

IFT 1010 - Programmation 1

Art de programmer 2

Professeurs:
Sébastien Roy & Balázs Kégl

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

automne 2004

1

Au programme

- Modularité, cohésion, couplage d'une méthode
- Modularité des classes
- Tester une méthode

2

Rappel: pourquoi les méthodes?

Modularité

- Avantages:
 - Éviter la répétition = réutilisation du code
 - Premier niveau d'abstraction
 - Programme plus simple à comprendre
 - Faciliter la détection/correction d'erreur
 - Facilite le travail d'équipe
- Ces avantages sont garantis?
 - Non! Il faut savoir bien écrire ses méthodes.

- Cohésion:
 - chaque méthode accomplit une tâche reliée à une seule notion
- Couplage:
 - chaque méthode effectue sa tâche de façon indépendante du reste du programme

3

4

Exemple de cohésion

```
public static void circonferenceCercle()
{
    // Lecture du rayon
    System.out.print("Entrer le rayon du cercle: ");
    double rayon = Keyboard.readDouble();
    while(Keyboard.error() || rayon < 0)
    {
        System.err.println("Mauvaise valeur de rayon.");
        System.out.print("Entrer le rayon du cercle: ");
        rayon = Keyboard.readDouble();
    }

    // Calcul du circonference
    double circonference = 2 * Math.PI * rayon;

    // Affichage du circonference
    System.out.println("La circonference est " + circonference);
}
```

5

Exemple de cohésion

- Analyse:
 - lire d'abord le rayon au clavier, le valider, calculer la circonférence, l'afficher
- Problèmes de cohésion
 - trois tâches: lecture, calcul et affichage.
 - effort un peu trop grand pour comprendre
 - répétition du code
 - réutilisation, maintenabilité: si le rayon n'est pas lu au clavier mais il est le résultat d'un autre calcul

6

Exemple de cohésion

- Fractionnement en méthodes plus cohésives

```
// Lit le rayon (positif) d'un cercle au clavier.
public static double lireRayon()
{
    System.out.print("Entrer le rayon du cercle: ");
    double rayon = Keyboard.readDouble();
    while(Keyboard.error() || rayon < 0)
    {
        System.err.println("Mauvaise valeur de rayon.");
        System.out.print("Entrer le rayon du cercle: ");
        rayon = Keyboard.readDouble();
    }
    return rayon;
}
```

7

Exemple de cohésion

- Fractionnement en méthodes plus cohésives

```
// Calcule la circonference d'un cercle de rayon donne.
public static double circonferenceCercle(double rayon)
{
    return 2 * Math.PI * rayon;
}

// Affiche la circonference d'un cercle
public static void afficheCirconference(double circonference)
{
    System.out.println("La circonference est " + circonference);
}
```

8

Exemple de cohésion

- Utilisation (de façon minimaliste)

```
Cercle.afficheCirconference(Cercle.circonferenceCercle(  
    Cercle.lireRayon()));
```

- Méthodes **spécialisées**

- si cette séquence de trois tâches est **utilisée souvent**

```
public static void circonferenceCercleInteractif()  
{  
    Cercle.afficheCirconference(Cercle.circonferenceCercle(  
        Cercle.lireRayon()));  
}
```

9

Encore plus de cohésion

- **lireRayon()** n'est pas assez cohésive

- Effectue trois tâches:

- **lire** un nombre avec sollicitation
- validation 1: le nombre est **bien tapé**
- validation 2: le nombre est **positif**

- Réutilisation, maintenabilité:

- si on voulait lire **seulement avec sollicitation**
- si on voulait lire **autre chose qu'un rayon**

10

Encore plus de cohésion

- Méthodes plus réutilisables

```
// Lit un nombre réel au clavier avec un message de sollicitation.  
public static double lectureReelSollicitee(String msg)  
{  
    System.out.print(msg);  
    return Keyboard.readDouble();  
}
```

11

Encore plus de cohésion

- Méthodes plus réutilisables

```
// Lit un nombre réel au clavier avec sollicitation. La lecture  
// est recommencée tant qu'un nombre réel n'est pas tapé.  
public static double lectureReelValidee(String msg)  
{  
    double valeur = lectureReelSollicitee(msg);  
    while(Keyboard.error())  
    {  
        System.err.println("Veuillez entrer un nombre réel.");  
        valeur = lectureReelSollicitee(msg);  
    }  
    return valeur;  
}
```

12

Encore plus de cohésion

- Méthodes plus réutilisables

```
// Lit un nombre réel positif au clavier avec sollicitation. La
// lecture est recommencée tant qu'un nombre réel n'est pas tapé.
public static double lectureReelPositif(String msg)
{
    double valeur = lectureReelValidee(msg);
    while(valeur < 0)
    {
        System.err.println("Veuillez entrer un nombre réel positif.");
        valeur = lectureReelValidee(msg);
    }
    return valeur;
}
```

- Exercice: lire un nombre réel dans un intervalle quelconque

13

Encore plus de cohésion

- Utilisation

```
double rayon =
    Cercle.lectureReelPositif("Entrez le rayon du cercle: ");
```

- méthode **spécialisée**:

```
public static double lireRayon()
{
    return Cercle.lectureReelPositif(
        "Entrez le rayon du cercle: ");
}
. . .
double rayon = lireRayon();
```

14

Cohésion

- Avantages de la cohésion

- **Réutilisation** plus facile
- Méthodes **complexes moins lourdes**
- **Compréhension**: l'unicité des tâches des méthodes
- Méthodes avec tâche simple (unique) plus faciles à **déboguer**

- Exercice

- `afficheCirconference()` plus cohésive?
- tâches réutilisables?

