TP de JAVA - Série 8
Design de classes

Université de Mons-Hainaut
2004 - 2005

1 Résumé du chapitre

- Une classe ne devrait représenter qu'un unique concept du domaine auquel appartient le problème, tel que la finance, la science ou les mathématiques.
- L'interface publique d'une classe est cohérente si tous ses éléments sont liés au concept que cette classe représente.
- Une classe dépend d'une autre classe qui elle utilise des objets de cette autre classe.
- Il est de bonne pratique de minimiser le couplage (c'est-à-dire les dépendances) entre les classes.
- Un "accessor" ne modifie pas l'état du paramètre implicite. Un "mutator" peut change cet état.
- Une classe immuable n'a pas de "mutator".
- L'effet de bords d'une méthode est un comportement observable de l'extérieur, hormis le paramètre implicite.
- En Java, une méthode ne peut jamais modifier les paramètres de type primitif.
- En Java, une méthode ne peut modifier l'état de paramètres référence objet, mais elle ne peut pas remplacer cette référence objet par une autre.
- Une pré-condition est un pré-requis que l'appelant d'une méthode doit respecter. Si une méthode est appelée en violation d'une pré-condition, la méthode n'a plus la responsabilité de fournir le résultat correct.
- Si une méthode est appelée en respectant ses pré-conditions, alors elle doit assurer que les post-conditions sont valides.
- Une méthode `static` n'a pas de paramètre implicite.
- Un champ (attribut) `static` fait partie de la classe, pas de chacun des objets de cette classe (il est partagé).
- La portée d'une variable est la région du programme dans laquelle vous pouvez utiliser cette variable par son nom.
- En Java, vous ne pouvez pas avoir deux variables locales ayant des portées qui se chevauchent.
- Un nom qualifié est préfixé par le nom de sa classe ou par une référence objet, tels que Math.sqrt ou other.balance.

- Un champ d'instance ou un nom de méthode non qualifiée se référe au paramètre implicite `this`.
- Une variable locale peut "cacher" un champ portant le même nom. On peut accèder au nom du champ "caché" en le qualifiant avec la référence `this`.
- Un paquetage est un ensemble de classes reliées.
- La directive `import` vous permet d'utiliser une classe de ce paquetage sans le prefix du paquetage.
- Utilisez un nom de domaine en plus quand vous souhaitez construire des noms de paquetages non ambigus.
- Le chemin d'une fichier `class` doit coincider avec son nom de paquetage.

Les packages, classes et méthodes introduites dans ce chapitre:

```
java.lang.IllegalArgumentException
```

2 Exs. de révision

R8.1 : Condidérer la description suivante de programme:
Des clients insèrent des pièces dans un distributeur et sélectionnent un produit en poussant un bouton. Si les pièces insérées sont suffisantes pour couvrir le prix d'achat du produit, le produit est délivré et la monnaie donnée. Dans le cas contraire, les pièces insérées sont rendu au client.

Quelles classes utiliseriez-vous pour l'implémenter?

R8.2 : Condidérer la description suivante de programme:
Des employés sont payés toutes les deux semaines. Ils recoivent leur salaire horaire pour chaque heure travaillée. Cependant, s'ils ont travaillé plus de 40 heures par semaine, les heures au-delà sont payées au taux de 150% leur salaire normal.

Quelles classes utiliseriez-vous pour l'implémenter?

R8.3 : Considérer la description suivante de programme:
Des consommateurs commandent des produits dans un magasin. Des factures sont établies pour effectuer la liste des produits, les quantités commandées, les paiements reçus et les sommes encore dues. Les produits sont livrés à l'adresse de livraison du client, et les factures sont envoyées à l'adresse de facturation.

Quelles classes utiliseriez-vous pour l'implémenter?

R8.4 : Regarder l'interface publique de la classe `System` et discuter de savoir si elle est cohérente ou non.

R8.5 : Supposons qu'un objet `Facture` contienne les descriptions des produits commandés, ainsi que les adresses de livraison et de facturation du client. Dessiner les graphes de dépendances entre les classes `Facture`, `Adresse`, `Client` et `Produit`.

