Conception objet en Java avec BlueJ une approche interactive

3. Interactions entre objets

Créer des objets coopérants

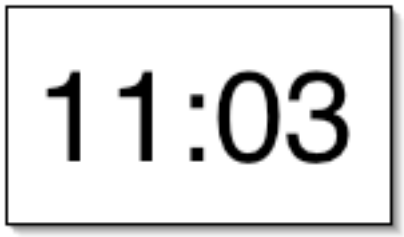
© David J. Barnes, Michael Kölling
Traduction et adaptation française :
© Patrice Moreaux
POLYTECH Annecy-Chambéry – France
Adapté pour le cours de « Programmation objet »
enseigné par Amine Brikci-Nigassa – Université de Tlemcen

Une horloge numérique

Abstraction et modularisation

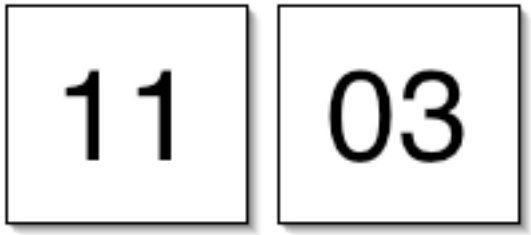
- L'**abstraction** consiste à ignorer les détails des divers éléments d'un problème et se placer à un plus haut niveau.
- La **modularisation** est le processus de division d'un tout en plusieurs parties bien définies, que l'on peut construire et analyser séparément et qui interagissent de manière bien précise.

Modulariser l'affichage numérique



Un afficheur à quatre chiffres ?

Ou
deux afficheurs à deux chiffres ?



Implémentation
NumberDisplay

```
public class NumberDisplay
{
    private int limit;
    private int value;

    //*** Constructeur et
    //    méthodes non reproduits
}
```

