Introduction à la programmation en Java

U.F.R. Sciences de Nice

Licence Math-Info 2006-2007

Module L1I1

Frédéric MALLET

Jean-Paul ROY

Où en sommes-nous ?

- Nous savons rédiger le texte d'une classe d'objets, avec ses **champs**, ses **constructeurs**, ses **méthodes**.
- Nous pouvons exprimer qu'une méthode a ou n'a pas de résultat.
- Nous savons utiliser deux collections particulières :
  - les listes d'objets numérotés (**`ArrayList`**)
  - les tableaux à une ou deux dimensions
- Nous savons construire et transformer du texte (**`String`**)
- Nous savons dessiner avec une tortue
- Nous savons programmer par récurrence

COURS 12

Java sans Bluej

Ligne de commande entrées/sorties fichiers

Hello, world !

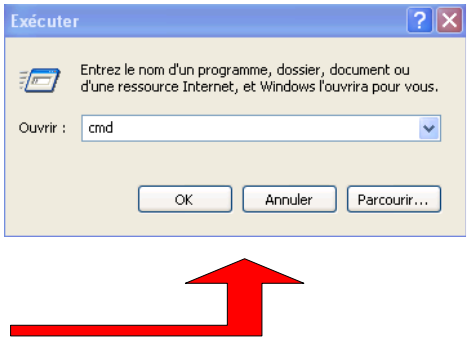
- En Java, on doit toujours avoir au moins une classe public, principale, c'est-à-dire qui contient une méthode **`main`**.
- Le fichier doit avoir le même nom que la classe
- La signature de la méthode **`main`** est imposée !
- Exemple : **`Hello.java`**

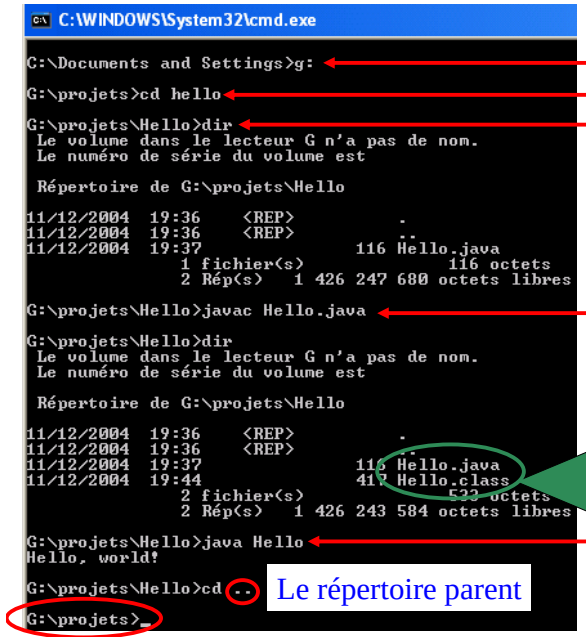
```
public class Hello {
   static public void main(String[] args) {
      System.out.println("Hello, world!");
   }
}
```

- Pour l'utiliser, le **`Java Development Kit`** propose deux outils :
  - Un **compilateur** **`javac`** qui produit un fichier **`.class`** (*bytecode*)
  - Un **interpréteur** **`java`** qui exécute le fichier **`Hello.class`**

La ligne de commande

- Tous les systèmes d'exploitation (Windows, MacOS-X, Linux) proposent une **ligne de commande**.
- Sous Windows :





Changer de lecteur g:, e:

Changer de répertoire

Lister les fichiers

Compiler la classe **`Hello`**

Exécuter la méthode **`main`** de la classe **`Hello`**

Le répertoire parent

Le répertoire courant

Les paramètres de la ligne de commande

- Le tableau `args` contient les paramètres de la méthode `main`.

```
public class Hello {
  static public void main(String[] args) {
    if (args.length == 0)
      System.out.println("Hello, world!");
    else
      System.out.println("Hello, " + args[0]);
  }
}
```

- Le tableau **`args`** est rempli par Java avec les mots qui suivent la commande **`java Hello`**

`>java Hello`	affiche toujours	**`Hello, world!`**
`>java Hello toto`	affiche	**`Hello, toto!`**

Les paramètres sont de type `String`

- Attention, même si on passe une valeur numérique

`>java Hello 12`

- **`args[0]`** est la chaîne de caractères "12", pas l'entier 12
- On utilise la classe **`Integer`** pour faire la conversion

```
public class Somme {
   static public void main(String[] args) {
      int somme = 0;
      for (int i = 0; i<args.length; i++) {
         somme += Integer.parseInt(args[i]);
      }
      System.out.println("Somme =" + somme);
   }
}
```

```
>javac Somme.java
>java Somme 2 8 25 -4 6
Somme=37
```

