Cours d'Environnement de Développement

Arnaud Sangnier

Partie 2

Les programmes informatiques sont sujet à des bogues

Exemple de bogues connus

- Entre 1985 et 1987 : problème de dosage dans le Therac 25
- Janvier 1990 : le réseau téléphonique de la compagnie AT & T s'est retrouvé hors service pendant 9 heurs
- Avril 1996 : explosion de la fusée Ariane V
- Bug de l'an 2000 (qui heureusement n'a pas eu de conséquences)
- Août 2003 : blackout au Nord-Est des États-Unis

Nécessité de développer des méthodes pour vérifier le comportement de logiciels

Détecter les bogues est une tâche difficile et coûteuse

D'òu viennent les bogues ?

Un bogue peut-être :

- un non-respect de la spécification du système
- un comportement inattendu non couvert par la spécification

Existe-t-il une méthode automatique et universelle pour garantir l'absence de bogues ?

Non

Comment éviter les bogues ?

- Règles de programmation stricte
- Techniques de programmation
- Méthodologies de développement
- Support de langages de programmation
- Le test
- Les méthodes formelles

Qu'est-ce-que le test ?

- Procédure de vérification (partielle) d'un système informatique
- Trouver un nombre maximum de comportements problématiques du logiciel

Un test est un ensemble de cas à tester éventuellement accompagné d'une procédure d'exécution. Il est lié à un objectif.

Important : Un système ne peut être testé que si l'on peut déterminer le comportement attendu en fonction des conditions auxquelles il est soumis

Un peu de vocabulaire

- **Cas de test** (TC) : une exécution du programme déclenchée par des données de test (DT)
- **Suite de tests** (TS) : un ensemble de données de test
- **Objectif de test** (TO) : comportement de la spécification qu'on veut tester
- **Système sous test** (SUT) : programme à tester

Le but du test est de déterminer si pour tous les DT de TS, le SUT à qui l'on donne entrée DT satisfait TO :

$$\forall DT \in TS, SUT(DS) \models TO$$

Incomplétude du test

Tester complètement un programme est impossible

- Le test ne garantit pas l'absence d'erreurs
- Comment sélectionner une suite de tests parmi tous les tests possibles de façon à détecter le plus d'erreurs possible ?

Utilisation de stratégies de test

De quoi a-t-on besoin ?

- La spécification du programme donnée sous la forme de :
  - une description informelle
  - un ensemble de scénario d'utilisation
  - des diagrammes de séquence
  - un automate
- Le programme sous test
- Une ou plusieurs stratégies de test
- Un critère de sélection de tests
- Un gestionnaire de test (par exemple JUnit)

Test en "boîte blanche"

- Cas de tests générés en partant du code
- Critère de sélection : comment le code est **couvert** par la suite de tests
  - toutes les instructions
  - toutes les branches
  - toutes les conditions ou leur combinaison
  - tous les chemins
  - toutes les définitions de variable
  - tous les chemins entre les définitions de variable et leur utilisation dans un calcul
  - tous les chemins entre les définitions de variable et leur utilisation dans une condition
  - ...

Test en "boîte noire"

Test des fonctionnalités

- Partition des entrées
- Analyse de valeurs limite
- Analyse du comportement
- Analyse des chaînes cause-effet
- Test aléatoire
- ...

Approches pour le test en boîte noire

1. Analyse du domaine des entrées/sortie du système à tester : choix de sous-ensembles **intéressants**
2. Analyse du comportement observable : appliquer les méthodes de test en boîte blanche
3. Heuristiques

Partition des entrées

Diviser le domaine des entrées en un nombre fini de classes tel que le programme réagit pareil pour toutes les valeurs de la classe, donc il suffit de tester qu'une valeur par classe

Stratégie de test

1. Identifier les classes d'équivalence des entrées
   - Sur la base des conditions sur les entrées/sorties
   - En prenant des classes d'entrées valides et invalied
2. Définir un ou quelques cas de test pour chaque classe

Analyse de valeurs limite

Les erreurs se nichent dans les cas limite, donc tester les valeurs aux limites des domaines ou des classes d'équivalence

Stratégie de test

- Tester les bornes des classes d'équivalence
- Tester les bornes du domaine des entrées
- Tester les entrées qui produisent les valeurs aux bornes pour les sorties

JUnit

Test de bon fonctionnement d'applications Java : l'outil JUnit

JUnit

- Open source destiné à tester des applications développées sous Eclipse
- Dans un même projet, classes normales et classes JUnit
- Code JUnit pour tester les autres classes du projet
- JUnit permet de construire un jeu de tests
- Après chaque modification, il est possible de vérifier si l'application passes avec les succès les jeux de tests

Fonctionnement de JUnit

Test de code au moyen d'assertions permettant de tester des situations d'exécutions