R8.6 : Supposons qu'un distributeur contienne des produits et qu'un client insère des pièces dans le distributeur pour acheter des produits. Dessiner les graphes de dépendances entre les classes `Distributeur`, `Pièce` et `Produit`.

R8.7 : De quelles classes dépend la classe `Integer` dans la librairie standard.

R8.8 : De quelles classes dépend la classe `Rectangle` dans la librairie standard.

R8.9 : Classer les méthodes de la classe `StringTokenizer` en accessors et mutators.

R8.10 : Classer les méthodes de la classe `Rectangle` en accessors et mutators.

R8.11 : Lesquelles de ces classes sont immuables?

- `Rectangle`
- `String`
- `Random`

R8.12 : Lesquelles de ces classes sont immuables?

- `PrintStream`
- `Date`
- `Integer`

R8.13 : Si elles en ont décrire les effets de bords des 3 méthodes suivantes.

```
public class Coin
{
  public void print()
  {
      System.out.println(name+" "+value);
  }
  public void print(PrintStream stream)
  {
      stream.println(name+" "+value);
  }
  public  String toString()
  {
      return name+" "+value;
  }
  ...
}
```

R8.14 : Idéalement, une méthode ne devrait pas avoir d'effet de bord. Pouvez-vous écrire un programme dans lequel aucune méthode n'a d'effet de bord? Un tel programme serait-il utile?

R8.15 : Ecrire les pré-conditions des méthodes suivantes. Ne pas implémenter les méthodes.

- `public static double sqrt(double x)`
- `public static String romanNumeral(int n)`
- `public static double slope(Line2D.Double a)`
- `public static String weekDay(int day)`

R8.16 : Quelles pré-conditions possèdent les méthodes suivantes dans la librairie standard.

- `Math.sqrt`
- `Math.tan`
- `Math.log`

- `Math.exp`
- `Math.pow`
- `Math.abs`

R8.17 : Quelles pré-conditions possèdent les méthodes suivantes dans la librairie standard.

- `Integer.parseInt(String s)`
- `StringTokenizer.nextToken()`
- `Random.nextInt(int n)`
- `String.substring(int m, int n)`

R8.18 : Quand une méthode est appelée avec des paramètres qui ne respectent pas sa précondition, elle peut lancer une exception, ou retourner à son appelant. Donner deux exemples de méthodes de librairie (librairie standard ou librairie utilisée dans le cours et les travaux pratiques) qui retournent des résultats à l'appelant dans le cas d'un appel avec paramètres invalides, et deux exemples lancant une exception.

R8.19 : Considérer une classe `Purse` avec les méthodes suivantes:

- `public void addCoin(Coin aCoin)`
- `public double getTotal()`

Donner une postcondition raisonnable pour la méthode `addCoin`. De quelle précondition avez-vous besoin pour que la classe `Purse` puisse assurer cette postcondition?

R8.20 : Considérer la méthode suivante dont le but est d'échanger les valeurs de deux nombres à virgule flottante:

```
public static void falseSwap(double a,double b)
{
   double temp=a;
   a=b;
   b=temp;
}
public static void main(String args[])
{
   double x=3;
   double y=4;
   falseSwap(x, y);
   System.out.println(x+" "+y);
}
```

Pourquoi la méthode `falseSwap` n'échange pas les contenus de x et y?

R8.21 : Comment pourriez-vous écrire une méthode qui échange deux nombres à virgule flottantes?
Indication: `Point2D.Double`

R8.22 : Essayer de compiler le programme suivant:

```
public class Ex7_22
{
  public void print(int x)
  {
      System.out.println(x);
  }
  public static void main(String arg[])
  {
      int n=13;
      print(n);
  }
}
```

R8.23 : Regarder les méthodes de la classe `Integer`. Lesquelles sont static? Pourquoi?

R8.24 : Regarder les méthodes de la classe `String` (ignorer celles qui prennent des paramètres de type `char[]`). Lesquelles sont static? Pourquoi?

