Le Langage Java

1re année

J. Beleho (bej) C. Leruste (clr) M. Codutti (mcd)
P. Bettens (pbt) F. Servais (srv) C. Leignel (clg)
D.P. Bishop (bis) S. Drobisz (sdr)

Haute École de Bruxelles-Brabant — École Supérieure d'Informatique

Année académique 2016 / 2017

Séance 6

Les alternatives et la gestion des erreurs

- Alternatives (survol)
- Écrire du code robuste
- Gérer les erreurs
- Confiner les problèmes

Alternatives

Crédit photo

Instructions de choix

Le Si

```
if ( condition ) {
    instructions
}
```

Le Si-sinon

```
if ( condition ) {
    instructions
} else {
    instructions
}
```

Exemple

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
      Scanner clavier = new Scanner(System.in);
      int nombre1;

      nombre1 = clavier.nextInt();
      System.out.println(négatif(nombre1));
  }

  public static String négatif(int nombre1){
      String chaine = "";
      if (nombre1 < 0) {
         chaine = nombre1 + " est négatif";
      }
      return chaine;
  }
}
```

Exemple

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
      Scanner clavier = new Scanner(System.in);
      int nombre1;

      nombre1 = clavier.nextInt();
      System.out.println(positifOuNégatif(nombre1));
  }

  public static String positifOuNégatif(int nombre){
      String chaine = nombre + " est un nombre ";
      if (nombre < 0) {
          chaine = chaine + "négatif";
      } else {
          chaine = chaine + "positif";
      }
      return chaine;
  }
```

Exercice

Comment traduire cet algorithme ?

```
Algorithme test (nombre1 : entier) -> chaine
    chaineAfficher : chaine
    chaineAfficher <- nombre1 + "est "
    Si nombre1 > 0 Alors
        chaineAfficher <- chaineAfficher + "positif"
    Sinon
        Si nombre1 = 0 Alors
            chaineAfficher <- chaineAfficher + "nul"
        Sinon
            chaineAfficher <- chaineAfficher + "négatif"
        Fin Si
    Fin Si
    Retourner chaineAfficher
Fin Algorithme
```

Le « si-sinon-si »

L'exemple précédant s'écrirait mieux (extrait) :

```
Si nombre1 > 0 Alors
    chaineAfficher <- chaineAfficher + "positif"
Sinon Si nombre1 = 0 Alors
    chaineAfficher <- chaineAfficher + "nul"
Sinon
    chaineAfficher <- chaineAfficher + "négatif"
Fin Si
```

Le « si-sinon-si »

Ce qui donnerait en Java

```
if (nb>0) {
    chaineAfficher = chaineAfficher + "positif";
} else if (nb==0) {
    chaineAfficher = chaineAfficher + "nul";
} else {
    chaineAfficher = chaineAfficher + "négatif";
}
```

Expressions booléennes

Les comparateurs

`<` `>` `<=` `>=` `==` `!=`

Les opérateurs booléens

`&&` (et) `||` (ou) `!` (non)

Exemple

```
import java. util .Scanner;
public class Exemple {
  public static void afficherParité (int nombre1){
      if ((nombre1 % 2) == 0) {
         System.out. println ("Le␣nombre␣est␣pair");
      } else {
         System.out. println ("Le␣nombre␣est␣impair");
      }
  }
}
```

Exemple

```
import java.util.Scanner;
public class Exemple {
  public static void afficherTarifRéduit (int âge){
      if ( âge<21 || âge>=60 ) {
         System.out.println ("Tarif␣réduit␣!");
      }
  }
}
```

Le « selon-que »

```
switch(produit) {
   case "Coca" :
   case "Sprite" :
   case "Fanta" :
         prixDistributeur =60;
        break;
   case "IceTea" :
         prixDistributeur =70;
        break;
   default :
         prixDistributeur =0;
        break;
}
```

- Notez le **break**
- Possible avec : entiers, caractères et chaines


Environnement défaillant
Crédit photo

Motivation

Un programme ne tourne pas dans un monde idéal

Il doit pouvoir **résister aux défaillances** de l'environnement

- On tente d'ouvrir un fichier qui n'existe pas
- L'utilisateur entre des données incorrectes
- ...

