CMI - L3 : Programmation

BULOUP Frank

Aix Marseille Université
Institut des Sciences du Mouvement

Aix*Marseille université

cnrs

INSTITUT DES SCIENCES ETIENNE DU MOUVEMENT JULES MAREY

Programmation

1. La notion d'objet
2. La classe String et les tableaux
3. Structures conditionnelles et boucles
4. Problème

Programmation

1. La notion d'objet
2. La classe String et les tableaux
3. Structures conditionnelles et boucles
4. Problème

fr.amu.fss.l3.cmi.geometry

Rectangle
width : double height : double
getPerimeter() : double getSurface() : double

Circle
radius : double
getPerimeter() : double getSurface() : double

Mais je fais quoi avec ces classes maintenant moi ? ? ? ! ! !

Comment créer un rectangle de hauteur 2 et de largeur 3 ?

J'utilise le **constructeur** de la classe et j'accède aux attributs et méthodes à l'aide de l'opérateur "."

```java
// Construction
Rectangle rect1 = new Rectangle();
// Attributes initialization
rect1.width = 3;
rect1.height = 2;
// Print rect1 surface
System.out.println(rect1.getSurface())
```

Si j'ai deux rectangles à créer :

```java
// Create two rectangles
Rectangle rect1 = new Rectangle();
Rectangle rect2 = new Rectangle();
// Init. their dimensions
rect1.width = 3; rect1.height = 2;
rect2.width = 0.5; rect2.height = 1.2;
// Print surfaces
System.out.println(rect1.getSurface())
System.out.println(rect2.getSurface())
```

```java
// Construction
Rectangle rect1 = new Rectangle();
// Attributes initialization
rect1.width = 3;
rect1.height = 2;
// Print rect1 surface
System.out.println(rect1.getSurface())
```

Si j'ai deux rectangles à créer :

```java
// Create two rectangles
Rectangle rect1 = new Rectangle();
Rectangle rect2 = new Rectangle();
// Init. their dimensions
rect1.width = 3; rect1.height = 2;
rect2.width = 0.5; rect2.height = 1.2;
// Print surfaces
System.out.println(rect1.getSurface())
System.out.println(rect2.getSurface())
```

⇒ **Les liens sémantiques sont conservés**

L'opérateur **new** à pour but de créer une occurence d'une classe en réservant de la place en mémoire : un **objet**. Le programme précédent donne donc naissance à deux objets lors de son exécution :

rect1: Rectangle
width = 3 height = 2

rect2: Rectangle
width = 0.5 height = 1.2

Un objet, une zone mémoire ! c.a.d ?

Un objet est une référence vers une zone mémoire : c'est un pointeur vers une adresse en mémoire. Les données sont contenues à cette adresse. En java, il n'est pas possible de manipuler ces références et il n'y a pas d'arithmétique des pointeurs. Il est par contre possible d'obtenir un identifiant unique d'un objet, un équivalent de l'adresse mémoire : c'est le **hash code**.

Exercice I - Comparaison d'objet

Reprennez l'exercice VII de la séquence précédente :

- Ouvrez la classe **Test**.
- Créez deux rectangles de dimensions identiques.
- Imprimez le résultat de la comparaison de ces deux classes.
- Pourquoi ces objets ne sont-ils pas identiques ?
- Imprimez le résultat de la comparaison des dimensions.

Égalité d'objets et de primitifs

- objet1 == objet2 compare les adresses mémoires des objets
- primitif1 == primitif2 compare les valeurs des primitifs

Exercice II - Objets et pointeurs : utilisation du debugger Eclipse

Reprennez l'exercice VII de la séquence précédente :

1. Ouvrez la classe **Test**.
2. Ajoutez un point d'arrêt sur la première ligne contenue dans la méthode **main(...)** de cette classe.
3. Lancez Eclipse en mode debug (accepter le passage en perspective debug) et observer les valeurs des variables locales.
4. Ces observations vont-elles dans le sens de la conclusion de l'exercice précedent ?
5. Quelle est la valeur par défaut d'un objet non instancié ?