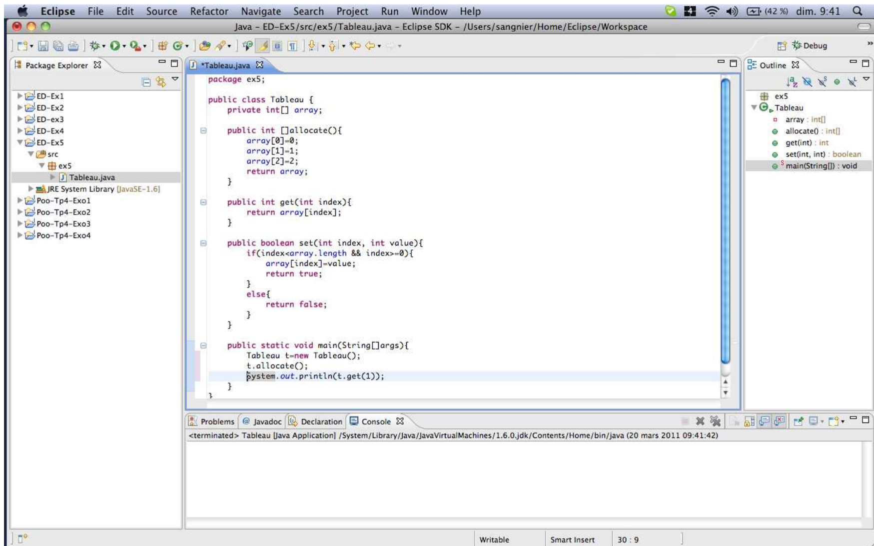
Exemple d'assertions

- `assertEquals(a,b)` teste si `a` est égal à `b` (`a` et `b` sont soit des valeurs primitives ou bien des objets équipés d'une méthode `equals`)
- `assertFalse(a)` teste si `a` est faux (`a` étant une valuer booléenne)
- `assertNotNull(a)` teste si `a` est différent de `null`
- `assertNotSame(a,b)` teste si `a` et `b` ne se réfèrent pas au même objet
- `assertNull(a)` teste si `a` est `null`
- `assertSame(a,b)` teste si `a` et `b` se réfèrent au même objet
- `assertTrue(a)` teste si `a` est vrai (`a` étant une valuer booléenne)

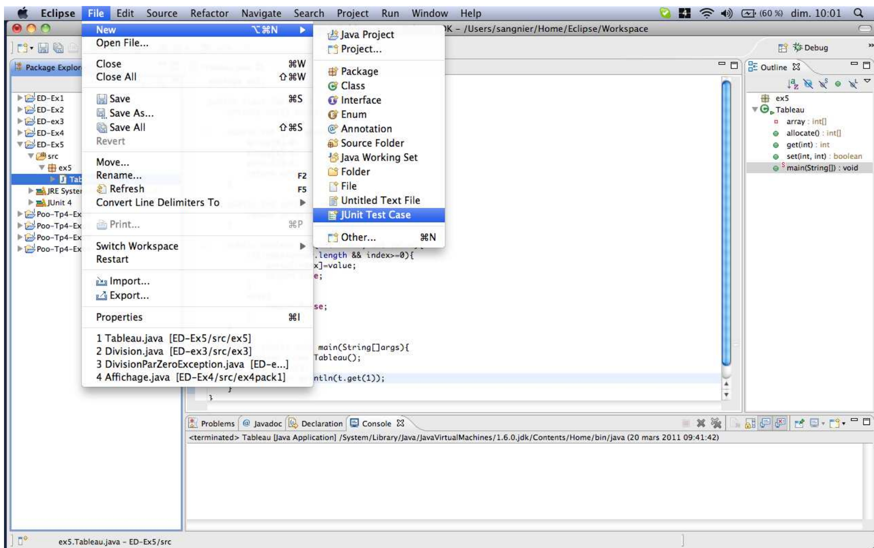
Un exemple de programme à tester

```
package ex5;
public class Tableau{
  private int[] array;
  public int []allocate(){
    array[0]=0;
    array[1]=1;
    array[2]=2;
    return array;
  }
  public int get(int index){
    return array[index];
  }
  public boolean set(int index, int value){
    if(index<array.length && index>=0){
      array[index]=value;
      return true;}
    else{return false;}
  }
```

Un exemple de programme à tester



Création d'une classe de tests



Création d'une classe de tests

New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

(•) New JUnit 3 test ( ) New JUnit 4 test

Source folder: ED-Ex5/src [Browse...]

Package: ex5 [Browse...]

Name: TableauTest

Superclass: junit.framework.TestCase [Browse...]

Which method stubs would you like to create?

- [ ] setUpBeforeClass() [ ] tearDownAfterClass()
- [x] setUp() [x] tearDown()
- [ ] constructor

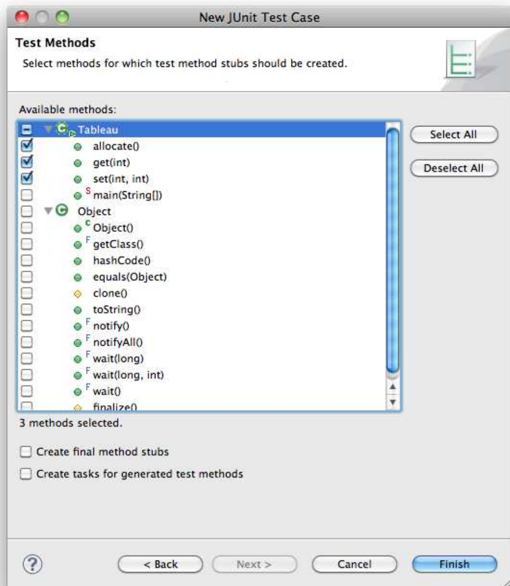
Do you want to add comments? (Configure templates and default value here)

- [ ] Generate comments

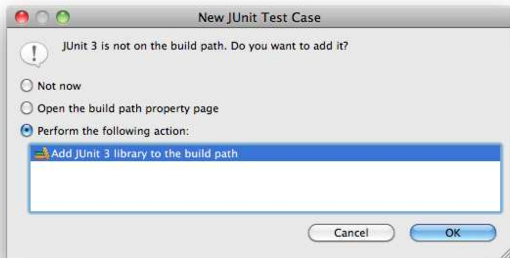
Class under test: ex5.Tableau [Browse...]

