IFT 1010 - Programmation 1

Méthodes 2

Professeurs:
Sébastien Roy & Balázs Kégl

B. Kégl, S. Roy, F. Duranleau
Département d'informatique et de recherche opérationnelle
Université de Montréal

automne 2004

Au programme

[Tasso:5]

- Passage des paramètres
- Portée (visibilité) des variables
- Variables locales
- Variables de classe

Passage des paramètres

- L'exécution des méthodes
  - Exemple sans paramètres et valeur de retour:

```
public static void affiche() {
    System.out.println("Hello");
}

public static void main(String[] args) {
    affiche();
}
```


Passage des paramètres

- Ré-écrire l'exemple sans méthode:

```
public static void main(String[] args) {
    // DEBUT methode re-ecrite affiche()
    System.out.println("Hello");
    // FIN methode re-ecrite
}
```

- Règle 1: á l'invocation, le corps de la méthode est exécuté.

Passage des paramètres

- Exemple avec paramètres mais sans valeur de retour:

```
public static void affiche(int a,int b) {
    System.out.println(a);
    System.out.println(b);
}

public static void main(String[] args) {
    affiche(3,4);
}
```

- paramètres formels: `a` and `b`
- paramètres actuels: `3` and `4`

Passage des paramètres

- Ré-écrire l'exemple sans méthode:

```
public static void main(String[] args) {
    // DEBUT methode re-ecrite affiche(3,4)
    int a = 3;
    int b = 4;
    System.out.println(a);
    System.out.println(b);
    // FIN methode re-ecrite
}
```

- Règle 2: avant l'exécution du corps, les valeurs des paramètres actuels sont affectées aux paramètres formels.

Passage des paramètres

- Exemple lorsque les paramètres actuels sont des variables:

```
public static void affiche(int a,int b) {
    System.out.println(a);
    System.out.println(b);
}

public static void main(String[] args) {
    int i = 1, j = 2;
    affiche(i,j);
}
```

- paramètres formels: `a` and `b`, paramètres actuels: `i` and `j`

Passage des paramètres

- Ré-écrire l'exemple sans méthode:

```
public static void main(String[] args) {
    int i = 1, j = 2;

    // DEBUT methode re-ecrite affiche(i,j)
    int a = i;
    int b = j;
    System.out.println(a);
    System.out.println(b);
    // FIN methode re-ecrite
}
```

Passage des paramètres

- Même si les paramètres actuels sont des variables, seulement leurs valeurs sont passées à la méthode.

Passage des paramètres

- Exemple lorsque les noms des paramètres actuels et des paramètres formels sont identiques

```
public static void affiche(int a,int b) {
    System.out.println(a);
    System.out.println(b);
}

public static void main(String[] args) {
    int a = 1, b = 2;
    affiche(a,b);
}
```

- paramètres formels: `a` and `b`, paramètres actuels: `a` and `b`

Passage des paramètres

- Ré-écrire l'exemple sans méthode:

```
public static void main(String[] args) {
    int a = 1, b = 2;

    // DEBUT methode re-ecrite affiche(a,b)
    int aAffiche = a;
    int bAffiche = b;
    System.out.println(aAffiche);
    System.out.println(bAffiche);
    // FIN methode re-ecrite
}
```


Passage des paramètres

- Même si les noms des paramètres actuels et des paramètres formels sont identiques, ils ne sont pas les mêmes variables.

Passage des paramètres

- Exemple lorsque les paramètres actuels sont des expressions:

```
public static void affiche(int a,int b) {
    System.out.println(a);
    System.out.println(b);
}

public static void main(String[] args) {
    int i = 2;
    affiche(i+2,i*i);
}
```

- paramètres formels: `a` and `b`, paramètres actuels: `i+2` and `i*i`

Passage des paramètres

- Ré-écrire l'exemple sans méthode:

```
public static void main(String[] args) {
    int i = 2;

    // DEBUT methode re-ecrite affiche(i+2,i*i)
    int a = i+2;
    int b = i*i;
    System.out.println(a);
    System.out.println(b);
    // FIN methode re-ecrite
}
```


Passage des paramètres

- Les paramètres actuels peuvent être des expressions. Elles sont évaluées, et les valeurs des évaluations sont affectées aux paramètres formels.