Les exceptions

Définition

- Le mécanisme d'exceptions permet d'interrompre proprement un programme lorsqu'une situation exceptionnelle se produit.
- Les exceptions sont des objets décrits par la classe **`Exception`**.
- Chaque exception porte un message qui décrit la raison qui a provoquée l'exception. La méthode **`String getMessage()`** permet de lire ce message.
- Exemples :
  - 2/0 message : "/ by zero" (avec *int* seulement !)
  - **`String s = null; s.length();`**
    message : "Null pointer exception"
  - **`int[] tab = new int[5]; tab[10];`**
    message : "Index Out of Bounds : 10>=5"

Effet de vague

- Lorsqu'une exception se produit, l'instruction courante est interrompue
- L'exécution de la méthode dans laquelle se trouvait l'instruction est également interrompue, elle ne renvoie aucune valeur
- L'exception se propage en cascade jusqu'à la méthode initialement invoquée.
- Finalement l'environnement (par exemple BlueJ) reçoit l'exception et affiche le message associé

Notes personnelles

Exceptions et assertions

- Quand l'assertion est violée, une exception est émise avec le message demandé par l'utilisateur
- L'instruction
  - `assert (valeur >= 0 && valeur < limite);`
- Pourrait se traduire par
  - `if (valeur < 0 || valeur >= limite) {`
    `… // émettre une exception`
  - `}`
- Les violations d'assertions sont plus graves que des exceptions, elles sont décrites par la classe `Error` au lieu de la classe `Exception`.

Exemple – la classe `Integer`

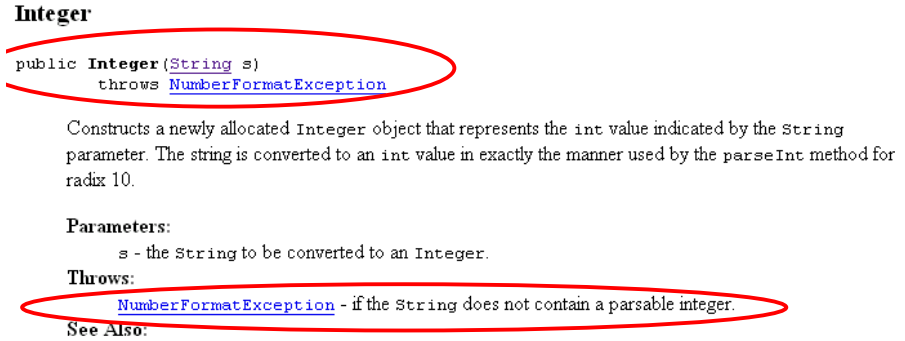
- La classe **`Integer`** permet de construire des objets qui représentent des entiers.
- C'est une classe enveloppante pour le type primitif **`int`**.
- Elle peut être utilisée avec **`ArrayList`** pour faire des listes d'entiers
  - Rappel : **`ArrayList`** contient des objets, pas de **`int`**.
- Elle a deux constructeurs :
  - **`Integer(int)`** : construit un **`Integer`** à partir d'un **`int`**
  - **`Integer(String)`** : construit un **`Integer`** à partir d'une chaîne de caractères (**`String`**).
    - `new Integer("125");  // Ok`
    - `new Integer("125s");  // ??`

- **`new Integer("125s");`** provoque une exception
  - *NumberFormatException for input String : 125s*
- La classe **`Integer`** a une méthode **`int intValue()`**
- On veut écrire la méthode `string2int()` qui convertit une chaîne de caractères en entier :

```
static int string2int(String s) {
   Integer i = new Integer(s);
   return i.intValue();
}
```


Comment savoir si une expression risque de lever une exception ?

- Pour les expressions : /0, [-1], null.length()
  - Il faut l'apprendre par cœur
- Pour les méthodes : **`new Integer("125s");`**
  - Il faut lire la documentation



Integer

`public Integer(String s)`
`throws NumberFormatException`

Constructs a newly allocated `Integer` object that represents the `int` value indicated by the `String` parameter. The string is converted to an `int` value in exactly the manner used by the `parseInt` method for radix 10.