15

Exemple de couplage

- Afficher un triangle

- rectangle isocèle
- l'angle droit en haut
- i^e ligne est remplie de la i^e lettre de l'alphabet en minuscule

- Solution **mal programmée**

- avec des **variables globales (statiques de classe)**
- **incompréhensible** à cause du couplage

16

Exemple de couplage

```
public class Triangle
{
    // Colonne courante, ligne courante et nombre total de lignes
    public static int colonne, ligne, total;

    // Affiche un triangle de nbLignes lignes.
    public static void afficheTriangle(int nbLignes)
    {
        total = nbLignes;
        for(ligne = 0; ligne < total; ++ligne)
        {
            colonne = 0;
            afficheEspaces();
            afficheLettres();
            System.out.println();
        }
    }
}
```

17

```
// Affiche les espaces au debut de la ligne courante.
public static void afficheEspaces()
{
    while(colonne < ligne)
    {
        System.out.print(' ');
        ++colonne;
    }
}

// Affiche les étoiles de la ligne courante.
public static void afficheLettres()
{
    while(colonne < total)
    {
        System.out.print((char)('a' + ligne));
        ++colonne;
    }
}
}
```

18

Exemple de couplage

- Utilisation:

```
Triangle.afficheTriangle(10);
```

- Où est le couplage?

- `afficheEspaces()` et `afficheLettres()` dépendent des variables globales `colonne`, `ligne` et `total`
- `afficheEspaces()` dépend d'`afficheTriangle()`
- `afficheLettres()` dépend d'`afficheEspaces()` et d'`afficheTriangle()`

19

Exemple de couplage

- Problèmes

- ces méthodes ne sont **pas réutilisables** (ou presque)
- **compréhension**: pour comprendre leur fonctionnement, il faut en plus comprendre le rôles des variables globales
- **maintenabilité**: le changement d'une méthode ou variable affectera le comportement d'autres méthodes

20

Exemple de couplage

- Mal-utiliser (abuser) des méthodes: **LA MAGIE!**
 - afficher une suite de n d'espaces

```
Triangle.colonne = 0;  
Triangle.ligne = n;  
Triangle.afficheEspaces();
```
 - afficher une suite de n lettres

```
Triangle.colonne = 0;  
Triangle.ligne = choixDeLaLettre;  
Triangle.total = n;  
Triangle.afficheLettres();
```
- ces programmes supposent qu'on connaît très bien le rôle de chaque variable globale

21

Exemple de couplage

- Effets de bord
 - en **changeant une variable**, le comportement d'une méthode ultérieure change (**sans changer ses paramètres**)
 - en **appelant une méthode** une variable change
- Compréhensibilité
 - **il faut lire tout le programme** afin de comprendre l'effet de l'invocation d'une méthode

22

Exemple de couplage

- Réécrire `afficheEspaces()` pour enlever son couplage

```
public static void afficheEspaces(int combien)  
{  
    int i;  
    for(i = 0; i < combien; ++i)  
        System.out.print(' ');  
}
```
- Réécrire `afficheLettres()` pour enlever son couplage
 - *afficher n espaces* est un sous-problème d'*afficher n fois un même caractère quelconque*

23

Exemple de couplage

- Généraliser `afficheEspaces()` pour `repeteCaracteres()`

```
public static void repeteCaracteres(char c, int combien)  
{  
    int i;  
    for(i = 0; i < combien; ++i)  
        System.out.print(c);  
}
```
- afficher n fois la i^e lettre:

```
Triangle.repeteCaracteres((char)('a' + i), n);
```
- afficher n espaces:

```
Triangle.repeteCaracteres(' ', n);
```

24

Exemple de couplage

- Réécrire `afficheTriangle()` pour enlever son couplage

```
public static void afficheTriangle(int nbLignes)
{
    int ligne;
    for(ligne = 0; ligne < nbLignes; ++ligne)
    {
        repeteCaracteres(' ', ligne);
        repeteCaracteres((char)('a' + ligne), nbLignes - ligne);
        System.out.println();
    }
}
```

25

Exemple de couplage

- Avantages des méthodes découplées
 - les variables globales ont disparues
 - il n'y a plus de dépendance entre méthodes
 - aucun danger d'effet de bord
 - méthodes réutilisables

26

Désavantages du couplage

- Effets de bord
- Réutilisation difficile
- Détection/correction d'erreurs plus difficile
- Compréhension des méthodes plus difficiles
- Inélégant: alourdit souvent l'écriture du code

