IFT 1015 - Fonctions 1

Professeur:

Stefan Monnier

B. Kégl, S. Roy, F. Duranleau, S. Monnier

Département d'informatique et de recherche opérationnelle

Université de Montréal

hiver 2006

Au programme

[Tasso:5]

- Algorithme paramétré
- Utilisation des méthodes
- Définition des méthodes
- Conception des méthodes

Les méthodes de style “recette”

Muffin aux raisins

- ingrédients: farine, lait, oeuf, beurre, sucre, raisins
- recette:

```
1. Mélanger la farine et le sucre.
2. Incorporez le lait, les oeufs et le beurre
3. Ajoutez les raisins.
4. Verser dans des moules et cuire.
```

Muffin au dattes/bleuets/chocolat?

Algorithme paramétré

Le muffin paramétré

Muffin à x

- ingrédients: farine, lait, oeuf, beurre, sucre, x
- recette(x):

```
1. Mélanger la farine et le sucre.
2. Incorporez le lait, les oeufs et le beurre
3. Ajoutez x.
4. Verser dans des moules et cuire.
```

Le raisin est l'ingrédient qui varie $=$ le paramètre

On réutilise le code

Choix des paramètres

- Extension simple: muffin au maïs, muffin au son, etc.
- Certains ingrédients et commandes de base ne changent pas
- Les paramètres sont seulement les ingrédients que l'on a l'intention de changer

Méthodes

Terminologie:

- algorithme paramétré = méthode ou fonction

Attributs:

- nom (muffin, gâteau, soupe, . . .)
- paramètres (fruit, . . .)
- corps: séquence d'instructions qui implémente l'algorithme
- valeurs de retour

Le résultat dépend des valeurs des paramètres

Méthodes

Vue “boîte noire”:

- nom = nom de la boîte
- entrée(s) = paramètre(s)
- sortie(s) = valeur(s) de retour
- implémentation = séquence d’instructions

Elles sont parmi nous...

Grille-pain

- Paramètres
  - tranche de pain
  - durée de chauffage
- Résultat
  - un toast

Laveuse à linge, sécheuse, lave-vaisselle, four micro-onde, etc...

Pourquoi les méthodes?

- Éviter la répétition = réutilisation du code
- Premier niveau d'abstraction et d'encapsulation
- Programme plus simple à comprendre
- Faciliter la détection/correction d'erreur
- Facilite le travail d'équipe

Abstraction/encapsulation

Les détails de l'implantation ne sont pas nécessaires pour l'utilisation

- nécessaires: le nom et les types de paramètres = la signature
- l'implantation est cachée

Méthodes prédéfinies

```
Math.random();
Integer.parseInt(<String>);
System.out.println(<String>);
Math.pow(<double>, <double>);
```

`http://java.sun.com/j2se/1.4.2/docs/api/`

Utilisation des méthodes

Syntaxe:

- `nomDeClasse.nomDeMethode(<liste des paramètres actuels>)`
- l'ordre des paramètres est important:
  `Math.pow(<double>,<double>)`)

Paramètre actuel:

- une expression dont le type est convertible au type du paramètre formel

Utilisation des méthodes

Paramètres formels se trouvent

- dans la définition de la méthode (plus tard . . .)
- dans la documentation de la méthode

Une méthode peut retourner une valeur

- le type de la valeur retournée se trouve dans la définition/documentation
- type `void` = pas de sortie/valeur retournée
- la valeur retournée est utilisée lorsque l'appel de méthode apparait dans une expression

Utilisation des méthodes

Terminologie

- la méthode/fonction est invoquée ou appelée

Exemples

```
System.out.println("Hello");
System.out.println(100);
int nombreADeviner = (int)(Math.random() * 100) + 1;
double n = floating * Math.pow(10,power);
double diagonale = Math.sqrt(c1 * c1 + c2 * c2);
double entree = Keyboard.readInt();
int max = Math.max(2 * a,b - 3);
```

Exemples & Exercices

`<double> f(<double>)`: $f(x) = -x^2 + 3x - 2$

- calculer $f(0)$
- afficher $f(x)$ pour les x entiers sur l'intervalle $-5 < x \leq 5$
- trouver la valeur de $f(f(...f(0.4)...))$ avec 10 f
- trouver le maximum de $f(x)$ sur les entiers $0 < x \leq 10$.