Parameters:
`s` - the `String` to be converted to an `Integer`.
Throws:
`NumberFormatException` - if the `String` does not contain a parsable integer.
See Also:
`parseInt(java.lang.String, int)`

Capter les exceptions : `try/catch`

- Il est **possible** de capter les exceptions

```
try {
   // instructions sur lesquelles une exception risque d'être levée
   // …
} catch (Exception e) {
   // ce qu'on fait si l'exception est levée
}
// ce qu'on fait dans tous les cas après.
```

- De même on **peut** capter les `Error` :

```
try {
…
} catch (Error e) {
}
```


Les flots de caractères d'entrée et de sortie

Le paquetage **`java.io`**

`FileWriter, FileReader :` pour les caractères

`PrintWriter, BufferedReader :` pour les **`String`**

Le flot de sortie standard

- L'attribut statique **`out`** de la classe **`System`** est le **flot de sortie standard** (par défaut, il est dirigé vers la console)

```
System.out.println("Bonjour");
System.out.print("Bonjour");
```

- Deux méthodes permettent d'écrire des **chaînes de caractères**
  - **`void print(String)`**
  - **`void println(String)`**
- Ces deux méthodes sont surchargées pour tous les types primitifs et le type **`Object`**
  - **`void print(int), void print(double), void print(Object), …`**
  - avec un **`Object`**, la méthode **`toString()`** est invoquée pour le transformer en **`String`**.

Notes personnelles

Le flot standard d'erreurs

- **`System.out`** est utilisé pour afficher les messages d'informations
- **`System.err`** est le flot standard d'erreurs
  - Il doit être utilisé pour afficher des messages d'erreurs
  - Souvent les deux flots sont dirigés vers l'écran
  - Dans BlueJ, il sont dirigés vers deux fenêtres séparées !

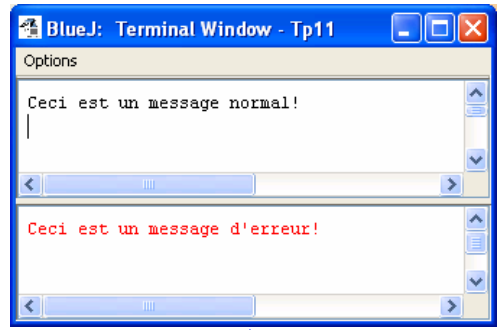
```
public class Standard {
  static public void main(String[] args) {
    System.out.println("Ceci est un message normal!");
    System.err.println("Ceci est un message d'erreur!");
  }
}
```


`System.err` dans BlueJ

```
public class Standard {
  static public void main(String[] args) {
    System.out.println("Ceci est un message normal!");
    System.err.println("Ceci est un message d'erreur!");
  }
}
```

- `>java Standard`
- Ceci est un message normal!
- Ceci est un message d'erreur!

Sans BlueJ
(sur la ligne de commande)



Avec BlueJ

`FileWriter` et `PrintWriter`

- On peut diriger un flot de sortie vers un fichier (on choisit le nom du fichier)
  - **`FileWriter fw = new FileWriter("fic.txt");`**
- **`FileWriter`** permet d'écrire un caractère à la fois

  `void write(int unicode);`
- Pour écrire un **`String`**, il faut écrire chaque caractère les uns après les autres
- Pour écrire un **`int`**, il faut aussi le décomposer en caractères !
- Pas très pratique ? On préfère généralement utiliser un **`PrintWriter`** :
  - **`PrintWriter pw = new PrintWriter(fw);`**
  - **`pw.print("message");`**

Obligatoire !

Écrire dans un fichier sur le disque ?

- Il faut choisir un nom pour le fichier : `fic.txt`
- Il faut **ouvrir un flot de sortie** vers le fichier :
  - **`FileWriter fw = new FileWriter("fic.txt");`**
  - **`java.io.FileWriter`** est un flot de caractères
- **Pour manipuler des lignes plutôt que des caractères :**
  - **`PrintWriter pw = new PrintWriter(fw);`**
  - On est obligé de passer par le **`FileWriter`** !
- Pour écrire ?
  - **`pw.println("Une ligne");`**
  - **`pw.println("Une autre ligne");`**
- Il faut **fermer le flot** quand on a fini
  - **`pw.close();`**
- Et si il y a une erreur lors de l'ouverture, la fermeture, l'écriture ?
  - Une **`Exception`** est levée, il faut l'attraper !

Un exemple – Un fichier texte

```
import java.io.FileWriter;
import java.io.PrintWriter;
public class Ecriture {
  static public void main(String[] args) {
    try {
      FileWriter fw = new FileWriter("fic.txt");
      PrintWriter pw = new PrintWriter(fw);
      for(int i=0; i<args.length; i+=1) {
        pw.println(i+" "+args[i]);
        System.out.println(args[i]);
      }
      pw.close();
    } catch (Exception ex) { // obligatoire !
       System.err.println("Erreur sur le fichier");
    }
  }
}
```

Ouvre un flot de sortie d'octets

Permet le println

Écrit dans le flot

Écrit sur la sortie standard

Ferme le flot

- **>java Ecriture je veux écrire ça!**
- Et le fichier **fic.txt** est créé ! Que contient-il ?