Implémentation

ClockDisplay

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

    //*** Constructeur et
    //    méthodes non reproduits.
}
```

Diagramme d'objets

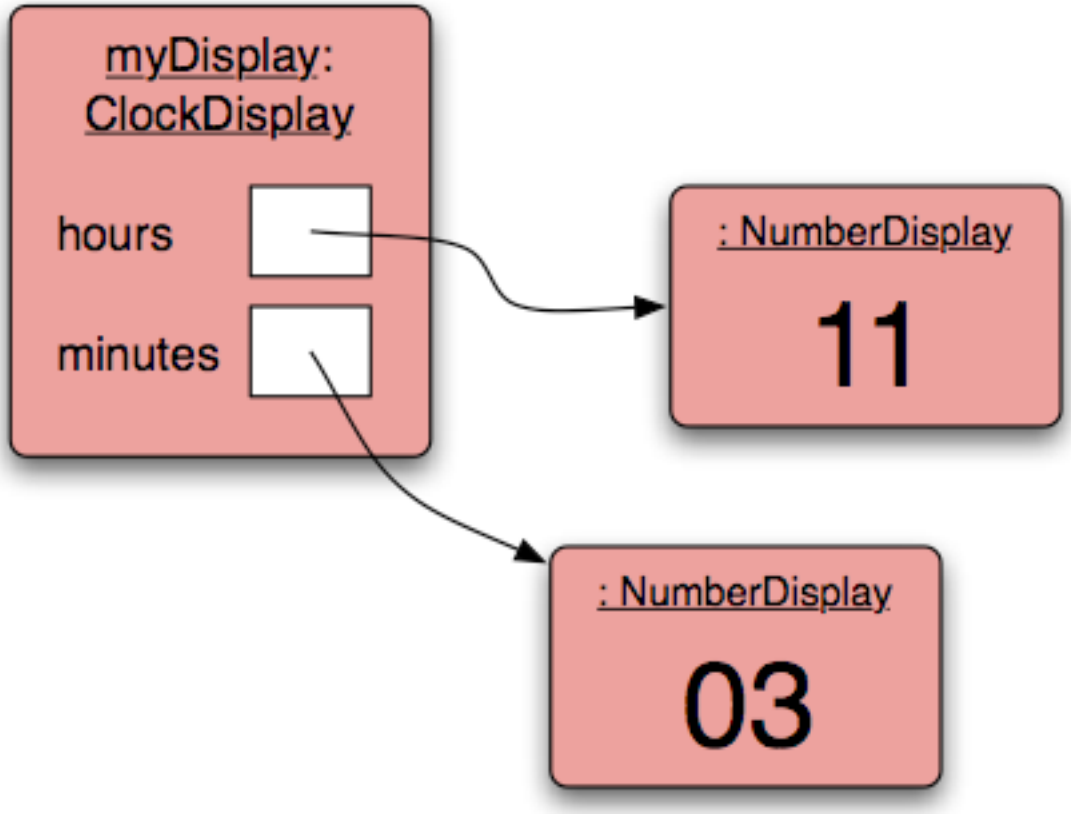
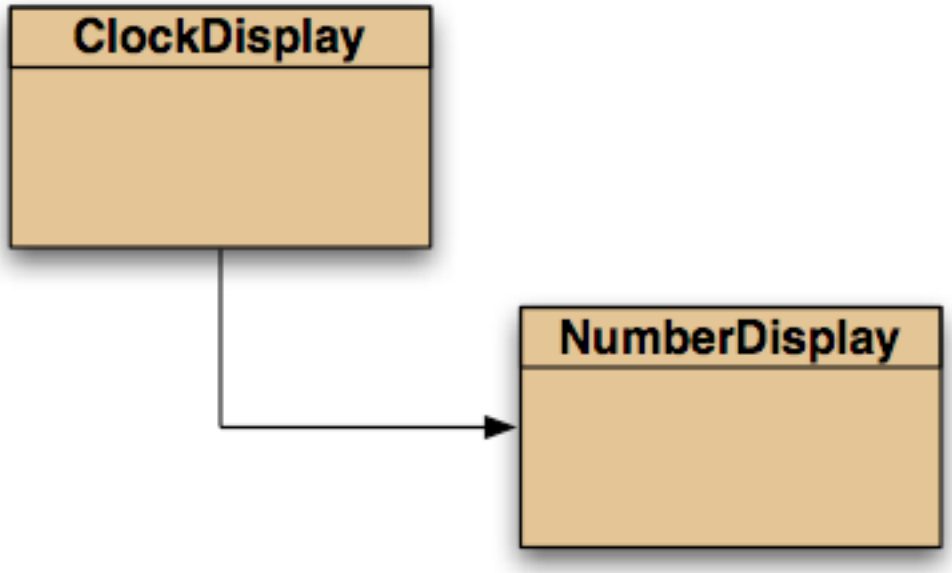


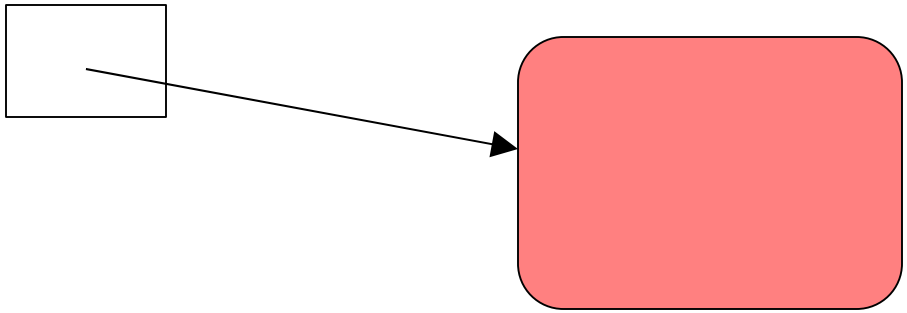
Diagramme de classes



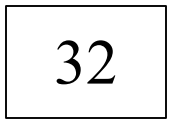
Types primitifs *versus* types objet (1)