Extraire une méthode: avant

```
public static void main(String [] args)
{
  // rayon: le rayon lu; perim: le périmètre du cercle.
  double rayon, perim;

  // Lecture du rayon.
  rayon = Double.parseDouble(args[0]);

  // Calcul du périmètre.
  perim = 2 * Math.PI * rayon;

  // Afficher le résultat.
  System.out.println("Le cercle de rayon " + rayon
    + " a le périmètre " + perim);
}
```

Extraire une méthode: après

```
public static void main(String [] args)
{
  // rayon: le rayon lu; perim: le périmètre du cercle.
  double rayon, perim;

  // Lecture du rayon.
  rayon = Double.parseDouble(args[0]);

  // Calcul du périmètre.
  perim = perimetre(rayon);

  // Afficher le résultat.
  System.out.println("Le cercle de rayon " + rayon
    + " a le périmètre " + perim);
}
```

Extraire une méthode: pendant

```
public class Cercle
{
  public static void main(String [] args)
  {
    ...
  }

  public static double perimetre(double rayon)
  {
    return 2 * Math.PI * rayon;
  }
}
```

Définition des méthodes

Syntaxe

```
public static <typeDeRetour>
    nomDeMethode(<liste des paramètres formels>)
{
   <séquence d'instructions>
}
```

Paramètre formel

- syntaxe: `<type> nomDeParamètre`
- comme une définition de variable
- initialisé par le paramètre actuel quand la méthode est invoquée

Définition des méthodes (2)

L'instruction `return`

- syntaxe: `return expressionARetourner;`
- le type de `expressionARetourner` doit égaler (ou être convertible à) le `typeDeRetour` de la méthode
- le type `void`: pas de sortie (e.g.: `System.out.println()`)

Où placer la définition

- dans le corps d'une classe
- l'ordre est arbitraire
- séparément les unes des autres

Définition des méthodes

Où placer la définition

```
public class Cercle
{
  public static void main(String[] args)
  { ... }

  public static double perimetre(double rayon)
  { ... }

  public static double aire(double rayon)
  { ... }
}
```

Invocation des méthodes

Comment invoquer

- en général:
  `nomDeClasse.nomDeMethode(<liste des paramètres actuels>)`
- dans la même classe:
  `nomDeMethode(<liste des paramètres actuels>)`

La méthode `main()`

- `public static void main(String[] args)`
- ne peut être appellée que par l'exécuteur Java
- n'est pas obligatoire (e.g.: `Math`)

Conception des méthodes

Pourquoi construire des méthodes

- éviter des répétitions
- représenter les niveaux d'abstraction du problème
- simplifier le développement et la vérification

Étape 1: identifier des opérations qui

- représentent des unités naturelles du problème
- sont probablement utilisées plusieurs fois
- sont simples et peuvent être testées individuellement

Conception des méthodes (2)

Étape 2: choisir un nom

- en général, une méthode fait qqchose: verbes
- si le verbe est “calcul”, on le supprime
- exemples: `print`, `getProperty`, `equals`, `pow`, `substring`

Étape 3: choisir les paramètres

- nommés comme des variables

Étape 4: choisir le type de retour

Conception des méthodes (3)

Étape 5: Décrire l'algorithme

- paramètres $\longrightarrow$ valeur de retour
- pseudocode

Étape 6: Coder

Étape 7: Tester!!

- cas différents $\equiv$ paramètres différents
- cas simples, spéciaux, extrêmes

Exemple: calculer le prix brut

Opérations

- calculer la TVQ
- calculer la TPS
- calculer le prix brut
- arrondir un prix arbitraire

Signatures

```
public static double arrondi(double prix)
public static double tvq(double prix)
public static double tps(double prix)
public static double prixBrut(double prix)
```

Solution finale

```
public class PrixBrut {
    public static final double TAUX_TPS = 0.07;
    public static final double TAUX_TVQ = 0.08;

    public static double arrondi(double prix)
    { return Math.round(prix * 100) / 100.0; }

    public static double tps(double prix)
    { return arrondi(prix * TAUX_TPS); }

    public static double tvq(double prix) {
        double base = prix + tps(prix);
        return arrondi(base * TAUX_TVQ);
    }

    public static double prixBrut(double prix)
    { return arrondi(prix + tps(prix) + tvq(prix)); }
}
```