[< Back] [Next >] [Cancel] [Finish]

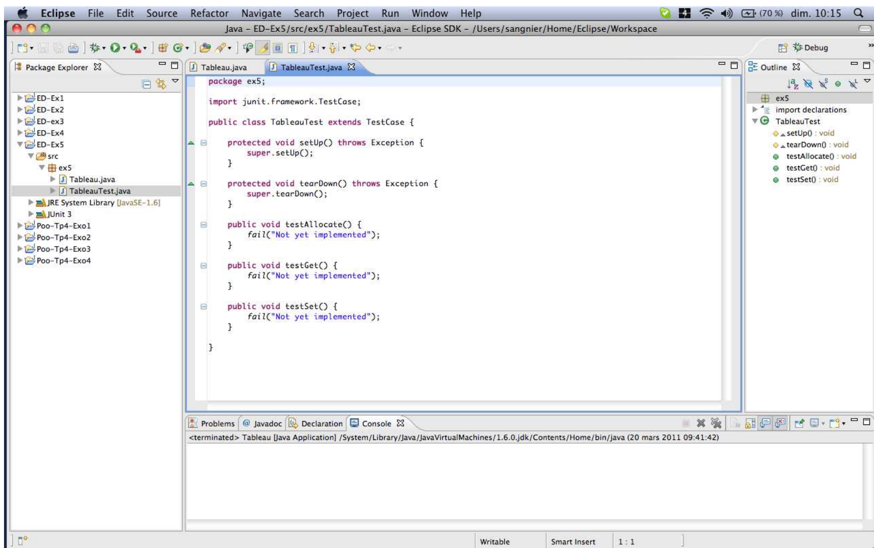
Création d'une classe de tests



Création d'une classe de tests



Création d'une classe de tests



Écrire la classe de tests

- Insérer du code dans le squelette généré par Eclipse
- Code qui appellera les méthodes à tester (ici `set`, `get` et `allocate`)
- Une instance de la classe à tester (`Tableau`) doit être créé par exemple :
  - Déclaration d'un champ `Tableau objetTeste;`
  - Et dans la méthode `setUp()`, faire `objetTeste=new Tableau();`

Un test pour chaque méthode

- Pour l'allocation du tableau :

```
public void testAllocate(){
  assertNotNull(objetTeste.allocate());
}
```

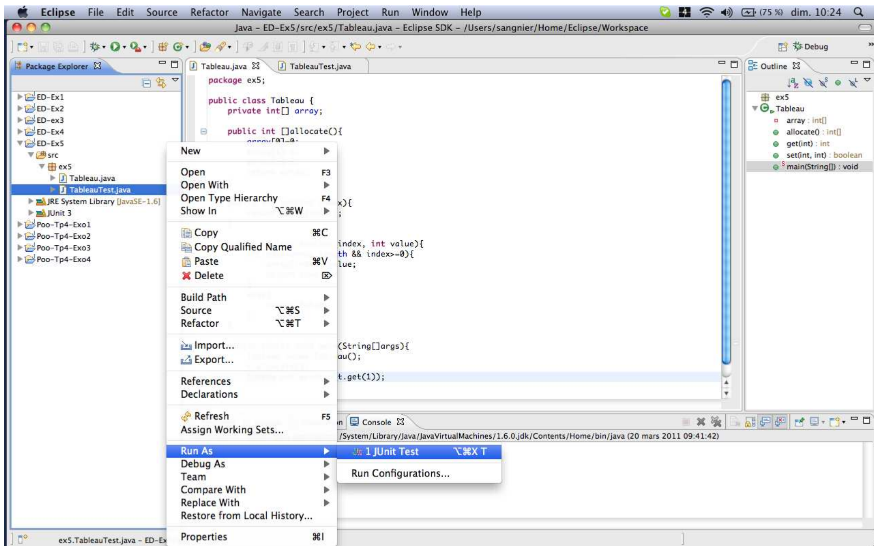
- Pour la lecture de valeurs dans le tableau :

```
public void testGet(){
  assertEquals(objetTeste.get(1),1);
}
```

- Pour l'écriture de valeurs dans le tableau :

```
public void testSet(){
  assertTrue(objetTeste.set(2),3);
}
```

Lancer le test



Analyse des résultats du test

Ici le test révèle un problème

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help

Java – ED-Ex5/src/ex5/Tableau.java – Eclipse SDK – /Users/sangnier/Home/Eclipse/Workspace

Package Explor | JUnit

Finished after 0,011 seconds

Runs: 3/3 Errors: 2 Failures: 1

- ex5.TableauTest [Runner: JUnit 3] (0,00
  - testAllocate (0,001 s)
  - testGet (0,000 s)
  - testSet (0,000 s)

Failure Trace

java.lang.NullPointerException
at ex5.Tableau.allocate(Tableau.java:7)
at ex5.TableauTest.testAllocate(TableauTe

Tableau.java | TableauTest.java

```java
package ex5;

public class Tableau {
    private int[] array;

    public int []allocate(){
        array[0]=0;
        array[1]=1;
        array[2]=2;
        return array;
    }

    public int get(int index){
        return array[index];
    }

    public boolean set(int index, int value){
        if(index<array.length && index>=0){
            array[index]=value;
            return true;
        }
        else{
            return false;
        }
    }

    public static void main(String[]args){
        Tableau t=new Tableau();
        t.allocate();
        System.out.println(t.get(1));
    }
}
```

Outline

- ex5
- Tableau
  - array : int[]
  - allocate() : int[]
  - get(int) : int
  - set(int, int) : boolean
  - main(String[]) : void

Problems | Javadoc | Declaration | Console

<terminated> TableauTest [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 10:25:28)

