utilisez le maximum entre ***width*** et ***height***. Un losange peut seulement être dessiné si la taille est impaire. Si la taille est paire, incrémentez-la. Sauvegardez les dimensions actuelles en appelant ***saveDimensions***, modifiez les dimensions en appelant ***setDimensions*** et à la fin restaurez les dimensions en appelant ***restoreDimensions***.
Méthode : printAll	Cette méthode affiche toutes les figures possibles l'une après l'autre.
Méthode : printRandom	Cette méthode affiche une figure aléatoire. Aide : Générez un nombre aléatoire entre 1 et 5 et selon ce nombre affichez une figure. Par exemple : 1 = carré, 2= rectangle, 3 = triangle, 4 = sapin, 5 = losange.

C.44) TABLE DE MULTIPLICATION

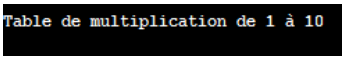
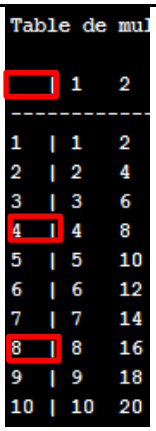
X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple	X	Boucles imbriquées		Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

Il s'agit de développer une classe permettant d'afficher en console la table de multiplication de 1 à n, c.-à-d. une matrice de n x n, avec les lignes et colonnes numérotées de 1 à n, et dans chaque case le produit de la ligne par la colonne.
Par exemple pour une table avec n = 10 ;

```
Table de multiplication de 1 à 10

   | 1   2   3   4   5   6   7   8   9   10
-------------------------------------------
1  | 1   2   3   4   5   6   7   8   9   10
2  | 2   4   6   8   10  12  14  16  18  20
3  | 3   6   9   12  15  18  21  24  27  30
4  | 4   8   12  16  20  24  28  32  36  40
5  | 5   10  15  20  25  30  35  40  45  50
6  | 6   12  18  24  30  36  42  48  54  60
7  | 7   14  21  28  35  42  49  56  63  70
8  | 8   16  24  32  40  48  56  64  72  80
9  | 9   18  27  36  45  54  63  72  81  90
10 | 10  20  30  40  50  60  70  80  90  100
```



Partie	Description
Attribut : **n**	Cet attribut représente le nombre de lignes et le nombre de colonnes de la matrice (matrice de ***n*** x ***n***).
Constructeur	Le constructeur initialise l'attribut ***n***. Si la valeur du paramètre ***pN*** est inférieure à 1, ***n*** est initialisé à 1, sinon la valeur de ***pN*** est utilisée. À la fin, la méthode ***printAll*** est appelée.
Méthode : **getLength**	Cette méthode détermine la longueur du nombre passé comme paramètre. Par exemple : • Le nombre 4 a une longueur de 1. • Le nombre 45 a une longueur de 2 • Le nombre 450 a une longueur de 3. Aide : Comptez le nombre de divisions entières par 10 possibles avant que le résultat soit de 0.
Méthode : **getMaximum MultiplicationLength**	Cette méthode détermine la longueur maximale pour les résultats de la multiplication. Par exemple : • Si ***n*** = 10 (longueur 2), la longueur maximale est de 3 (nombre 100). • Si ***n*** = 100 (longueur 3), la longueur maximale est de 5 (nombre 10000). La formule à utiliser est : 2 * <Longueur de ***n***> - 1
Méthode : **getCharacterLine**	Cette méthode retourne une chaîne de caractères de longueur ***pLength***, contenant seulement le caractère spécifié par ***pChar***. Par exemple : ***pChar*** = '-' et ***pLength*** = 3 => « --- » ***pChar*** = '#' et ***pLength*** = 5 => « ##### »
Méthode : **getPaddedNumber**	Cette méthode retourne une chaîne de caractères contenant le nombre spécifié en paramètre avec un certain nombre (allemand : Anzahl) d'espaces à la fin. La longueur totale doit être de <longueur maximale pour les résultats de la multiplication> + 1 Le nombre d'espaces à utiliser est à calculer dans la méthode.