R8.25 : Dans la classe suivante, la variable `n` apparait avec plusieurs portées. Quelles déclarations sont légales et lesquelles ne le sont pas?

```
public class x
{
   public int f()
   {
      int n=1;
      return n;
   }
   public int g(int k)
   {
      int a;
      for(int n=1; n<=k;n++)
         a=a+n;
      return a;
   }

   public int h(int n)
   {
      int b;
      for(int n=1; n<=10;n++)
         b=b+n;
      return b+n;
   }
   public int k(nit n)
   {
      if(n<0)
      {
         int k=-n;
         int n=(int)(Math.sqrt(n));
         return n;
      }
      else return n;
```

```
    }

    public int m(int k)
    {
        int a;
        for(int n=1; n<=k;n++)
            a=a+n;
        for(int n=k; n>=1;n++)
            a=a+n;
        return a;
    }
    private int n;
}
```

R8.26 : Qu'est-ce qu'un nom qualifié? Qu'est-ce qu'un nom non qualifié?

R8.27 : Quand vous accèdez à un nom qualifié dans une méthode, qu'est-ce que cet accès signifie? Discuter entre instance et éléments static.

R8.28 : Qu'est-ce que le paquetage par défaut? L'avez-vous utilisé dans vos programmes avant ce chapitre?

3 Exs. programmation

P8.1 : Implémenter les classes `Purse` et `Coin` décrites dans la section 7.2 du livre.

P8.2 : Modifier la classe `BankAccount` en ajoutant des pré-conditions pour le constructeur et la méthode `deposit` qui nécessite un paramètre `amount` qui doit être supérieur à 0, et une pré-condition pour la méthode `withdraw` qui nécessite un `amount` inférieur à la valeur courrante de `balance`. Lancer des exceptions si la pré-condition n'est pas remplie.

P8.3 : Ecrire les méthodes static:

- public static double sphereVolume(double r)
- public static double sphereSurface(double r)
- public static double cylindreVolume(double r, double h)
- public static double cylindreSurface(double r, double h)
- public static double coneVolume(double r, double h)
- public static double coneSurface(double r, double h)

qui calcule le volume et la surface d'une sphère de rayon `r`, d'un cylindre de base de rayon `r` et de hauteur `h`, et d'un cône de base circulaire d erayon `r` et de hauteur `h`. Placer ces méthodes dans une classe appropriée. Ecrire un programme qui demande à l'utilisateur d'entrer les valeurs de `r` et de `h`, qui appelle ces 6 méthodes et affiche les résultats.

P8.4 : Résoudre l'exercice P8.3 en implémentant des classes `Sphere`, `Cylindre` et `Cone`. Laquelle des deux approches est le plus orienté objet.

P8.5 : Ecrire une méthode
public static double distance(Point2D.Double p, Point2D.Double q)

qui calcule la distance entre deux points. Ajouter la méthode à une classe appropriée. Écrire un programme de test qui demande à l'utilisateur d'entrer les deux points. Afficher la distance. Pourquoi cela à un sens d'utiliser une méthode static dans ce cas.

P8.6 : Ecrire une méthode
public static boolean isInside(Point2D.Double p, Ellipse2D.Double e)
qui teste qui un point est à l'intérieur d'une ellipse. Ajouter la méthode à une classe appropriée. Écrire un programme de test qui demande à l'utilisateur d'entrer un point et une ellipse et qui affiche si le point est à l'intérieur de l'ellipse ou non..

P8.7 : Ecrire une méthode
public static int readInt(String prompt, int min, int max)
qui affiche le paramètre `prompt` comme message d'invitation, lit un entier et teste si celui-ci est compris entre le minimum et le maximum. Si ce n'est pas le cas, afficher un message d'erreur et relire l'entrée. Si l'utilisateur annule la fenêtre de saisie, afficher un message d'erreur et continuer à lire. Ajouter la méthode à une classe appropriée et fournir un programme de test.

P8.8 : Codes barre postaux
Pour trier les lettres plus vite, les services postaux encouragent les sociétés envoyant de grande quantité de courrier, à utiliser un code barre représentant le ZIP code (voir Figure 1). Le schéma de codage pour un ZIP code à 5 chiffres est présenté à la Figure 2. Il y a des barres complètes de chaque côté. Les 5 chiffres codés sont suivis par un chiffre de correction, qui est calculé comme suit. Additionner tous les chiffres et placer le chiffre de correction de manière à ce que cette somme atteigne un multiple de 10.
Exemple: pour le ZIP code 95014, la somme des chiffres est 19, le chiffre de correction est alors 1, pour atteindre 20.