Modification de l'exemple

- Le premier test échoue à cause de la fonction allocate :

```
public int []allocate(){
    array[0]=0;
    array[1]=1;
    array[2]=2;
    return array;}
```

- Le Tableau n'est jamais alloué
- Pour résoudre le problème :
  - Rajout de la ligne `array=new int[3];` au début de la fonction
- Ensuite on recommence le test

Nouvelle exécution du test

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help

Java - ED-Ex5/src/ex5/TableauTest.java - Eclipse SDK - /Users/sangnier/Home/Eclipse/Workspace

Package Explorer | JUnit

Finished after 0,022 seconds

Runs: 3/3 Errors: 2 Failures: 0

- ex5.TableauTest [Runner: JUnit 3] (0,000 s)
  - testAllocate (0,000 s)
  - testGet (0,000 s)
  - testSet (0,000 s)

Failure Trace

java.lang.NullPointerException
at ex5.Tableau.get(Tableau.java:15)
at ex5.TableauTest.testGet(TableauTest.java:23)

Tableau.java | TableauTest.java

```java
package ex5;

import junit.framework.TestCase;

public class TableauTest extends TestCase {

    Tableau objetTeste;

    protected void setUp() throws Exception {
        super.setUp();
        objetTeste=new Tableau();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testAllocate() {
        assertNotNull(objetTeste.allocate());
    }

    public void testGet() {
        assertEquals(objetTeste.get(1),1);
    }

    public void testSet() {
        assertTrue(objetTeste.set(2),3);
    }

}
```

Outline

- ex5
- import declarations
- TableauTest
  - objetTeste : Tableau
  - setUp() : void
  - tearDown() : void
  - testAllocate() : void
  - testGet() : void
  - testSet() : void

Problems | Javadoc | Declaration | Console

<terminated> TableauTest [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 11:02:51)

Encore des erreurs ?

Que se passe-t-il pour les méthodes `get` et `set` ?

- Les tests JUnit sont effectués de façon indépendante
- Le test de la méthode `allocate` appelle bien la méthode `allocate`, cette méthode n'est cependant pas appelée par les deux autres tests
- Il faut donc initialiser le tableau pour les deux autres tests
- Pour cela, on rajoute l'appel `objetTeste.allocate()` au début des deux autres tests
- On relance ensuite le test

Suppression des erreurs

```
package ex5;

import junit.framework.TestCase;

public class TableauTest extends TestCase {

    Tableau objetTeste;

    protected void setUp() throws Exception {
        super.setUp();
        objetTeste=new Tableau();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testAllocate() {
        assertNotNull(objetTeste.allocate());
    }

    public void testGet() {
        objetTeste.allocate();
        assertEquals(objetTeste.get(1),1);
    }

    public void testSet() {
        objetTeste.allocate();
        assertTrue(objetTeste.set(2,3));
    }
}
```

Finished after 0,022 seconds

Runs: 3/3 Errors: 2 Failures: 0

- ex5.TableauTest [Runner: JUnit 3] (0,000 s)
  - testAllocate (0,000 s)
  - testGet (0,000 s)
  - testSet (0,000 s)

Failure Trace

```
java.lang.NullPointerException
at ex5.Tableau.get(Tableau.java:15)
at ex5.TableauTest.testGet(TableauTest.java:23)
```

<terminated> TableauTest [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 11:02:51)

Syntax error, insert ")" to complete Expression — Writable — Smart Insert — 29 : 39

Nouvelle exécution du test

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help

Java - ED-Ex5/src/ex5/TableauTest.java - Eclipse SDK - /Users/sangnier/Home/Eclipse/Workspace

Package Explorer | JUnit

Finished after 0,005 seconds

Runs: 3/3 Errors: 0 Failures: 0

ex5.TableauTest [Runner: JUnit 3] (0,000 s)

Failure Trace

Tableau.java | TableauTest.java

```
package ex5;

import junit.framework.TestCase;

public class TableauTest extends TestCase {

    Tableau objetTeste;

    protected void setUp() throws Exception {
        super.setUp();
        objetTeste=new Tableau();
    }

    prot                                  tion {

    }

    public void testAllocate() {
        assertNotNull(objetTeste.allocate());
    }

    public void testGet() {
        objetTeste.allocate();
        assertEquals(objetTeste.get(1),1);
    }

    public void testSet() {
        objetTeste.allocate();
        assertTrue(objetTeste.set(2,3));
    }
}
```

Tableau ex5.TableauTest.objetTeste

Outline

- ex5
- import declarations
- TableauTest
  - objetTeste : Tableau
  - setUp() : void
  - tearDown() : void
  - testAllocate() : void
  - testGet() : void
  - testSet() : void

Problems | Javadoc | Declaration | Console

<terminated> TableauTest [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 11:09:28)

Debug

Débogage

- JUnit ne teste que la conformité à un jeu de tests donné
- Si un problème plus profond se pose, il faut **déboguer** le code
- Eclipse offre de nombreuses possibilités pour le débogage de code, comme par exemple :
  - Détection d'erreur logique
  - Création de configurations de lancement du débogueur
  - Modification de code durant une session de débogage
  - Arrêt de l'exécution d'un programme en boucle infinie

Qu'est-ce-qu'un débogueur ?