```
SomeObject obj;
```

Type objet



```
int i;
```



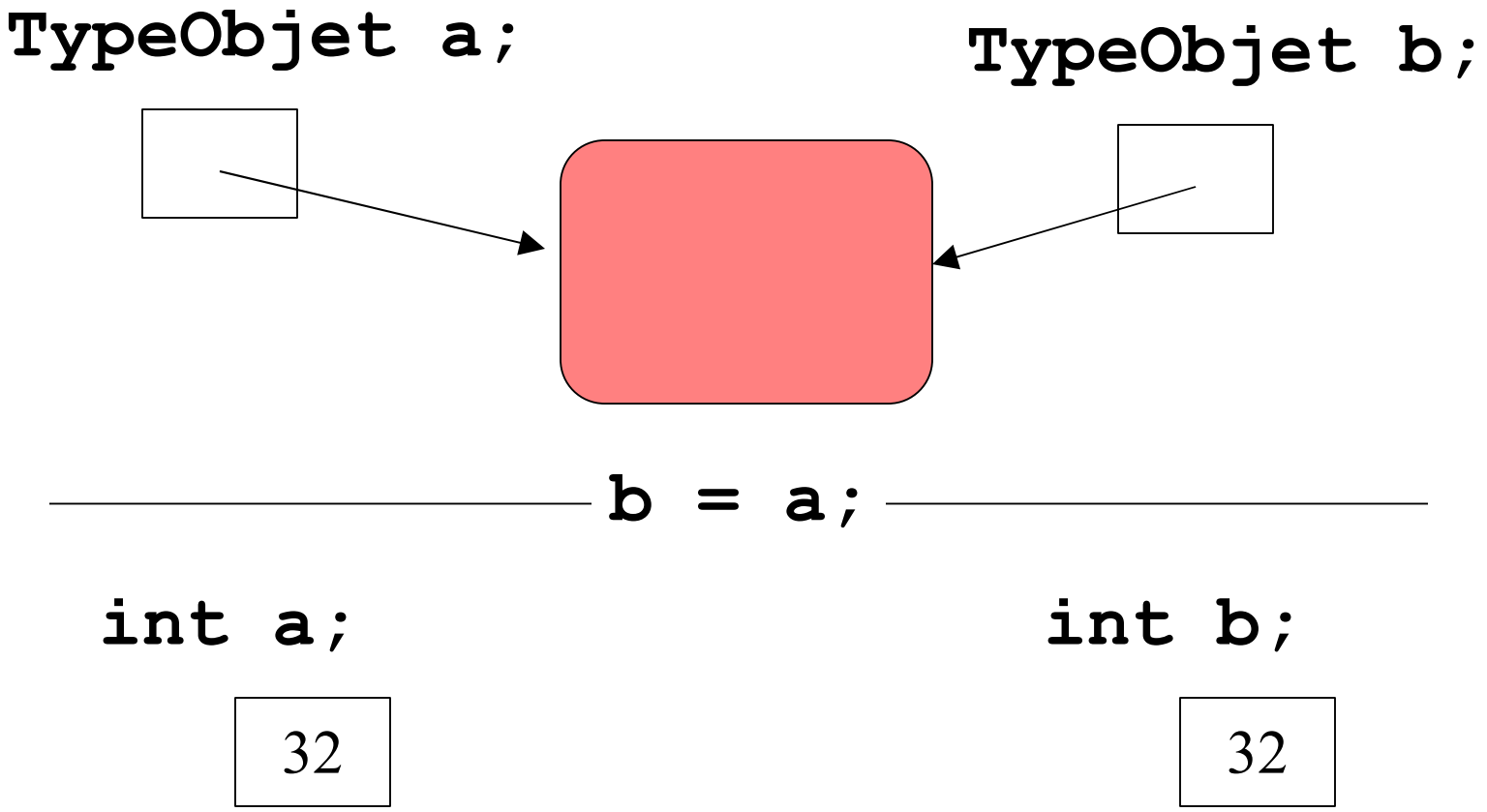
Type primitif

Exercice :
Qu'affiche le code suivant ?