Passage des paramètres

- Exemple lorsque la méthode change son paramètre formel

```
public static void sommeAffiche(int a,int b) {
    a = a + b;
    System.out.println(a);
}

public static void main(String[] args) {
    int i = 1, j = 2;
    sommeAffiche(i,j);
    // i est encore 1!!
}
```

- paramètres formels: a and b, paramètres actuels: i and j

Passage des paramètres

- Ré-écrire l'exemple sans méthode:

```
public static void main(String[] args) {
    int i = 1, j = 2;

    // DEBUT methode re-ecrite sommeAffiche(a,b)
    int a = i;
    int b = j;
    a = a + b;
    System.out.println(a);
    // FIN methode re-ecrite
    // i est encore 1, personne ne l'a change!!!
}
```


Passage des paramètres

- Même si une méthode change ses paramètres formels, les valeurs des paramètres actuels ne sont jamais changées.

Passage des paramètres

- Exemple d'une méthode avec une valeur de retour:

```
int somme(int a,int b) {
    int somme = a + b;
    return somme;
}

public static void main(String[] args) {
    int i = 1, j = 2;
    i = i + somme(i,j);
}
```

- paramètres formels: a and b, paramètres actuels: i and j

Passage des paramètres

- Ré-écrire l'exemple sans méthode:

```
public static void main(String[] args) {
    int i = 1, j = 2;

    // DEBUT methode re-ecrite i = i + somme(i,j);
    int a = i;
    int b = j;
    int somme = a + b;

    i = i + somme;
    // FIN methode re-ecrite
}
```

Passage des paramètres

- Règle 3: Une méthode avec une valeur de retour se comporte comme une expression. On peut dire que sa "valeur actuelle" est la valeur de retour calculée des paramètres actuelles de la méthode.

Portée (visibilité) des variables

- Variables locales
  - elles existent à partir de leur déclaration jusqu'à la fin du bloc
  - une variable ne peut pas être redéfinie dans la portée de la même variable

Portée (visibilité) des variables

- Variables locales
  - exemple

```
{
   int i = 0;
   if (true) {
      int i = 1; // illegal
      int j = 0;
   }
   if (true) {
      int j = 0; // legal
   }
   int j = 1; // legal
}
```


Portée (visibilité) des variables

- Variables `statics` de classe
  - elles existent pendant toute l'exécution du programme
  - elles sont déclarées dans le bloc de la classe
  - référence dans les méthodes de la même classe: `nomDeVariable` (ex.: `TAUX_TPS`)
  - référence dehors de la classe: `nomDeClasse.nomDeVariable` (ex.: `Math.PI`)
  - en principe, elles peuvent être redéclarées dans les méthodes, mais c'est fortement déconseillé
  - normalement, elles sont `final`es, les variables `statics` non-`final`es sont fortement déconseillées

Portée (visibilité) des variables

- Variables `statics` de classe (exemple)

```
public class Test {
    public static final int ZERO = 0; // legal
    public static int a = 0; // legal mais deconseille

    public static void d(double d){
        System.out.println(a); // affiche 0
        double a = 1.0; // legal mais deconseille
        System.out.println(a); // affiche 1
    }
}
```


Programme légal mais déconseillé

```
public class PrixBrutMauvais {
    public static double prix;

    public static double arrondi() {
        double prixEnSous = prix * 100;
        double prixEnSousArrondi = (int)(prixEnSous + 0.5);
        return prixEnSousArrondi / 100.0;
    }

    public static double prixNet() {
        return arrondi();
    }

    public static void main(String[] args) {
        prix = 34.589;
        System.out.println(prixNet());
    }
}
```


Portée (visibilité) des variables

- Variables non-`statics` de classe (plus tard ...)
  - elles existent pendant l'existence d'un objet de la classe
  - elles sont déclarées dans le bloc de la classe
  - référence dans les méthodes de la même classe: `nomDeVariable`
  - référence dehors de la classe: `nomDObjet.nomDeVariable`
  - en principe, elles peuvent être redéclarées dans les méthodes, mais c'est fortement déconseillé
  - normalement, elles sont `private`s, les variables non-finales `public`s sont fortement déconseillées

`for`: exception

- Programme avec `while`

```
int i = start;
while (i < end) {
   <séquence d'instructions>
   i++;
}
```

- Même programme(?) avec `for`

```
for (int i = start;i < end;i++) {
   <séquence d'instructions>
}
```