En prenant comme entrée un exécutable E, le débogueur permet :

- d'exécuter pas à pas E
- d'exécuter E de façon sélective
- d'inspecter l'environnement de E (variables, pile, tas,...)
- de rejouer facilement les exécutions précédentes de E
- d'afficher dans un format proche du langage source le code machine de E

Programmer et compiler pour déboguer

Programmation

- Ecrire les instructions sur des lignes séparées
- Appeler les méthodes sur des lignes séparées
- Donner des noms de variables facilement identifiables

Compilation

- Ajouter les informations utiles au débogueur si nécessaire (par exemple en C `gcc -c`). JDT le fait de façon implicite

Rappels JVM (Java Virtual Machine)

La machine virtuelle de Java s'articule autour de 4 parties :

1. Les registres (pc (*program counter* et 3 registres de pile : optop, frame et vars)
2. La pile de contextes d'appels de méthodes (*stackframe*) qui contient :
   - les paramètres d'appel
   - la valeur de retour
   - les variables locales
   - les paramètres des opérations
3. le tas *garbage-collecté*
4. la zone des méthodes qui contient leur *byte-code*

Les points d'arrêt (breakpoints)

Un point d'arrêt suspend l'exécution du programme au point de contrôle du programme auquel il est associé

États des points d'arrêt

- activé et éligible
- activé mais pas éligible
- activé et sans futur
- désactivé
- mort

Types de points d'arrêt

- **général** (*breakpoint*) : activé si le point de contrôle correspondant est éligible
- **supervision** (*watchpoint*) : activé si un changement des données est éligible
- **conditionnel** (*condition point*) : activé si une condition booléenne devient vrai à un point de contrôle
- **d'événement** (*catch point*) : activé si un événement (exception, signal) a lieu

Un nouvel exemple

Calcul de factorielle récursif

```
package ex6;
public class Fact {
  public static void main(String [] args){
    System.out.println(factorielle(6));
  }
  public static int factorielle(int val){
    if(value==0) return val;
    else return(value * factorial(val-1));
  }
}
```

Que donne l'exécution de ce programme ?

Mauvais comportement car $6! = 720$

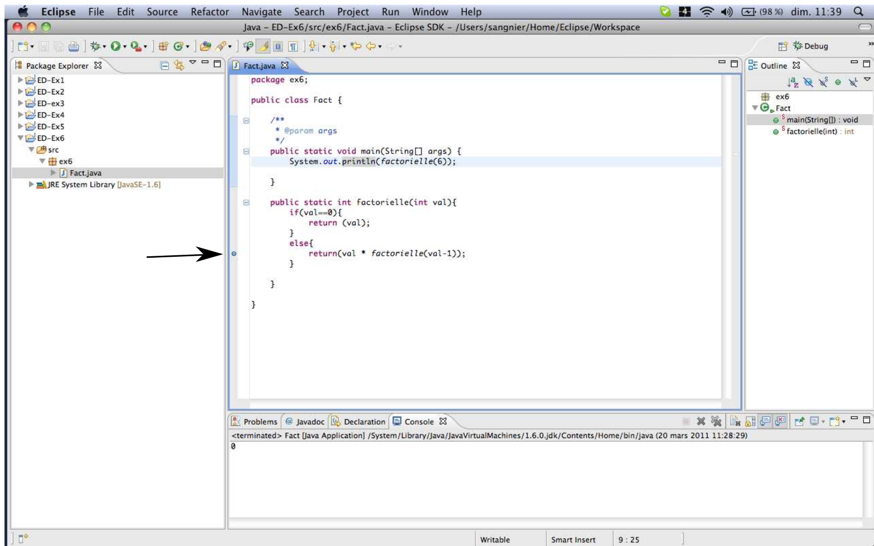
```java
package ex6;

public class Fact {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println(factorielle(6));

    }

    public static int factorielle(int val){
        if(val==0){
            return (val);
        }
        else{
            return(val * factorielle(val-1));
        }

    }

}
```

```
<terminated> Fact [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 11:28:29)
0
```

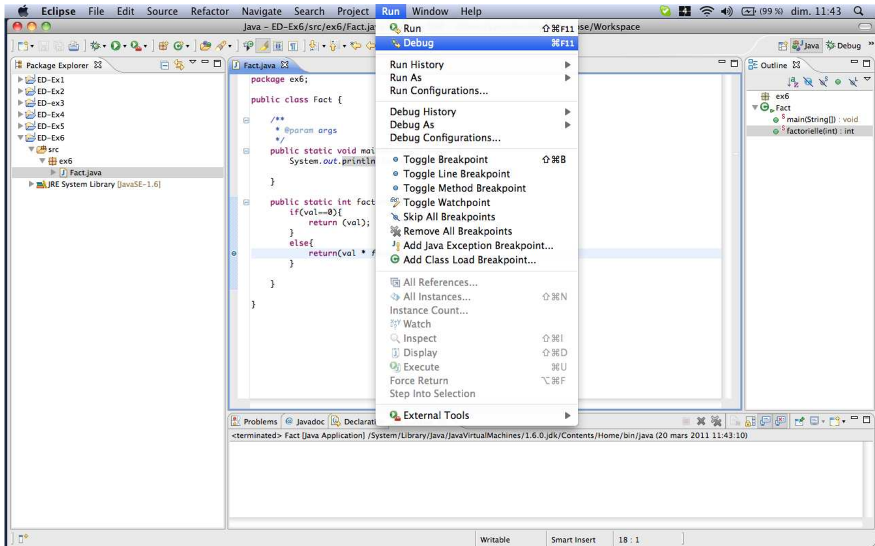
Instauration de points d'arrêt

- Le mauvais comportement du programme n'a levé aucune exception
- Il s'agit d'un problème logique dans l'écriture de l'algorithme
- On va examiner le code en cours d'exécution
- Pour cela, on va suspendre l'exécution à un endroit précis au moyen d'un point d'arrêt (*breakpoint* en anglais)
- Pour mettre un point d'arrêt, il faut double-cliquer dans la marge à gauche de la ligne souhaitée (une boule bleue apparaît alors en face de cette ligne)
- Pour retirer le point d'arrêt, il suffit de double-cliquer dessus