Figure 1

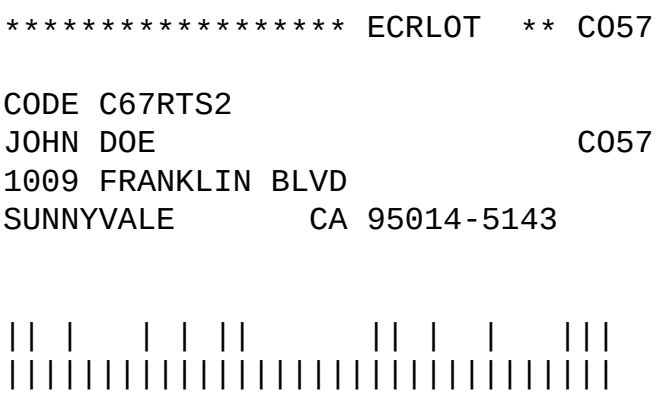
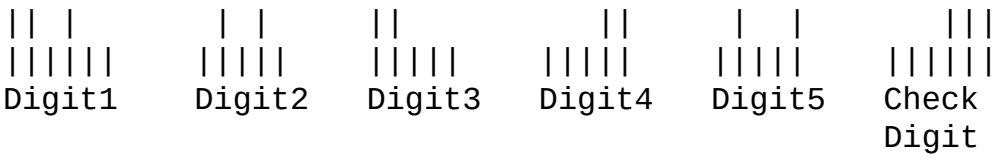


Figure 2



Chaque chiffre d'un ZIP code, et le chiffre de correction sont codés selon le modèle suivant:
Ou 0 représente une demi-barre et 1 une barre complète. Noter que ceci représente toutes les combinaisons de 2 barres complètes et 3 demis barres. Le chiffre peut facilement être calculé depuis le code barre en utilisant les poids de colonnes 7, 4, 2, 1, 0.

Par exemple, 01100 est 0*7 + 1*4 + 1*2 + 0*1 + 0*0 = 6. La seule exception est 0 qui devrait donner 11 selon la formule.

	7	4	2	1	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	1	0	0	0	1
8	1	0	0	1	0
9	1	0	1	0	0
0	1	1	0	0	0

Ecrire un programme qui demande à l'utilisateur un ZIP code et affiche le code barre. Utiliser ':' pour les demi-barre et '|' pour les barres pleines.

Par exemple, 95014 devient ||:|:::|:|:||::::::||:|::|:::|||

Utiliser des classes `BarCode` et `Digit` dans votre solution.

P8.9 : Ecrire un programme qui lit un code barre (avec des ':' pour les demi barres et des '|' pour les barres complètes), et afficher le ZIP code représenté. Afficher un message d'erreur si le code barre n'est pas complet.

P8.10 : Considérer l'algorithme suivant pour calculer x^n pour un entier : `n`.
Si $n < 0$, x^n est $1/x^{-n}$.
Si n est positif et pair, $x^n = (x^{n/2})^2$.
Si n est positif et impair, $x^n = x^{n-1}.x$.
Implémenter une méthode static `intPower(double x, int n)` qui utilise cet algorithme. Ajouter cette méthode à la classe `Numeric`.

P8.11 : Considérer la classe `Die` du Chapitre 6 (page 263). Tranformer le champ `generator` en champ static de façon à ce que toutes les instances partagent un générateur de nombre aléatoire.

P8.12 : Cet exercice suppose que vous ayez une adresse e-mail. Ecrire des classes `Greeter` et `GreeterTest` dans un paquetage dont le nom est dérivé de votre adresse e-mail, tel que décrit Section 7.9 du livre (page 310).

P8.13 : Implémenter les classes `Purse` et `Coin` décrites dans la section 7.2 du livre (page 283). Placer ces classes dans un paquetage nommé `Money`. Ecrire une classe `MoneyTest` dans le paquetage par défaut.