<table>
<tr><td></td><td>Par exemple, si la longueur maximale est de 5 :
Pour le nombre pNumber = 5, on obtient 5 avec 5 espaces à la fin. Pour illustrer ceci, représentons les espaces avec le caractère '_'. Le résultat retourné par la méthode sera donc « 5_ _ _ _ _ ».
Pour le nombre pNumber = 10, on obtient 10 avec 4 espaces à la fin. Le résultat retourné par la méthode sera donc « 10_ _ _ _ ».
Pour le nombre pNumber = 100, on obtient 100 avec 3 espaces à la fin. Le résultat retourné par la méthode sera donc « 100_ _ _ ».</td></tr>
<tr><td>Méthode :
printHeader</td><td>Cette méthode affiche l’en-tête dans la console.
L’en-tête est un texte indiquant la valeur de n, suivi d’une ligne blanche.
Par exemple pour n = 10 :
  </td></tr>
<tr><td>Méthode :
getLineStart</td><td> 
Cette méthode retourne la première partie de la ligne.
Pour la ligne en haut de la matrice (ligne contenant la numérotation des colonnes), le début consiste en un certain nombre d'espaces suivis du caractère « | ». Par exemple : « _ _ _ _ _| ».
Pour les autres lignes, le numéro de la ligne est ajouté au début et remplace les premiers espaces (chaque chiffre du numéro remplaçant un espace). Par exemple : « <nombre>_ _ _| ».
Si la valeur du paramètre pNumber est 0, aucun numéro n’est ajouté (première ligne en haut), sinon le numéro pNumber est ajouté.
La longueur totale de la première partie (jusqu’au caractère « | » exclus) est calculée avec la formule
<longueur de n> + 1
Le nombre d’espaces se déduit d’après la longueur du nombre pNumber par rapport à la longueur totale.</td></tr>
</table>

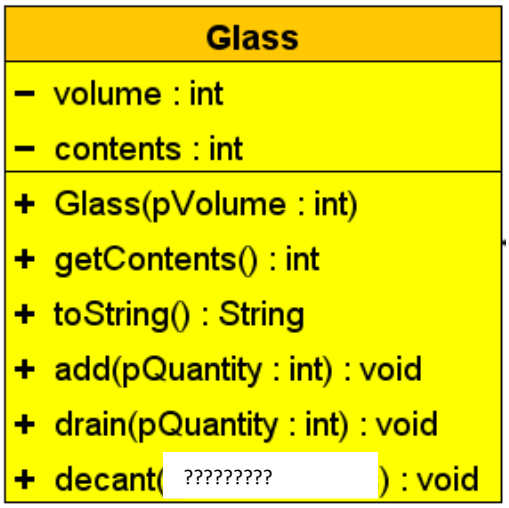
Méthode : printTableHeader	Cette méthode affiche l'en-tête du tableau. Par exemple pour ***n*** = 10 :  Elle affiche toujours deux lignes : • le résultat de la méthode ***getLineStart*** suivi des numéros de colonnes. (• faites attention aux espacements entre les nombres, utilisez la méthode ***getPaddedNumber***), • une ligne de « - » dont vous devez calculer la longueur.
Méthode : printTable	Cette méthode affiche le reste du tableau, ligne par ligne. Utilisez la méthode ***getLineStart*** pour le début de la ligne et la méthode ***getPaddedNumber*** pour les espacements. Référez-vous à la capture d'écran de la première page pour vous guider.
Méthode : printAll	Cette méthode affiche la table de multiplication au complet, en appelant les méthodes ***printHeader***, ***printTableHeader*** et ***printTable***.

C.45) GLASS[3]

X	Concepts de base		Boucles (FOR / WHILE)		Classe Test
X	IF simple		Boucles imbriquées	X	Classe comme paramètre
	IF imbriqué		Opérateurs logiques		

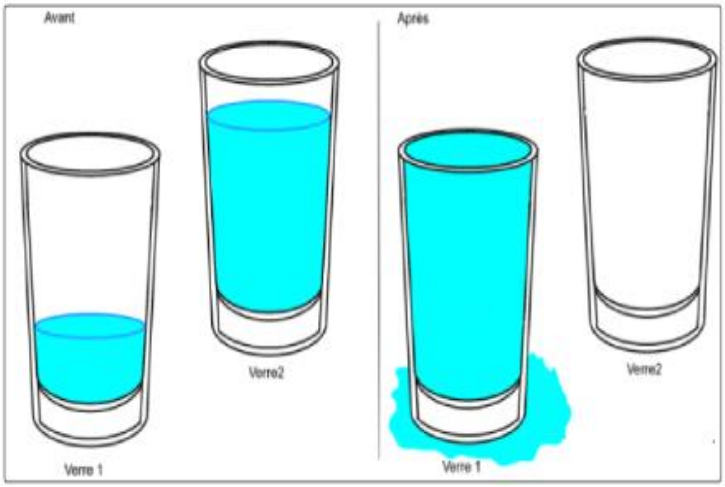
C.45.1) CLASSE « GLASS »

Développez la classe ***Glass*** telle que décrite ci-dessous.