27

Directives

- Écrire des méthodes indépendantes
- Éviter autant que possible les variables globales (statiques de classe)
- Toute information devrait être communiquée à une méthode par l'intermédiaire de paramètres
 - Exception: les méthodes qui font des lectures sur des périphériques externes (le clavier, l'écran)

28

Variables globales

- À stocker une information **générale** et **commune** dans **tout** le programme
 - Exemples: `System.out`, `System.err`, `System.in`
- À définir des **constantes générales**
 - Exemples: `Math.PI`, `Integer.MAX_VALUE`

29

Modularité des classes

- La notion de modularité ne s'applique pas qu'aux méthodes
- Une **classe** devrait aussi être **modulaire**: être **cohésive** et avec **peu de couplage**:
 - une classe devrait être reliée à une **seule notion**
 - chaque classe devrait être **indépendante**

30

Exemple de classes modulaires

```
public class Cercle
{
    public static double lectureReelSollicitee(String msg);
    public static double lectureReelValidee(String msg);
    public static double lectureReelPositif(String msg);
    public static double lireRayon();
    public static double circonferenceCercle(double rayon);
    public static void afficheCirconference(double circonference);
}
```

- Cette classe **n'est pas cohésive**:
 - les trois **méthodes de lecture** n'ont rien à voir avec la notion de **cercle**

31

Exemple de classes modulaires

- On se crée une classe par "module"

```
public class Cercle // Classe pour la notion de cercle
{
    public static double lireRayon();
    public static double circonferenceCercle(double rayon);
    public static void afficheCirconference(double circonference);
}
```

```
public class Lecture // Classe pour la notion de lecture
{
    public static double lectureReelSollicitee(String msg);
    public static double lectureReelValidee(String msg);
    public static double lectureReelPositif(String msg);
}
```

32

Exemple de classes modulaires

- `Cercle.lireRayon()`:

```
public static double lireRayon()
{
    return Lecture.lectureReelPositif(
        "Entrez le rayon du cercle: ");
}
```

- Noms redondants:

- `Cercle.circonferenceCercle()`
→ `Cercle.circonference()`
- `Triangle.afficheTriangle()` → `Trangle.afficher()`

33

Exemple

- Avec les deux classes `Lecture` et `Cercle`, on aurait deux fichiers:

- `Lecture.java` pour la classe `Lecture`
- `Cercle.java` pour la classe `Cercle`

- Chacun est **compilé séparément**:

- `javac Lecture.java`
- `javac Cercle.java`

35

Séparation en fichier

- Chaque classe dans **un fichier** propre à elle
- Rappel: chaque fichier doit porter **le même nom** que la classe plus l'extension `.java`
- Un fichier ne peut contenir qu'**une classe publique**
- **CLASSPATH**: une **variable d'environnement** qui contient une suite de **répertoires** où le compilateur (et la commande `java`) cherche la **définition des classes**
- Chaque fichier doit être **compilé séparément**

34

Tester une méthode

- Pour tester les méthodes, écrire une classe à part qui appellera les méthodes.

```
public class TestTriangle
{
    public static void main(String[] arg)
    {
        Triangle.afficheTriangle(Integer.parseInt(arg[0]));
    }
}
```

- Pourquoi?

- **éviter les modifications** aux classes mêmes
- faciliter le **travail d'équipe**: plusieurs personnes peuvent faire des tests **en même temps**

36

Construction de tests simples

- Scénario
 - **MaMethode**: la méthode à tester
 - **conversion i (String arg)**: conversion vers un type désiré pour le i^e paramètre de la méthode **MaMethode**
- Tester une méthode d'affichage ou interactive

```
public class TestMaMethode
{
    public static void main(String[] arg)
    {
        MaMethode(conversion0(arg[0]),conversion1(arg[1]),...);
    }
}
```

37

Exemples

- Tester **Lecture.lectureReelValidee()**
 - pas de conversion, le paramètre est de type **String**
- ```
public class TestLectureReelValidee
{
 public static void main(String[] arg)
 {
 System.out.println(Lecture.lectureReelValidee(arg[0]));
 }
}
```
- compilation: `javac TestLectureReelValidee.java`
  - exécution: `java TestLectureReelValidee "Entrez un nombre reel: "`

39

## Construction de tests simples

- Tester le retour d'une méthode

```
public class TestMaMethode
{
 public static void main(String[] arg)
 {
 System.out.println(MaMethode(conversion0(arg[0]),
 conversion1(arg[1]),
 ...));
 }
}
```

- Compilation: `javac TestMaMethode.java`
- Exécution: `java TestMaMethode arg0 arg1 ...`

38

## Exemples

- Tester **Cercle.circonference()**
    - conversion, **String** → **double**
- ```
public class TestCirconference
{
    public static void main(String[] arg)
    {
        System.out.println(Cercle.circonference(
            Double.parseDouble(arg[0])));
    }
}
```
- compilation: `javac TestCirconference.java`
 - exécution: `java TestCirconference 2.15`

40

Tests plus complexes

- **Boucles** pour tester la méthode systématiquement pour **plusieurs valeurs** des paramètres
- Pour les méthodes qui utilisent **des entrées et des sorties**, construire des fichiers d'entrées et procéder par **redirection**