```
int a;
int b;
a = 32;
b = a;
a = a + 1;
System.out.println(b);

Person a;
Person b;
a = new Person("Pierrette");
b = a;
a.changeName("Jean");
System.out.println(b.getName());
```

Types primitifs *versus* types objet (2)



Code source : NumberDisplay

```
public NumberDisplay(int rollOverLimit)
{
    limit = rollOverLimit;
    value = 0;
}

public void increment()
{
    value = (value + 1) % limit;
}
```

L'opérateur modulo

- L'opérateur de « division » (/) renvoie le quotient de la division entière lorsque ses opérandes sont entiers.
- L'opérateur modulo (%) renvoie le reste de la division entière.
- Exemple :
  - 17 / 5 : quotient 3, reste 2.
  - En Java : 17 / 5 = 3 et 17 % 5 = 2.

Questions

- Quelle est la valeur de l'expression (8 % 3) ?
- Quelles sont les valeurs possibles de l'expression (n % 5) ?

Code source : NumberDisplay

```
public String getDisplayValue()
{
    if(value < 10)
        return "0" + value;
    else
        return "" + value;
}
```

Concepts

- Abstraction.
- Modularisation.
- Les classes définissent des types.
- Diagramme de classes.
- Diagramme d'objets.
- Références à objets.

- Types primitifs.
- Types objet.

Objets qui créent des objets

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }
}
```

Objets qui créent des objets

- Les objets peuvent créer d'autres objets en utilisant l'opérateur **`new`**.
- L'opérateur **`new`** fait deux choses :
  – Il crée un nouvel objet de la classe nommée.
  – Il exécute le constructeur de cet objet.
- Si le constructeur de la classe est défini ayant des paramètres, alors les paramètres effectifs doivent être fournis dans l'instruction new :

```
hours = new NumberDisplay(24);
minutes = new NumberDisplay(60);
```