Et pour les entrées ? `System.in`

- **`System.in`** est le **flot d'entrée standard**

```
int read(byte[] b);
```

- La méthode **`read`** remplit le tableau **`b`** avec les octets lus et renvoie le nombre d'octets lu.
- Heureusement, il y a des classes enveloppantes pour traduire les octets en chaînes de caractères : **`BufferedReader`**.
- On peut aussi utiliser la classe **`java.util.Scanner`**.

```
try {
  Scanner sc = new Scanner(System.in);
  String lu = sc.nextLine();
  // ou int v = sc.nextInt(); // si on lit un entier
  // utiliser lu et/ou v …
} catch (Exception e) { // obligatoire !
  System.err.println("Erreur de lecture : " + e.getMessage());
}
```


Lecture d'un fichier texte

```
import java.io.*;
import java.util.Scanner;
public class Lecture {
  static public void main(String[] args) {
    try {
      File f = new File("fic.txt");
      Scanner sc = new Scanner(f);
      while(sc.hasNextLine()) { // toutes les lignes
        String lu = sc.nextLine();
        System.out.println("Lu:"+lu);  // affiche la ligne
      }
      sc.close();
    } catch (Exception ex) {
      System.err.println("Erreur de lecture : "+ex);
    }
  }
}
```

Ouvre un descripteur de fichier

Permet le readLine

Lit une ligne du flot

Ferme le flot

- **>java Lecture**

```
Lu:0 je
Lu:1 veux
Lu:2 écrire
Lu:3 ça!
```


Notes personnelles

Exemple – la classe `Etudiant`

```
import java.util.ArrayList;
public class Etudiant {
  private String nom;
  private ArrayList<Integer> notes;

  public Etudiant(String nom) {
    this.nom = nom;
    notes = new ArrayList<Integer>();
  }
  public void addNote(int note) {
// un ArrayList ne contient que des objets !
    notes.add(note);
  }
  public String toString() {
    String res = nom ;

    for(int i = 0; i < notes.size(); i = i + 1)
      res = res + " " + notes.get(i);

    return res;
  }
}
```


La classe `Promotion`

```
import java.util.ArrayList;
import java.io.*;

public class Promotion {
  private ArrayList<Etudiant> etudiants;
  public Promotion() { // une promotion sans étudiant
    etudiants = new ArrayList<Etudiant>();
  }
  public Promotion(String nomFich) {
    this();
    … // retrouve à partir d'un fichier
  }
  public void ajoute(Etudiant e) {
    etudiants.add(e);
  }
  public void sauve(String nomFich) {
    … // sauvegarde dans un fichier
  }
}
```


Sauvegarde dans un fichier

```
public void sauve(String nomFich) {
  try {
    FileWriter fw = new FileWriter(nomFich);
    PrintWriter pw = new PrintWriter(fw);
// Pour chaque étudiant
    for(int i = 0; i<etudiants.size(); i+=1) {
      pw.println(etudiants.get(i).toString());
    }
    pw.close();
  } catch (Exception ex) {
    System.err.println("Erreur d'écriture : "+ex);
  }
}
```


Lire à partir d'un fichier

```
public Promotion(String nomFich) {
  try {
    File fr = new File(nomFich);
    Scanner sc = new Scanner(fr);
// Tant qu'il y a des lignes à lire
    while (sc.hasNextLine()) {
      String ligne = sc.nextLine();
// Il faut traduire la ligne en étudiant!
      ajouteEtudiant(ligne);
    }
    sc.close();
  } catch (Exception ex) {
    System.err.println("Erreur de lecture : "+ex);
  }
}
```

Traduire la ligne en `Etudiant`

- Exemple : **`toto 12 15 8 4`**
- Ou sinon : **`titi 6 18`**

```
public void ajouteEtudiant(String ligne) {
   nom ← bout de ligne du début jusqu'au 1er ' '
   Soit e un nouvel Etudiant qui s'appelle nom

   TANT QUE il y a des notes (int) dans la ligne FAIRE
      note ← Extraire la note suivante
      Ajouter note à l'Etudiant e
   FIN TANT QUE
   Ajouter l'Etudiant e à la liste d'étudiants
}
```



```
private void ajouteEtudiant(String ligne) {
   Scanner sc = new Scanner(ligne);
   String nom = sc.next();
   Etudiant e = new Etudiant(nom);

   while (sc.hasNextInt()) {
      int note = sc.nextInt();
      e.addNote(note);
   }
   etudiants.add(e);
}
```


Une méthode `main` pour lancer

```
/** lit une promotion, ajoute un étudiant toto et sauvegarde la
   modification.
   Les noms des fichiers source et destination sont lus en
   paramètre de la ligne de commande */
static public void main(String[] args) {
   assert(args.length == 2);
   Promotion p = new Promotion(args[0]);
   Etudiant e = new Etudiant("toto");
   p.ajoute(e);
   e.addNote(10); e.addNote(15);
   p.sauve(args[1]);
}
```


Notes personnelles