ULB UNIVERSITÉ LIBRE DE BRUXELLES

CoDe

INFO-H-301
Programmation orientée objet

TP2

- Un peu plus sur les objets
- Introduction aux exceptions
- Test unitaires

Les packages

Package Java

- **Un *package* Java est un ensemble de types** (classes, interfaces, énumérations, …).

- Par convention, le nom d'un package commence par une minuscule et est souvent un nom de domaine inversé (`be.ac.ulb.code`) pour éviter les conflits.
- Par exemple, les classes fondamentales se trouvent dans `java.lang` (importé par défaut), les classes d'entrée/sortie se trouvent dans `java.io`, …

- A la **première ligne** d'un fichier, il faut préciser le package concerné (sinon il sera dans le package par défaut) :
  - `package nomDuPackage;`

Package Java

- Pour utiliser des types d'un package dans un fichier d'un autre package :
  - soit on importe tous les types d'un package au début du fichier :
    `import tp1.*;` (comparable à `using namespace` en C++)
  - soit on importe un type contenu dans un package au début du fichier :
    `import tp1.Date;`
  - soit on utilise le nom complet dans le code :
    `tp1.Date date = new tp1.Date(12,11,09);`

Encapsulation

- **L'encapsulation** est un **concept OO** qui consiste à **protéger** l'information contenue dans un objet.

- En Java, les attributs et méthodes peuvent être :
  - `public` : accessibles de partout (intérieur, extérieur)
  - `private` : accessibles seulement à l'intérieur de la classe
  - ***private-package* (par défaut si on ne précise rien) : accessibles à l'intérieur de la classe et à l'intérieur d'un *package*.**
  - `protected` : *voir plus loin*

- ***Une classe doit être le vigile de l'intégrité de ses objets.*** Pour ce faire, on va généralement déclarer tous ses attributs en `private` et gérer leur intégrité via des **accesseurs**. Attention, il ne faut faire des accesseurs que si c'est nécessaire.

Niveaux de protection d'une classe

- Les niveaux de protection pour une classe sont :
  - `public` : la classe peut être vue par tout le monde
  - *private-package* (si rien n'est précisé) : la classe n'est vue qu'à l'intérieur de son *package*
- Il ne peut y avoir au maximum qu'une classe publique par fichier `.java`.

Le mot clé *static*

Mot clé `static`

- Un attribut ou une méthode statique **se réfère à une classe** et non à une instance particulière.

- Exemples :
  - `Math.PI` : pas besoin d'instancier un objet de type Math
  - `Math.max(int i, int j)`
  - `static void main(…)` : méthode dans laquelle on va créer les premiers objets.

- Une méthode statique dans une classe ne peut accéder qu'aux membres statiques de cette classe.

Méthode statique : exemple

```
public Date{
  …
  private static int[] daysInMonth =
     new int[]{31,28,31,30,31,30,31,31,30,31,30,31}

  private static int daysInMonth(int month, int year) {
     return daysInMonths[month-1]
          + (isLeapYear(year) && month==2? 1 : 0 );
  }
}

…

int daysInFeb12 = Date.daysInMonth(2,2012);
```

défini statiquement dans Date

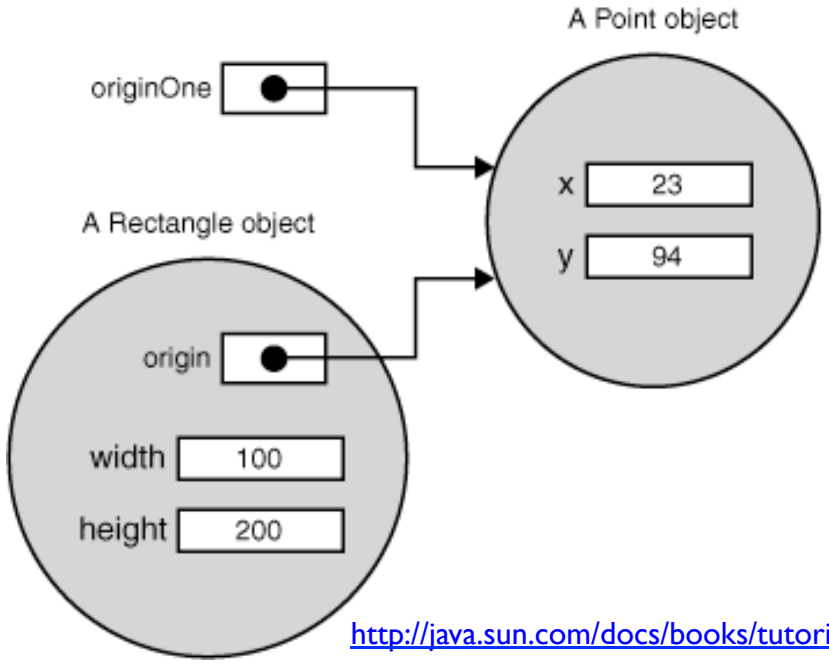
Appel sur la classe et non sur une instance

Les références

Objets et références

- Les objets sont toujours utilisés par références.
- Une instance = un `new`.

```
Point originOne = new Point(23, 94);
Rectangle rectOne = new Rectangle(originOne, 100, 200);
```



http://java.sun.com/docs/books/tutorial/java/javaOO/objectcreation.html

Les exceptions

Les exceptions en Java

- En Java, quand quelque chose se passe mal, une exception peut être déclenchée :
  - Une condition d'exécution anormale a été détectée par la machine virtuelle (division par zéro, dépassement de tableau, …)
  - Une exception a été lancée par une méthode via l'instruction `throw`
  - Une exception asynchrone a été déclenchée (voir plus loin)

- Une exception est un objet d'une classe ou d'une sous classe de la classe `Throwable`. Par exemple un objet de type `Exception`.

- http://java.sun.com/docs/books/jls/third_edition/html/exceptions.html

Attraper une exception

- On peut attraper (*catch*) une exception en Java à l'aide de l'instruction `try catch`
- Exemple :

```
try {
    // code pouvant déclencher une exception
    // si un exception se déclenche,
    // on saute dans le bloc catch
} catch (Exception e) {
    // gestion de l'exception
    // par exemple :
    System.out.println(e.getMessage());
}
```

Attraper une exception

- L'instruction `try catch finally` permet d'exécuter du code qu'une exception ait été déclenchée ou non.
- Exemple :

```
try {
    // code pouvant déclencher une exception
    // si un exception se déclenche,
    // on saute dans le bloc catch
} catch (Exception e) {
    // gestion de l'exception
} finally {
    // code exécuté de toute façon
}
```

Checked / Unchecked Exceptions

- Exception ***unchecked*** :
  - **Error** (plutôt hardware):
    - Erreurs qui peuvent arriver à beaucoup d'endroits. Diffile ou impossible à réparer. (division par zero, sortir d'un tableau, …)
  - **RuntimeException** (plutôt software)
    - Exceptions qui ne devraient pas arriver comme un pointeur nul mais qui pourraient arriver tellement souvent que les gérer serait fastidieux.
- Exceptions ***checked*** :
  - **Toutes les autres** exceptions.

Declencher une exception

- L'instruction `throw` permet de **déclencher** une exception et sortir de la méthode courante.
- Les méthodes qui peuvent déclencher une exception ***checked*** doivent le **déclarer** dans leur signature via le mot clé `throws`. Et donc le code qui appelle ces méthodes doivent attraper cette exception *checked*.
- Exemple :

```
public void aMethod() throws Exception {
    …
    if(somethingHappens()) {
        throw new Exception("Boumshanka ! An interception !");
    }
    …
}
```

Les tests unitaires