Mettre un point d'arrêt dans le programme

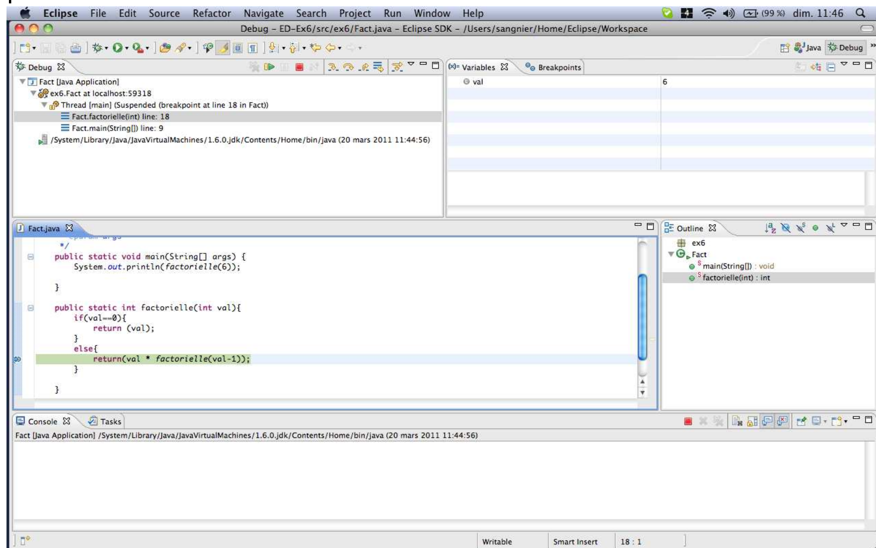


Pour déboguer le code



La perspective Debug

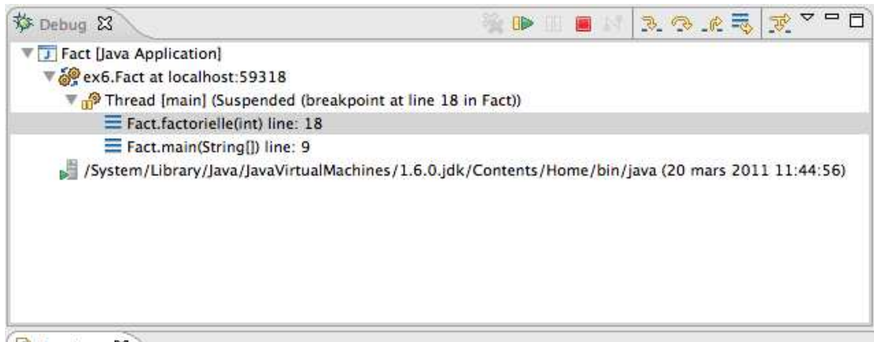
Le programme est pour l'instant arrêté sur la ligne marquée par le point d'arrêt



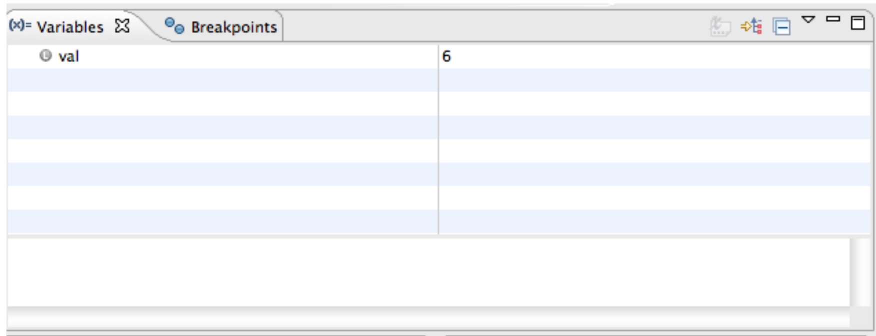
À propos de la perspective debug

- Dans la fenêtre en haut à gauche :
  - Affichage des contextes d'exécution (précédés de 3 petites barres horizontales)
  - Dans la barre de titre, différents boutons de contrôle qui permettent en particulier de :
    1. Reprendre l'exécution du code
    2. Interrompre temporairement l'exécution du code
    3. Terminer l'application (arrêt du débogage)
- Dans la fenêtre en haut à droite :
  - Vue variables : pour examiner le contenu des variables locales
  - Vue points d'arrêts : pour gérer les points d'arrêt du système
  - Vue expressions : permet d'évaluer des expressions