Exercice II - Objets et pointeurs : utilisation du debugger Eclipse

Reprennez l'exercice VII de la séquence précédente :

1. Ouvrez la classe **Test**.
2. Ajoutez un point d'arrêt sur la première ligne contenue dans la méthode **main(...)** de cette classe.
3. Lancez Eclipse en mode debug (accepter le passage en perspective debug) et observer les valeurs des variables locales.
4. Ces observations vont-elles dans le sens de la conclusion de l'exercice précedent ?
5. Quelle est la valeur par défaut d'un objet non instancié ?

Hash code et ID

Le debugger donne des informations sur les intances des classes :

1. L'ID : automatiquement déterminé par le debugger d'Eclipse
2. Le hash code : NomClasse@hashCode (fonction de hachage)

```
1  class Rectangle {
2    double width;
3    double height;
4    double getSurface() {
5      return width*height;
6    }
7  }
8  class Test {
9    public static void main(String[] args) {
10     Rectangle rect1 = new Rectangle();
11     rect1.width = 10;
12     rect1.height = 10;
13   }
14 }
```

Les attributs **width** et **height** sont visibles :

- dans la classe **Rectangle** bien sûr
- mais aussi en dehors de la classe **Rectangle**

⇒ **Est-ce une bonne idée ?**

Les attributs **width** et **height** sont visibles :

- dans la classe **Rectangle** bien sûr
- mais aussi en dehors de la classe **Rectangle**

$\Rightarrow$ **Est-ce une bonne idée ? NON !**

Certains attributs servent au fonctionnement interne d'une classe. S'il est possible de les modifier à l'extérieur, c'est potentiellement source de problèmes, de bugs.

Les attributs **width** et **height** sont visibles :

- dans la classe **Rectangle** bien sûr
- mais aussi en dehors de la classe **Rectangle**

$\Rightarrow$ **Est-ce une bonne idée ? NON !**

Certains attributs servent au fonctionnement interne d'une classe. S'il est possible de les modifier à l'extérieur, c'est potentiellement source de problèmes, de bugs.

Modificateur d'accès **private**

Le modificateur d'accès **private**, appliqué à un attribut ou une méthode d'une classe, permet de rendre cette attribut ou cette méthode invisible à l'extérieur de la classe.

Les attributs **width** et **height** sont visibles :

- dans la classe **Rectangle** bien sûr
- mais aussi en dehors de la classe **Rectangle**

$\Rightarrow$ **Est-ce une bonne idée ? NON !**

Certains attributs servent au fonctionnement interne d'une classe. S'il est possible de les modifier à l'extérieur, c'est potentiellement source de problèmes, de bugs.

Modificateur d'accès **private**

Le modificateur d'accès **private**, appliqué à un attribut ou une méthode d'une classe, permet de rendre cette attribut ou cette méthode invisible à l'extérieur de la classe.

Modificateur d'accès **public**

Ce modificateur d'accès **public** permet de rendre un attribut ou une méthode visible partout.

```
1  class Rectangle {
2    private double width;  // private modifier added
3    private double height; // private modifier added
4    private double getSurface() {
5      return width*height;
6    }
7  }
8  class Test {
9    public static void main(String[] args) {
10     Rectangle rect1 = new Rectangle();
11     //rect1.width = 10; => impossible
12     //rect1.height = 10; => impossible
13   }
14 }
```

```java
class Rectangle {
  private double width;  // private modifier added
  private double height; // private modifier added
  private double getSurface() {
    return width*height;
  }
}
class Test {
  public static void main(String[] args) {
    Rectangle rect1 = new Rectangle();
    //rect1.width = 10; => impossible
    //rect1.height = 10; => impossible
  }
}
```

Super ! Parfait !

Mais comment initialiser les attributs d'une classe alors ?

Comment connaitre la valeur des ces attributs ?

Création d'un objet ou instance de classe

Pour créer un objet dont la classe est nommée **TheClass**, il suffit d'utiliser cette syntaxe :

```
TheClass object = new TheClass();
```

Création d'un objet ou instance de classe

Pour créer un objet dont la classe est nommée **TheClass**, il suffit d'utiliser cette syntaxe :

```
TheClass object = new TheClass();
```

TheClass() est le **contructeur par défaut par défaut**

Est-il possible de créer d'autres constructeurs ?

