Comment spécifier les arguments pour `new`

7.2 CONSTRUCTEURS

Rectangle avec constructeur

Code/src/course/object_intro/RectangleDemoBegin.java

File Edit Source Refactor Navigate Search Project Run Window Help

Rectangle.java

```java
package course.object_intro;

public class Rectangle {
    int width;
    int height;

    public int area() {
        return width * height;
    }
}
```

RectangleDemoBegin.java

```java
package course.object_intro;

public class RectangleDemoBegin {

    public static void main(String args[]) {
        Rectangle r1 = new Rectangle();
        r1.height = 4;
        r1.width = 4;

        Rectangle r2 = new Rectangle();
        r2.height = 5;
        r2.width = 6;

        Rectangle r3 = new Rectangle();
        r3.height = 2;
        r3.width = 2;

        int totalArea = r1.area();
        totalArea += r2.area();
        totalArea += r3.area();

        System.out.println(totalArea);

```

Console Problems Debug Shell

<terminated> RectangleDemoBegin [Java Application] C:\Program Files\ojdkbuild\java-11-openjdk-11.0.4-1\bin\javaw.exe (5 nov. 2020 à 19:37:31)

```
50
```

Constructeur

Le constructeur permet de spécifier ce qui se passe lors de la **création** d'un objet et les éventuels paramètres pour la construction.

Définition

Constructeur (2)

- Avec constructeur :

```
class Rectangle{
  int width = 0;
  int height = 0;

  Rectangle(int w, int h){
    width = w;
    height = h;
  }
}
```

```
Rectangle rect = new Rectangle(3, 5);
```

Constructeur (3)

- Jusqu'à maintenant : constructeur par défaut

```
class Rectangle{
  int width;
  int height;
}
```

```
Rectangle r1 = new Rectangle();
r1.width = 3;
r1.height = 5;
```

Constructeur (4), caractéristiques

1. Méthode qui ne retourne RIEN (pas void)
2. Nom du constructeur = nom classe
3. Nombre de constructeurs possibles :
   - aucun (*par défaut*)
   - un
   - plusieurs

L'attribut **this**

- Référence sur l'objet utilisé, optionnel
- Pour enlever les ambiguïtés

```
class Foo{
   int x, int y;

   Foo(int initx, int inity)
   {
      x = initx;
      y = inity;
   }
}
…
Foo f1 = new Foo(20, 30);
```

```
class Foo{
   int x, int y;

   Foo(int x, int y)
   {
      this.x = x;
      this.y = y;
   }
}
…
Foo f1 = new Foo(20, 30);
```

Constructeur dans constructeur ?

```
class Rectangle{
  int width, int height;
  // Constructor A
  Rectangle(int w, int h){
    width = w; height = h;
  }
  // Constructor B
  Rectangle(int side){
    this(side, side);
  }
}
```

```
Rectangle r1 = new Rectangle(3, 5);
Rectangle r2 = new Rectangle(8);
```

```
public private
```

7.3 VISIBILITÉ DES MEMBRES

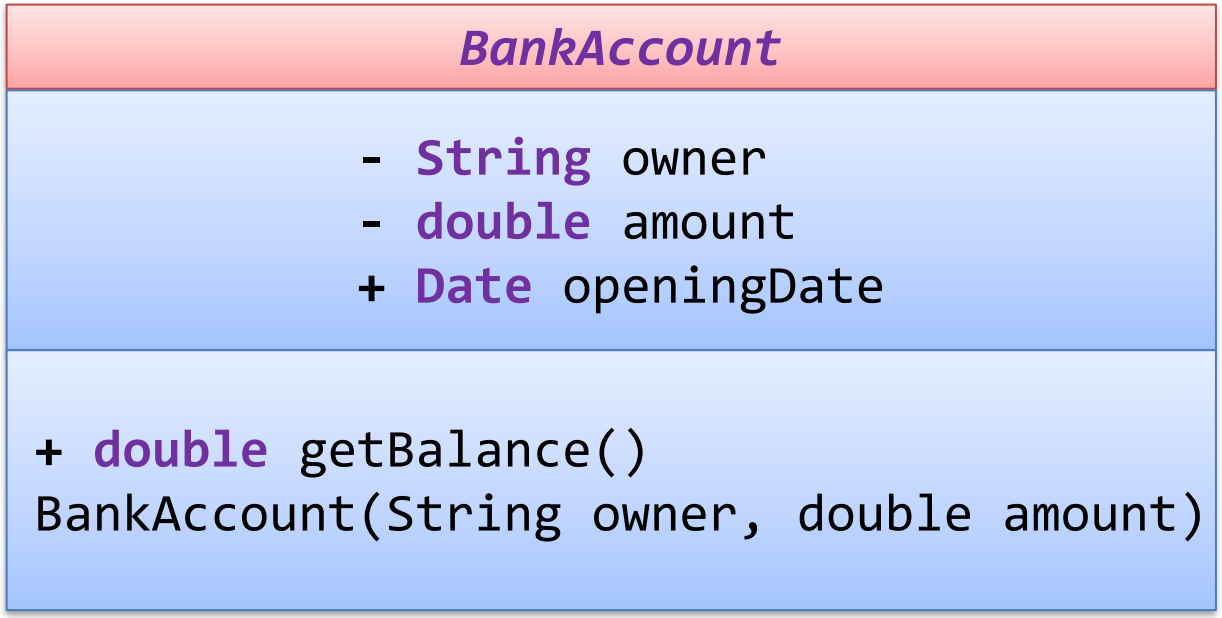
Visibilité des membres

- Deux modificateurs principaux
  - `public`, `private`
- Modifient les règles de visibilité
  -
  -

Visibilité des membres (2)

- Membres **`private`** : accessibles dans la classe uniquement.
- Membres **`public`** : accessibles partout (en dehors de la classe)
- Sans modificateur : presque comme **`public`** (dans le dossier)

Notation UML pour la visibilité



+ signifie `public`
- signifie `private`
Rien signifie visible dans le dossier

Plusieurs classes par fichier

- Normalement, une classe par fichier
- Si plusieurs classes dans fichier, une seule `public`
- Classe → modificateur `public` ou aucun (pour l'instant)
- Règles visibilité classes ≠ différent visibilité membres (même si vocabulaire commun)