La gestion des erreurs

Exemple :

```
import java.util.Scanner;
public class Affiche {
  /**
   * Affiche un nombre entier lu au clavier.
   * @param args non utilisé
   */
  public static void main(String[] args) {
      Scanner clavier = new Scanner(System.in);
      int nb;

      nb = clavier.nextInt();
      System.out.println(nb);
  }
}
```

La gestion des erreurs

Et si l'utilisateur entre une lettre ?

```
> javac Affiche.java
> java Affiche
a
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:857)
        at java.util.Scanner.next(Scanner.java:1478)
        at java.util.Scanner.nextInt(Scanner.java:2108)
        at java.util.Scanner.nextInt(Scanner.java:2067)
        at Affiche.main(Affiche.java:7)
```

Nom de l'exception

Pile d'appels (par où il est passé)

Comment (essayer) de gérer le problème ?


catch

La gestion des erreurs

L'instruction **try catch**

- **try** : contient les instructions qui **peuvent mal se passer**
- **catch** : contient le code qui est **en charge** de gérer le problème

Quand un problème se présente dans le **try**

- Le code du **try** est interrompu
- Le code du **catch** est exécuté
- Ensuite, on continue après le **try–catch**

La gestion des erreurs

Exemple : on affiche un message plus clair

```
package be.he2b.esi.lgj1;
import java.util.Scanner;
public class Affiche {
  /**
   * Affiche l'entier lu au clavier ou un message si ce n'est pas un entier.
   * @param args inutilisé.
   */
  public static void main(String[] args) {
      Scanner clavier = new Scanner(System.in);
      int nb;
      try {
        nb = clavier.nextInt();
        System.out.println(nb);
      }
      catch(Exception e) {
        System.out.println("Ce n'est pas un entier!");
      }
  }
}
```

Confiner les problèmes - Motivation

Imaginons la situation suivante :

- Un programme demande un entier à l'utilisateur
- Il doit être positif
- L'utilisateur entre un nombre négatif
- Le programme ne le vérifie pas tout de suite

Confiner les problèmes - Motivation

On aura un problème :

- Un plantage
- Un résultat erroné
- Un effet indésiré (perte de données, ...)

Mais le problème va survenir :

- Plus tard dans le temps
- Plus loin dans le code

$\Longrightarrow$ **Difficile** à comprendre et **corriger**

Confiner les problèmes

Besoin de **confiner** les problèmes

- Un problème est **détecté rapidement** avant qu'il ne se propage dans le reste du code

Cas pratique : vérifier les **paramètres**

- Si une contrainte est associée à un paramètre
  - Le vérifier en début de méthode
  - Que faire si pas valide ?

Lancer une exception

Confiner les problèmes

Lancer une exception

```
/**
 * Calcule la racine carrée d'un nombre.
 * @param nb le nombre dont on veut la racine carée.
 * @return la racine carrée de <code>nb<\code>.
 * @throws IllegalArgumentException si <code>nb</code> est négatif.
 */
public static double racineCarrée(double nb) {
  if (nb<0) {
    throw new IllegalArgumentException("nb doit être positif !");
  }
  // Traitement normal. On est sûr que le paramètre est OK.
}
```

Confiner les problèmes

Exemple

```
try {
  System.out.println( racineCarrée( val ) );
} catch (Exception ex) {
  System.out.println( "Calcul␣impossible␣!" );
}
```

On peut aussi préciser qu'on n'attrape **que** les IllegalArgumentException

```
try {
  System.out.println( racineCarrée( val ) );
} catch (IllegalArgumentException ex) {
  System.out.println( "Calcul␣impossible␣!" );
}
```

Crédits

Ces slides sont le support pour la présentation orale de l'unité d'enseignement **DEV1-JAV** à HE2B-ÉSI

Crédits

Les distributions Ubuntu et/ou debian
du système d'exploitation GNU Linux.
LaTeX/Beamer comme système d'édition.
Git et GitHub pour la gestion des versions et le suivi.
GNU make, rubber, pdfnup, ... pour les petites tâches.

Images et icônes

deviantart, flickr, The Noun Project