$\Rightarrow$ **Constructeur par défaut**

```
1  public class Rectangle {
2
3    private double width;
4    private double height;
5    // Default constructor
6    public Rectangle() {
7      width = 1.2;  // Init. width
8      height = 6.5; // Init. height
9    }
10
11 }
```

Est-il possible de créer d'autres constructeurs ?

```
1  public class Rectangle {
2
3    private double width;
4    private double height;
5
6    // Default constructor
7    public Rectangle() {
8      width = 1.2;
9      height = 6.5;
10   }
11   // Another constructor
12   public Rectangle(double w, double h) {
13     width = w;
14     height = h;
15   }
16 }
```

Est-il possible de créer d'autres constructeurs ?

```
1  public class Rectangle {
2
3    private double width;
4    private double height;
5
6    // Default constructor
7    public Rectangle() {
8      this(1.2, 6.5)
9    }
10   // Another constructor, another syntax
11   public Rectangle(double width, double height) {
12     this.width = width;
13     this.height = height;
14   }
15 }
```

$\Rightarrow$ Utilisation du mot clé **this**

S'il est nécessaire de connaitre les valeurs de certains des attributs d'une classe à l'extérieur de celle-ci, il est possible de rajouter des accesseurs.

```
1  public class Rectangle {
2
3    private double width;
4    private double height;
5    //...
6
7    //width accessor
8    public double getWidth() {
9      return width;
10   }
11   //height accessor
12   public double getHeight() {
13     return height;
14   }
15 }
```

S'il est nécessaire de modifier les valeurs de certains des attributs d'une classe de l'extérieur de celle-ci, il est possible de rajouter des modificateurs.

```
1  public class Rectangle {
2
3    private double width;
4    private double height;
5    //...
6
7    //width mutator
8    public void setWidth(double width) {
9      this.width = width;
10   }
11   //height mutator
12   public void setHeight(double height) {
13     this.height = height;
14   }
15 }
```

Bien définir la partie visible d'une classe

- toujours mettre les attributs en accès privé
- ajouter des "getters" uniquement si nécessaire
- éviter les "setters" lorsque c'est possible

Bien définir la partie visible d'une classe

- toujours mettre les attributs en accès privé
- ajouter des "getters" uniquement si nécessaire
- éviter les "setters" lorsque c'est possible

Notion d'interface

L'interface d'une classe est l'ensemble des méthodes accessibles depuis l'extérieur de cette classe. C'est un contrat que s'engage à remplir la classe qui **implémente** cette interface.

Programmation

1 La notion d'objet

2 La classe String et les tableaux

3 Structures conditionnelles et boucles

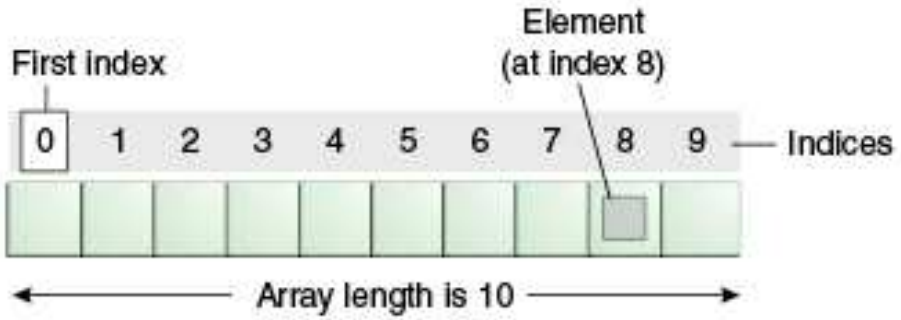
4 Problème

```java
//Create strings
String s1 = new String("ABC");
String s2 = "This string has"
String s3 = new String(23);
String s4 = new String();//Empty string

System.out.println(s1 == s2);//Compare pointers
System.out.println(s1.equals(s2));//Compare strings

//Concatenate
String s4 = s2.concat(s3);
String s5 = s1 + s2 + " chars";

System.out.println(s1.length);//String length
System.out.println(s1.substring(1,5));//Substring

s4 = s4.valueOf(23);//Parse integer
```

Voir ce tutoriel et la JavaDoc.