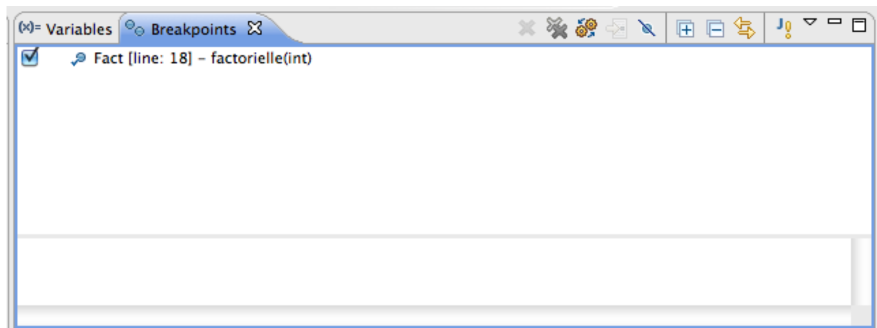
La vue Debug



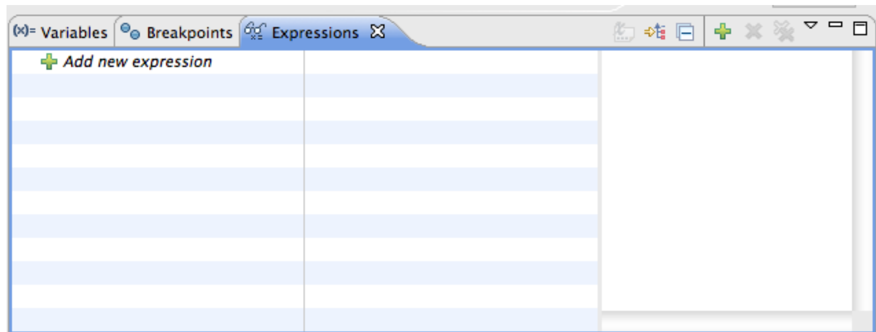
La vue Variables



La vue Breakpoints



La vue Expressions

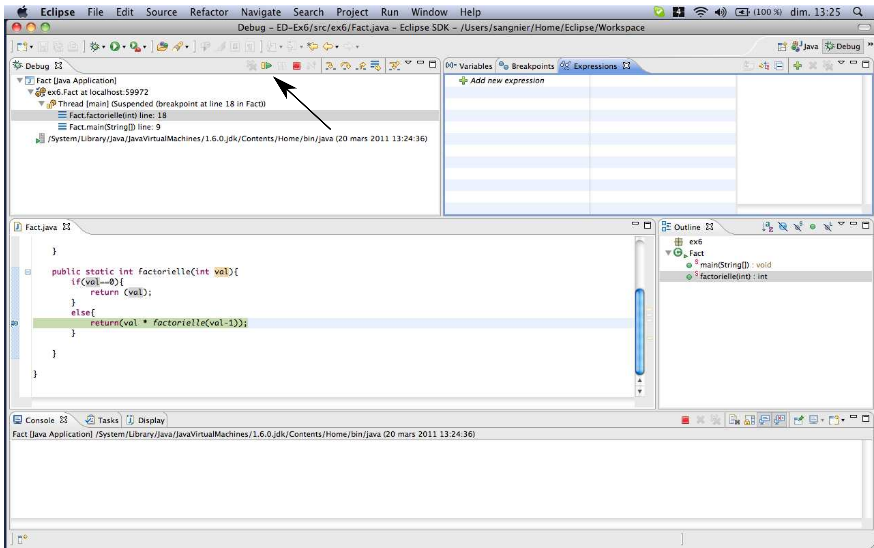


Exécution du code pas à pas

Différents modes possibles :

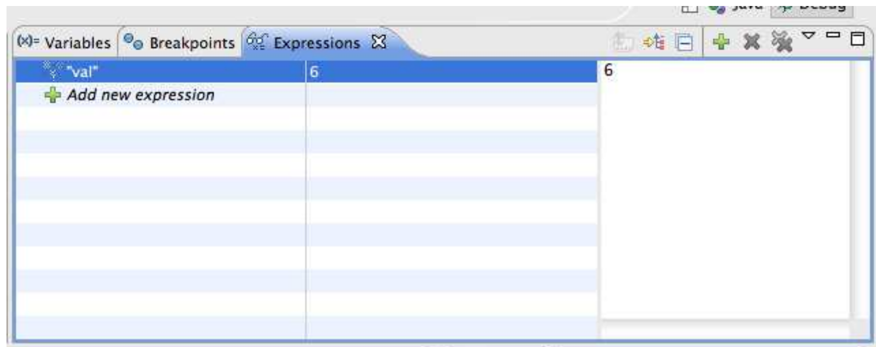
- Avancer d'un pas avec filtres : permet d'entrer dans le code d'une fonction avec l'aide de filtres déterminés
- Avancer d'un pas avec entrée : exécute l'instruction sélectionnée, s'il s'agit d'un appel de fonction, on entre dans le code de celle-ci
- Avancer d'un pas sans entrée : exécute l'instruction sélectionnée, s'il s'agit d'un appel de fonction, celle-ci est exécutée globalement
- Exécuter jusqu'à l'instruction de retour

Reprise de l'exécution jusqu'au point d'arrêt suivant



Visualisation permanente de la variable `val`

On suit cette variable dans la vue Expressions



Compteur d'occurences de points d'arrêt

- Le programme est censé appeler 6 fois la méthode `factorielle`
- Il faudrait donc cliquer 6 fois sur le bouton Reprise (*Resume*)
- Avec un compteur initialisé à n, le point d'arrêt ne sera actif qu'au n-ième passage sur celui-ci

Création d'un compteur de points d'arrêt

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help (100 %) dim. 13:35

Debug – ED-Ex6/src/ex6/Fact.java – Eclipse SDK – /Users/sangnier/Home/Eclipse/Workspace

Java Debug

Debug

Fact [Java Application]
ex6.Fact at localhost:59974
Thread [main] (Suspended (breakpoint at line 18 in Fact))
Fact.factorielle(int) line: 18
Fact.main(String[]) line: 9
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 13:30:26)

Variables Breakpoints Expressions

"val"	6
Add new expression	

6

Fact.java

```
    }

    public static int factorielle(int val){
        if(val==0){
            return (val);
        }
        else{
              ... ctorielle(val-1));
```

Toggle Breakpoint
Disable Breakpoint
Go to Annotation ⌘1
Add Bookmark...
Add Task...
✓ Show Quick Diff ^⇧Q
Show Annotation
Show Line Numbers
Folding
Preferences...
Breakpoint Properties...