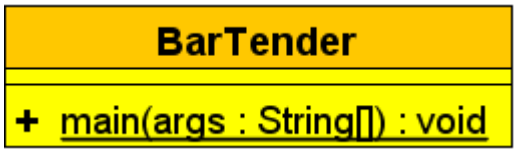
Partie	Description
Attribut : volume	Cet attribut contient le volume (c.-à-d. la contenance maximale) du verre en centilitres (en cl).
Attribut : contents	Cet attribut contient le volume du liquide contenu dans le verre (en cl).
Accesseurs	Créez l'accesseur indiqué.
Constructeur	Le constructeur initialise les attributs : • ***volume*** avec les valeurs du paramètre • ***contents*** à 0
Méthode : toString	Cette méthode retourne un texte donnant l'état du verre, c'est-à-dire son contenu actuel (le volume de liquide) par rapport au volume maximum, par exemple : « 5cl / 15cl ».
Méthode : add	Cette méthode ajoute du liquide dans le verre. Dépasser le volume maximal est à éviter.
Méthode : drain	Cette méthode retire du liquide du verre. On ne peut pas avoir un volume actuel négatif.

[3] Auteur: Carlos Gamboa (LTETT)

Partie	Description
Méthode : decant	Cette méthode prend un verre comme paramètre et ne retourne aucune valeur. Elle transvase (allemand : umfüllen) tout le contenu du verre passé en paramètre dans le verre courant. 

C.45.2) Classe « BarTender »

Développez la classe Test nommée ***BarTender*** qui joue le rôle du barman et permet de tester si la classe ***Glass*** fonctionne correctement.



Pour cela, suivez les étapes suivantes :

- Créez les objets ***glass1*** (volume : 15cl) et ***glass2*** (volume : 20cl) et affichez leur état dans la console.
- Remplissez le premier verre avec 5cl et le second verre avec 18cl et affichez leur état dans la console.
- Transvasez le contenu du second verre dans le premier et affichez leur état.

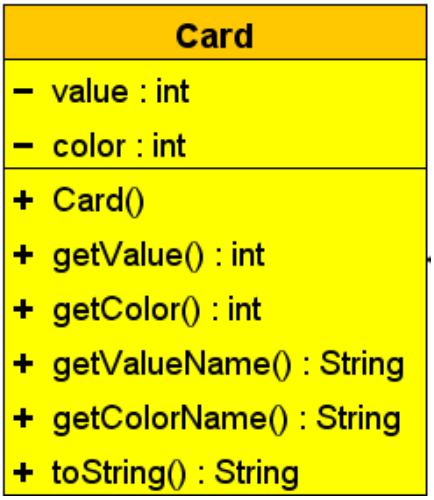
Le résultat de cet exemple est :

```
Verre d'eau 1: 0cl / 15cl
Verre d'eau 2: 0cl / 20cl
... remplir ...
Verre d'eau 1: 5cl / 15cl
Verre d'eau 2: 18cl / 20cl
... transvaser ...
Verre d'eau 1: 15cl / 15cl
Verre d'eau 2: 0cl / 20cl
```

C.46) JEU DE CARTES[4]

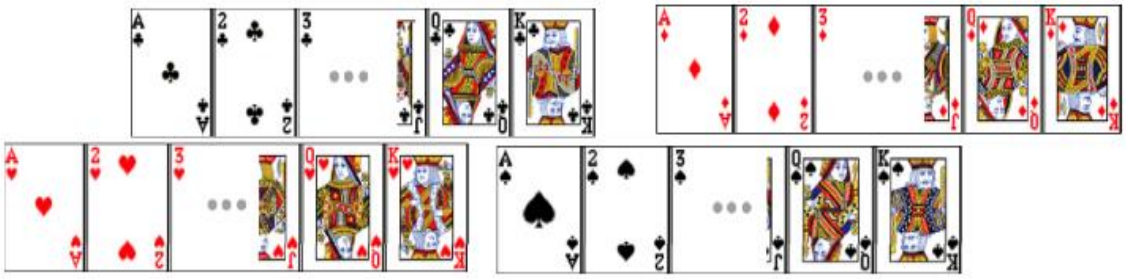
X	Concepts de base	X	Boucles (FOR / WHILE)	X	Classe Test
X	IF simple		Boucles imbriquées	X	Classe comme paramètre
X	IF imbriqué	X	Opérateurs logiques		

Développez les classes ***Card***. Voici le diagramme UML.