Défintion

un tableau est un objet conteneur comportant un nombre fixe d'éléments d'un type donné (primitif ou objet).




```
1 //Create an int array of three elements, init. to zero
2 int[] a1 = new int[3];
3 //Initialize elements
4 a1[0] = 12; a1[1] = 13; a1[2] = 14;
5 //Two elements array of Rectangle, init. to null
6 Rectangle[] a2 = new Rectangle[2];
7 //Initialize elements
8 a2[0] = new Rectangle(1,2);
9 a2[1] = new Rectangle(3,4);
```

Programmation

1 La notion d'objet

2 La classe String et les tableaux

3 Structures conditionnelles et boucles

4 Problème

```
1  if (x > 10) {
2
3    System.out.println("x < 10");
4
5  }
```

```
1  if (x < 10) {
2
3     System.out.println("x␣<␣10");
4
5  } else {
6
7     System.out.println("x␣>=␣10");
8
9  }
```

```java
int month = 1;
String monthString;

switch (month) {

  case 1 : monthString = "January";
           break;
  case 2 : monthString = "February";
           break;
  case 3 : monthString = "March";
           break;
  //...
  case 12 : monthString = "December";
            break;
  default : monthString = "Invalid month";
            break

}
System.out.println(monthString);
```

```
1  int n, sum;
2
3  while (n < 100) {
4
5    n++;
6
7    sum = sum + n;
8
9  }
10 System.out.println("Sum : " + sum);
```

```
1  int n, sum;
2
3  do {
4
5    n++;
6
7    sum = sum + n;
8
9  } while (n < 100)
10 System.out.println("Sum : " + sum);
```

Cette boucle est toujours effectuée au moins une fois

```java
int sum;

for (int i = 100; i > 0; i--) {

  sum = sum + i;

}
System.out.println("Sum : " + sum);
```

Programmation

1 La notion d'objet

2 La classe String et les tableaux

3 Structures conditionnelles et boucles

4 Problème

Calcul d'aire. . .

Vous travaillez chez un équipementier sportif. Vous récupérez des résultats d'enregistrements relatifs au transfert de puissance d'une raquette de tennis vers la balle en fonction de la taille du tamis. Vous êtes donc amené à effectuer des calculs d'aires. Votre boss vous demande d'écrire une application qui devra, dans un premier temps, remplir ces exigences :

- l'utilisateur devra pouvoir saisir l'intervalle d'intégration
- l'utilisateur devra pouvoir saisir le nombre de sous intervalles
- la fonction test à intégrer sera la suivante : $f(x) = sin(x)$
- la méthode utilisée sera celle des points milieux
- le code devra être commenté
- des tests pour vérifier l'exactitude des calculs seront prévus

Calcul d'aire. . .

Compte tenu de ce petit cahier des charges :

1. proposez une implémentation objet sous la forme d'un diagramme (classes, attributs et méthodes).
2. Mettez en oeuvre et testez cette conception (n'hésitez pas à vous servir d'internet pour récupérer l'algorithme des points milieux).

Calcul d'aire. . .

Seulement voilà, dans le cadre d'une modélisation, vous ☻ ne souhaitez plus connaitre l'aire de cette fonction test, mais plutôt celles de fonctions impulsionnelles :

$$\frac{1}{1+x^2} \; ; \; 3e^{-2|x|} \; ; \; 4e^{-\pi x^2}$$

3. Proposez les modifications nécessaires.
4. Portez un regard critique sur celles-ci.

Vous rencontrez un consultant qui vous propose une solution sous la forme d'un librairie externe nommée JEXL. Vous décidez de l'utiliser :

5. En quoi cette librairie solutionne-t-elle votre problème ?
6. Téléchargez, installez JEXL et modifiez votre programme

Calcul d'aire. . .

Dans un accès de folie intense, vous souhaitez maintenant avoir les résultats selon trois méthodes d'intégration numériques différentes : points milieux, trapèzes et Sympson.

7. Proposez les modifications nécessaires.
8. Portez un regard critique sur celles-ci.

Vous rencontrez un consultant qui vous propose une solution sous la forme d'un patron de conception nommé "stratégie" :

9. En quoi cela solutionne-t-il votre problème ?
10. Modifiez votre programme pour mettre en oeuvre ce patron de conception.

Le propre du développement logiciel c'est le

CHANGEMENT

Le propre du développement logiciel c'est le

CHANGEMENT

Adopter des principes de développement OO
permet de mieux faire face à ces changements