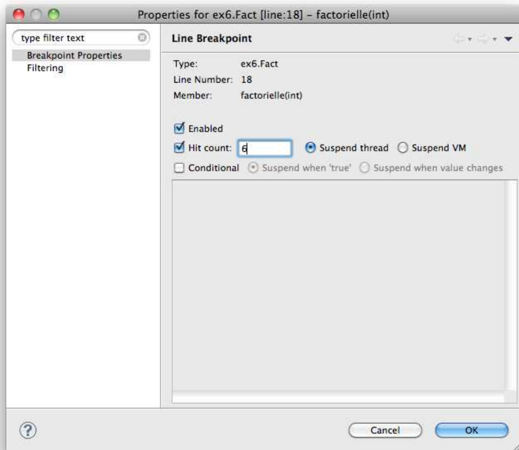
Outline

ex6
Fact
main(String[]) : void
factorielle(int) : int

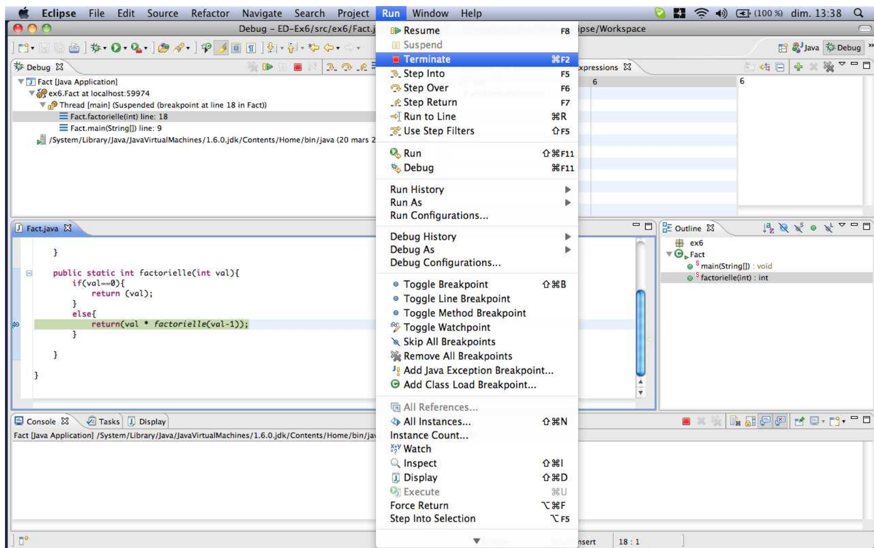
va/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 13:30:26)

Writable Smart Insert 18 : 1

Création d'un compteur de points d'arrêt

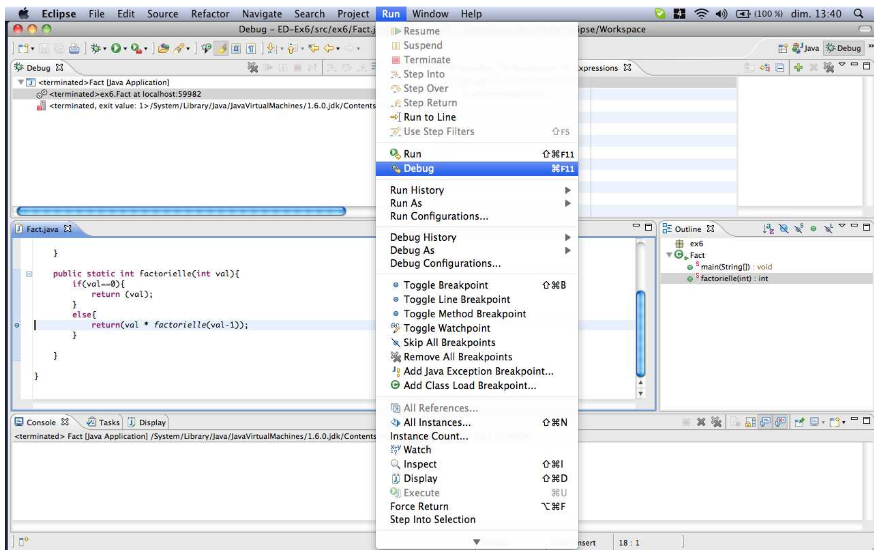


On termine la session courante de débogage



On commence une nouvelle session

Celle-ci se bloquera au 6-ème passage sur le point d'arrêt



On commence une nouvelle session

Ensuite on exécute pas à pas pour comprendre le problème

Debug – ED-Ex6/src/ex6/Fact.java – Eclipse SDK – /Users/sangnier/Home/Eclipse/Workspace

```
Fact [Java Application]
  ex6.Fact at localhost:59984
    Thread [main] (Suspended (breakpoint at line 18 in Fact))
      Fact.factorielle(int) line: 18
      Fact.factorielle(int) line: 18
      Fact.factorielle(int) line: 18
      Fact.factorielle(int) line: 18
      Fact.factorielle(int) line: 18
      Fact.factorielle(int) line: 18
      Fact.main(String[]) line: 9
  /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 13:41:40)
```

Expressions: "val" = 1

```java
    }

    public static int factorielle(int val){
        if(val==0){
            return (val);
        }
        else{
            return(val * factorielle(val-1));
        }

    }

}
```

Console: Fact [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (20 mars 2011 13:41:40)

Résolution du problème

La méthode `factorielle` :

```
public static int factorielle(int val){
  if(value==0) return val;
  else return(value * factorial(val-1));
}
```

devrait avoir la forme :

```
public static int factorielle(int val){
  if(value==0) return 1;
  else return(value * factorial(val-1));
}
```

Exécution après correction