Objets qui créent des objets

dans la classe **`ClockDisplay :`**

```
hours = new NumberDisplay(24);
```

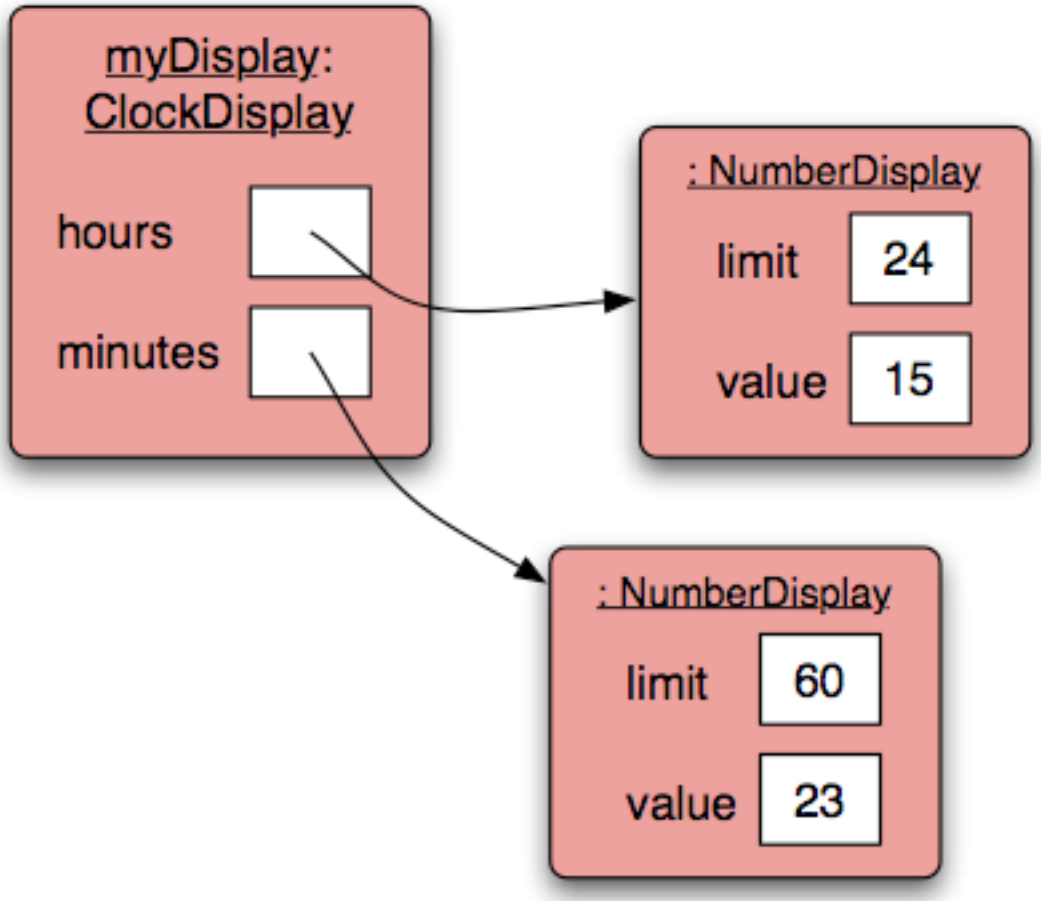
paramètre effectif

dans la classe **`NumberDisplay :`**

```
public NumberDisplay(int rollOverLimit);
```

paramètre formel

Diagramme d'objets de `ClockDisplay`



Appel de méthode

```
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {
        // une heure de plus!
        hours.increment();
    }
    updateDisplay();
}
```

Appel de méthode externe

- Appel de méthodes externes

```
minutes.increment();
```

objet.*nomDeMéthode*(*liste-des-paramètres*)

notation *pointée*

Appel de méthode interne

- Appel de méthodes internes

```
updateDisplay();
```

- Pas besoin de nom de variable.
- **`this`**
  - peut être utilisé comme référence à l'objet appelant, mais guère employé pour les appels de méthodes

```
this.updateDisplay(); //inutile
```

Méthode interne

```
/**
 * Mettre à jour la chaîne interne
 * qui représente l'affichage.
 */
private void updateDisplay()
{
    displayString =
        hours.getDisplayValue() + ":" +
        minutes.getDisplayValue();
}
```

Appels de méthodes

- N.B. : Un appel d'une méthode d'un autre objet du même type sera un appel externe.
- « interne » signifie « cet objet ».
- « externe » signifie « tout autre objet », quel que soit son type.

null

- `null` est une valeur spéciale en Java.
- Tous les champs objets sont initialisés à `null` par défaut.
- Vous pouvez affecter la valeur `null` et comparer avec `null` :