C.46.1) LA CLASSE « CARD »

Cette classe décrit une carte à jouer classique.

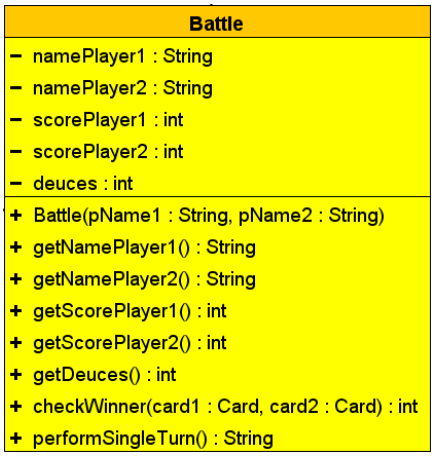


[4] Auteur: Carlos Gamboa (LTETT)

Partie	Description
Attribut : **value**	Cet attribut est un entier correspondant à la valeur de la carte. Cette valeur est égale au nombre marqué sur la carte pour les cartes de 2 à 10. Dans les autres cas, l'attribut prend les valeurs suivantes : • 1 pour l'as • 11 pour le valet (allemand : Bube) • 12 pour la dame (allemand : Dame) • 13 pour le roi (allemand : König)
Attribut : **color**	Cet attribut est un entier qui décrit la couleur de la carte (allemand : Farbwert). Il prend les valeurs suivantes : • 1 pour trèfle (allemand : Kreuz) • 2 pour carreau (allemand : Karo) • 3 pour cœur (allemand : Herz) • 4 pour pique (allemand : Pik)
Accesseurs	Créez les accesseurs indiqués.
Constructeur	Le constructeur initialise la valeur de la carte avec un nombre aléatoire entier compris dans l'intervalle [1 ; 13] et sa couleur avec un nombre aléatoire de l'intervalle [1 ; 4].
Méthode : **getValueName**	Cette méthode retourne la valeur de la carte sous forme de texte dans lequel les valeurs 1, 11, 12 et 13 sont remplacées par leur véritable nom (« as », « valet », « dame » ou « roi »).
Méthode : **getColorName**	Cette méthode retourne la couleur sous son véritable nom (« trèfle », « carreau », « cœur » ou « pique »).
Méthode : **toString**	Cette méthode retourne la description de la carte sous forme de texte. Par exemple : • « As de trèfle » • « 3 de pique » • « 10 de carreau » • « Roi de cœur »

C.46.2) La classe « Battle »

Développez les classes ***Battle***. Voici le diagramme UML.



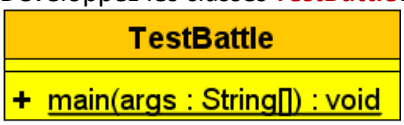
Cette classe permet de réaliser la base du jeu de la bataille entre deux cartes à jouer.

Partie	Description
Attribut : namePlayer1	Cet attribut contient le nom du premier joueur.
Attribut : namePlayer2	Cet attribut contient le nom du deuxième joueur.
Attribut : scorePlayer1	Cet attribut contient les points du premier joueur.
Attribut : scorePlayer2	Cet attribut contient les points du deuxième joueur.
Attribut : deuces	Cet attribut contient le nombre d'égalités obtenues.
Accesseurs	Créez les accesseurs indiqués.
Constructeur	Le constructeur initialise les attributs.
Méthode : checkWinner	Cette méthode détermine (vérifie) la meilleure des deux cartes « passées en paramètre ». La méthode retourne : • 1 lorsque la carte 1 est meilleure, • -1 lorsque c'est la carte 2 qui est meilleure • 0 en cas d'égalité La meilleure carte est celle qui a la valeur la plus haute, **quelle que soit la couleur !** Lorsque les 2 cartes ont la même valeur, c'est la couleur qui détermine la meilleure carte en suivant l'ordre du plus faible au plus fort : trèfle, carreau, cœur et pique.

<table>
<tr><td></td><td>Exemples :
<table>
<tr><th>Carte 1</th><th></th><th>Carte 2</th><th>Résultat</th></tr>
<tr><td>4 de trèfle</td><td>est meilleur que</td><td>2 de pique</td><td>1</td></tr>
<tr><td>9 de trèfle</td><td>est moins bon que</td><td>9 de carreau</td><td>-1</td></tr>
<tr><td>Valet de pique</td><td>est égal à</td><td>Valet de pique</td><td>0</td></tr>
</table></td></tr>
<tr><td>Méthode :
performSingleTurn</td><td>Cette méthode construit deux cartes, une pour le joueur 1 et une pour le joueur 2.
Ensuite elle détermine la meilleure carte (c’est-à-dire celle qui gagne) et actualise les attributs.
Elle retourne un texte qui montre les cartes des deux joueurs et le vainqueur du tour. Par exemple :
« 3 de trèfle <> as de pique --> vainqueur = <nom joueur 1> »
« 10 de carreau <> 10 de carreau --> égalité! »</td></tr>
</table>

C.46.3) La classe « TestBattle »

Développez les classes ***TestBattle***. Voici le diagramme UML.



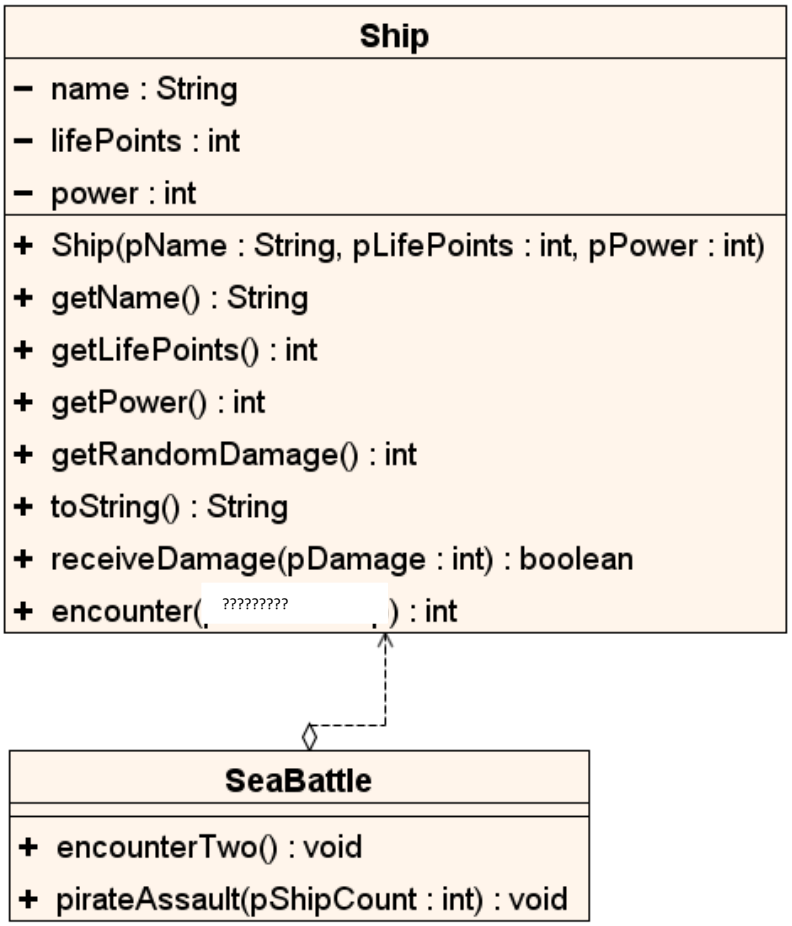
Cette classe permet de tester le jeu en effectuant 10 tours et d’afficher les statistiques à la fin, comme montrée dans l’exemple ci-dessous.

```
Tour 1: 6 de pique <> 4 de trefle -->  vainqueur = Alex
Tour 2: 10 de coeur <> dame de trefle -->  vainqueur = Bobby
Tour 3: valet de coeur <> 5 de coeur -->  vainqueur = Alex
Tour 4: 4 de pique <> 8 de trefle -->  vainqueur = Bobby
Tour 5: ass de trefle <> valet de carreau -->  vainqueur = Bobby
Tour 6: 6 de trefle <> 7 de coeur -->  vainqueur = Bobby
Tour 7: 7 de coeur <> 9 de coeur -->  vainqueur = Bobby
Tour 8: valet de carreau <> valet de carreau -->  vainqueur =  égalité
Tour 9: roi de trefle <> ass de coeur -->  vainqueur = Alex
Tour 10: 4 de pique <> 10 de coeur -->  vainqueur = Bobby

Alex: 3
Bobby: 6
égalités: 1
```