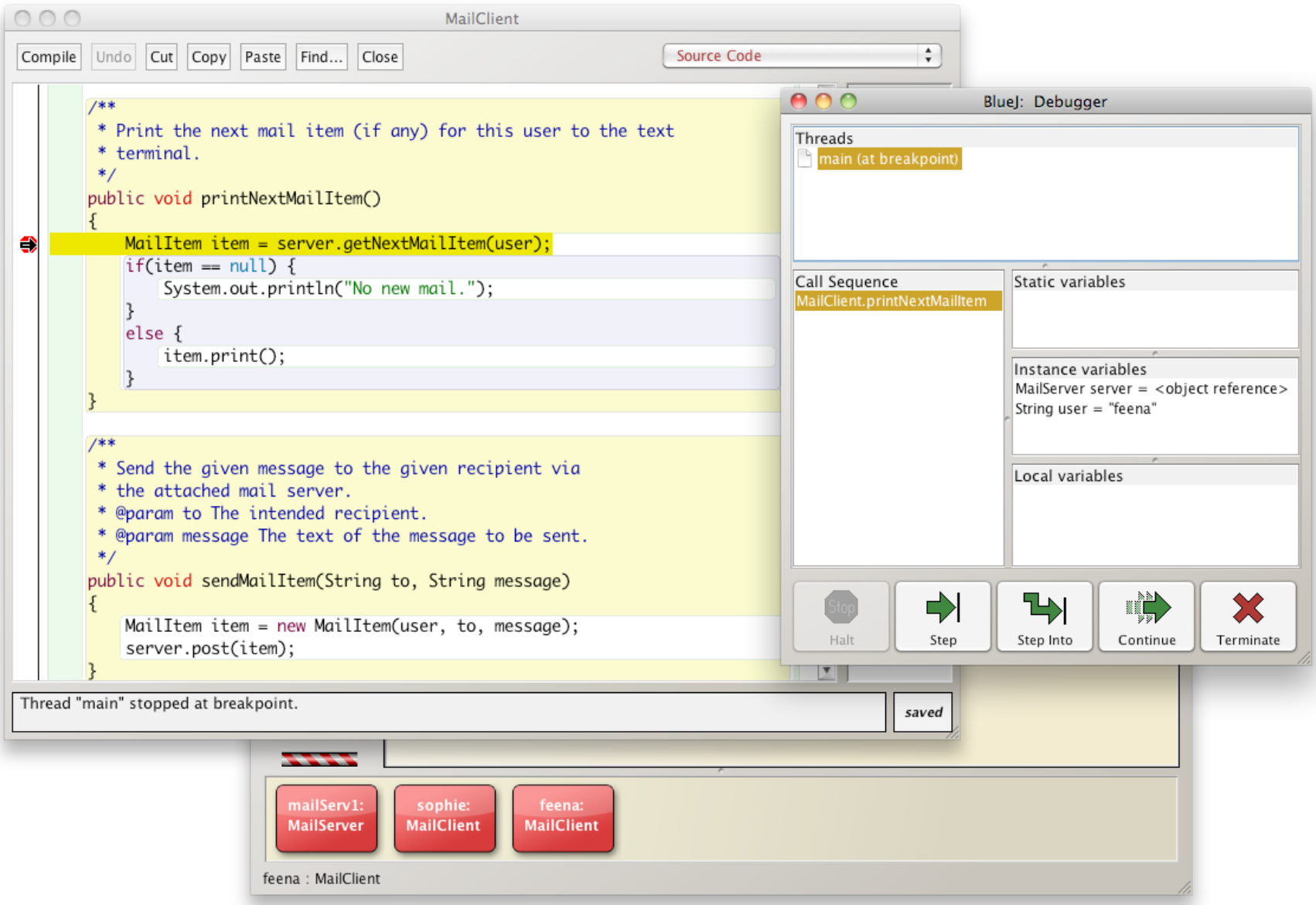
```
private NumberDisplay hours;
if(hours != null) { ... }

hours = null;
```

Le débogueur

- Utile pour obtenir des informations sur le comportement du programme...
- … qu'il y ait une erreur dans le programme ou qu'il n'y en ait pas.
- Définir des points d'arrêt.
- Examiner les variables.
- Exécuter du code pas à pas.

Le débogueur



Appel de méthode (récap.)

- Syntaxe des appels de méthodes internes :

  ***nomDeMéthode*(*liste-de-paramètres*)**

- Syntaxe des appels de méthodes externes :

  ***objet*.*nomDeMéthode*(*liste-de-paramètres*)**

Concepts

- Création d'objet
- Surcharge
- Appel de méthode interne/externe
- Débogueur (metteur au point)

Sommaire général