C.47) PIRATES[5]

X	Concepts de base	X	Boucles (FOR / WHILE)		Classe Test
X	IF simple	X	Boucles imbriquées	X	Classe comme paramètre
X	IF imbriqué		Opérateurs logiques		

Voici un diagramme UML contenant deux classes. Les classes sont les deux composantes d'un petit jeu contenant des bateaux (anglais : ships) qui se combattent.



[5] Auteurs: Carlos Gamboa et Olinger Alex (LTETT) ; modifications: Bourone Jean-Marie et Dziadek Pierre

C.47.1) Classe « Ship »

Cette classe représente un bateau.

Partie	Description
Attribut : name	Cet attribut représente le nom du bateau.
Attribut : lifePoints	Cet attribut représente les points de vie. Si ces points arrivent à 0, le bateau est détruit.
Attribut : power	Cet attribut représente la force du bateau. La force est le maximum de dommages que le bateau peut faire lors d'une attaque.
Accesseurs	Créez les accesseurs indiqués.
Constructeur	Le constructeur initialise tous les attributs.
Méthode : getRandom Damage	Cette méthode calcule et retourne un nombre aléatoire de l'intervalle [1 ; ***power***].
Méthode : toString	Cette méthode retourne l'état du bateau sous forme de texte. Par exemple, si le nom du bateau est « Licorne » et le bateau possède 7 points de vie et une force de 10, on veut comme résultat : « Licorne → 7 life 10 power » Cependant, si le bateau ne possède plus de points de vie (les points sont à 0), il faut afficher : « Licorne → DESTROYED !! »

Méthode : receive Damage	Cette méthode est appelée lors d'une attaque sur le bateau.
	• Le paramètre ***pDamage*** représente les points de vie à perdre. Cette méthode soustrait les points à perdre des points actuels. Veillez à ce que les points ne deviennent pas négatifs, le minimum à atteindre est 0.
	• Si le bateau est détruit, la méthode retourne ***true***, sinon ***false***.
	Cette méthode affiche aussi le statut avant l'attaque et le statut après l'attaque (faites appel à la méthode ***toString***). Voici deux exemples :
	• Le bateau « Admiral » possède 50 points de vie et est attaqué avec 10 points.
	`+++ Admiral - status BEFORE receiving damage: Admiral -> 50 life 20 power`
	`+++ Admiral - receives 10 damage`
	`+++ Admiral - status AFTER receiving damage: Admiral -> 40 life 20 power`
	• Le bateau « Pirate » possède 10 points de vie et est attaqué avec 8 points.
	`+++ Pirate - status BEFORE receiving damage: Pirate -> 10 life 10 power`
	`+++ Pirate - receives 8 damage`
	`+++ Pirate - status AFTER receiving damage: Pirate -> 2 life 10 power`

<table>
<tr><th>Partie</th><th>Description</th></tr>
<tr><td>Méthode :
encounter</td><td>
Cette méthode est utilisée quand un bateau rencontre un deuxième bateau passé en paramètre.

Deux tours sont à réaliser, chaque tour est une attaque (faites appel à la méthode receiveDamage) :

Le bateau sur lequel la méthode encounter est appelée, attaque le deuxième bateau (paramètre pOther)
Puis le deuxième bateau attaque le premier, si et seulement si ce deuxième bateau n’a pas été détruit lors de l’attaque du premier tour !
Utilisez la valeur de retour de la méthode receiveDamage pour déterminer ceci.

Les points de vie à perdre (passés comme paramètres à la méthode receiveDamage) sont des valeurs aléatoires générées par la méthode getRandomDamage.

La méthode encounter retourne un code renseignant sur le statut des deux bateaux :

Si aucun bateau n’a été détruit, elle retourne 0.
Si le bateau passé comme paramètre a été détruit, elle retourne 1.
Si le bateau sur lequel la méthode a été appelée a été détruit, elle retourne -1.

La méthode affiche certains messages dans la console. Les messages utilisent la méthode toString pour les données à afficher :

Au début, un message indique quel bateau attaque. Par exemple :
<code>Pirate -> 10 life 10 power **** ATTACKS **** Admiral -> 50 life 20 power</code>
Au début du deuxième tour, un message indique quel bateau contre-attaque. Par exemple :
<code>Admiral -> 40 life 20 power **** COUNTERS **** Pirate -> 10 life 10 power</code>
Si un des bateaux est détruit, un message indique quel est le vainqueur. Par exemple :
<code>Admiral -> 32 life 20 power **** WON THE BATTLE ****</code>
La méthode se termine en affichant « ---- » dans la console

Voici un exemple d’exécution :
<pre>
Pirate -> 10 life 10 power **** ATTACKS **** Admiral -> 50 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 50 life 20 power
+++ Admiral - receives 10 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 40 life 20 power
Admiral -> 40 life 20 power **** COUNTERS **** Pirate -> 10 life 10 power
+++ Pirate - status BEFORE receiving damage: Pirate -> 10 life 10 power
+++ Pirate - receives 8 damage
+++ Pirate - status AFTER receiving damage: Pirate -> 2 life 10 power
----
</pre>
</td></tr>
</table>

C.47.2) Classe « SeaBattle »

Cette classe simule une rencontre entre multiples bateaux.

Partie	Description

Méthode : encounterTwo	Cette méthode simule une rencontre entre deux bateaux :
	• « Admiral », vie 18, force 20
	• « Pirate », vie 11, force 15
	Créez les objets appropriés et faites appel à la méthode ***encounter*** sur le bateau « Pirate ».
	Si un des deux bateaux a été détruit, un message est affiché dans la console. Ce message indique quel bateau a été détruit. Si aucun bateau n'a été détruit, faites appel à la méthode ***encounter*** sur le bateau « Admiral ». Dans cet exemple d'exécution, le bateau « Pirate » est détruit :
	`Pirate -> 11 life 15 power **** ATTACKS **** Admiral -> 18 life 20 power`
	`+++ Admiral - status BEFORE receiving damage: Admiral -> 18 life 20 power`
	`+++ Admiral - receives 3 damage`
	`+++ Admiral - status AFTER receiving damage: Admiral -> 15 life 20 power`
	`Admiral -> 15 life 20 power **** COUNTERS **** Pirate -> 11 life 15 power`
	`+++ Pirate - status BEFORE receiving damage: Pirate -> 11 life 15 power`
	`+++ Pirate - receives 12 damage`
	`+++ Pirate - status AFTER receiving damage: Pirate -> DESTROYED !!`
	`Admiral -> 15 life 20 power **** WON THE BATTLE ****`
	`----`
	`PIRATE was destroyed !`

Méthode : pirateAssault	Cette méthode simule une rencontre avec de multiples bateaux. Le nombre de bateaux est passé en paramètre. Les bateaux en bataille sont : • « Admiral », vie 50, force 20 • Les bateaux des pirates « Pirate *i* », vie 10, force 15 Le bateau « Admiral » est attaqué par de multiples bateaux pirates, qui attaquent l'un après l'autre. • Au début du programme, on affiche combien de bateaux attaquent. • Un premier bateau pirate attaque le bateau « Admiral » de façon répétitive jusqu'à ce que celui-ci soit détruit ou que le bateau pirate soit détruit. Le tour se termine lorsqu'un des bateaux est détruit. • Si un bateau pirate est détruit, un autre est créé et prend sa place. Les noms des bateaux pirates contiennent le numéro du tour (en commençant à 0). Le premier bateau commence ainsi avec le nom « Pirate 0 ». • Le bateau « Admiral » n'est pas recréé. • On doit compter combien de bateaux ont été utilisés pour détruire le bateau « Admiral ».

- À la fin, un message est affiché : Si le bateau « Admiral » n'a pas été détruit, on affiche « ### The Admiral was victorious. », sinon on affiche combien de bateaux il a fallu pour le détruire.

Voici deux exemples d'exécution :

- On attaque avec 5 bateaux, le bateau « Admiral » est détruit.
- On attaque avec 3 bateaux, le bateau « Admiral » n'est pas détruit.

```
### Attacking the admiral with 5 ships.
Pirate 0 -> 10 life 15 power **** ATTACKS **** Admiral -> 50 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 50 life 20 power
+++ Admiral - receives 15 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 35 life 20 power
Admiral -> 35 life 20 power **** COUNTERS **** Pirate 0 -> 10 life 15 power
+++ Pirate 0 - status BEFORE receiving damage: Pirate 0 -> 10 life 15 power
+++ Pirate 0 - receives 18 damage
+++ Pirate 0 - status AFTER receiving damage: Pirate 0 -> DESTROYED !!
Admiral -> 35 life 20 power **** WON THE BATTLE ****
----
Pirate 1 -> 10 life 15 power **** ATTACKS **** Admiral -> 35 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 35 life 20 power
+++ Admiral - receives 14 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 21 life 20 power
Admiral -> 21 life 20 power **** COUNTERS **** Pirate 1 -> 10 life 15 power
+++ Pirate 1 - status BEFORE receiving damage: Pirate 1 -> 10 life 15 power
+++ Pirate 1 - receives 6 damage
+++ Pirate 1 - status AFTER receiving damage: Pirate 1 -> 4 life 15 power
----
Pirate 1 -> 4 life 15 power **** ATTACKS **** Admiral -> 21 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 21 life 20 power
+++ Admiral - receives 8 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 13 life 20 power
Admiral -> 13 life 20 power **** COUNTERS **** Pirate 1 -> 4 life 15 power
+++ Pirate 1 - status BEFORE receiving damage: Pirate 1 -> 4 life 15 power
+++ Pirate 1 - receives 14 damage
+++ Pirate 1 - status AFTER receiving damage: Pirate 1 -> DESTROYED !!
Admiral -> 13 life 20 power **** WON THE BATTLE ****
----
Pirate 2 -> 10 life 15 power **** ATTACKS **** Admiral -> 13 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 13 life 20 power
+++ Admiral - receives 13 damage
+++ Admiral - status AFTER receiving damage: Admiral -> DESTROYED !!
Pirate 2 -> 10 life 15 power **** WON THE BATTLE ****
----
### It took 3 ships to destroy the admiral.
```

```
### Attacking the admiral with 3 ships.
Pirate 0 -> 10 life 15 power **** ATTACKS **** Admiral -> 50 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 50 life 20 power
+++ Admiral - receives 11 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 39 life 20 power
Admiral -> 39 life 20 power **** COUNTERS **** Pirate 0 -> 10 life 15 power
+++ Pirate 0 - status BEFORE receiving damage: Pirate 0 -> 10 life 15 power
+++ Pirate 0 - receives 4 damage
+++ Pirate 0 - status AFTER receiving damage: Pirate 0 -> 6 life 15 power
----
Pirate 0 -> 6 life 15 power **** ATTACKS **** Admiral -> 39 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 39 life 20 power
+++ Admiral - receives 14 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 25 life 20 power
Admiral -> 25 life 20 power **** COUNTERS **** Pirate 0 -> 6 life 15 power
+++ Pirate 0 - status BEFORE receiving damage: Pirate 0 -> 6 life 15 power
+++ Pirate 0 - receives 6 damage
+++ Pirate 0 - status AFTER receiving damage: Pirate 0 -> DESTROYED !!
Admiral -> 25 life 20 power **** WON THE BATTLE ****
----
Pirate 1 -> 10 life 15 power **** ATTACKS **** Admiral -> 25 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 25 life 20 power
+++ Admiral - receives 15 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 10 life 20 power
Admiral -> 10 life 20 power **** COUNTERS **** Pirate 1 -> 10 life 15 power
+++ Pirate 1 - status BEFORE receiving damage: Pirate 1 -> 10 life 15 power
+++ Pirate 1 - receives 18 damage
+++ Pirate 1 - status AFTER receiving damage: Pirate 1 -> DESTROYED !!
Admiral -> 10 life 20 power **** WON THE BATTLE ****
----
Pirate 2 -> 10 life 15 power **** ATTACKS **** Admiral -> 10 life 20 power
+++ Admiral - status BEFORE receiving damage: Admiral -> 10 life 20 power
+++ Admiral - receives 3 damage
+++ Admiral - status AFTER receiving damage: Admiral -> 7 life 20 power
Admiral -> 7 life 20 power **** COUNTERS **** Pirate 2 -> 10 life 15 power
+++ Pirate 2 - status BEFORE receiving damage: Pirate 2 -> 10 life 15 power
+++ Pirate 2 - receives 12 damage
+++ Pirate 2 - status AFTER receiving damage: Pirate 2 -> DESTROYED !!
Admiral -> 7 life 20 power **** WON THE BATTLE ****
----
### The Admiral was victorious.
```