Le langage Java

Approche orientée objet – Encapsulation et association

Programme détaillé ou sommaire

L'encapsulation
- Encapsulation des variables
- Règles de visibilité
- Problématique
- Exemple

L'Association
- Définition
- Références multiples

Chapitre 2
L'encapsulation

Définition et exemples

Encapsulation des variables (1/2)

Méthodes getter et setter

Bonne pratique : déclarer les attributs d'une classe comme privés.
Utiliser des méthodes getter et setter pour accéder aux attributs.

```
public class Avion {
  private long matricule;

  public long getMatricule() {
    return matricule;
  }

  public void setMatricule(long l) {
    matricule = l;
  }
}
```

Tous les environnements de développement proposent des assistants pour générer les getters-setters

Encapsulation des variables (2/2)

Intérêt des getters et setters ?

Il est possible de jouer sur la visibilité des méthodes d'accès.

Ex : variable en lecture seule en dehors du package.

Visibilité package

```
public class Avion {
  private long matricule;

  public long getMatricule() {
    return matricule;
  }

  void setMatricule(long l) {
    matricule = l;
  }
}
```

Un traitement spécifique peut être placé à chaque accès à une variable.

Message de log…

Règles de visibilité (1/2)

Attributs et méthodes : plusieurs niveaux de visibilté

public :élément visible partout (mot-clé 'public').
private : élément à usage interne de la classe. Invisible de l'extérieur (mot-clé 'private').
package : élément visible partout dans le même package (pas de mot-clé, cas par défaut).

```
public class Test {
    public int var1; // attribut public
    private int var2; // attribut privé
    int var3; // attribut de visibilité 'package'

    public Test() {} // constructeur public

    private int method1() {} // méthode privée

}
```

Le quatrième niveau de visibilité (protected) sera vu plus loin dans les concepts objets (avec l'Héritage)

Règles de visibilité (2/2)

Classes : deux niveaux de visibilté

public :classe visible partout (mot-clé 'public').

package : classe visible dans son package uniquement

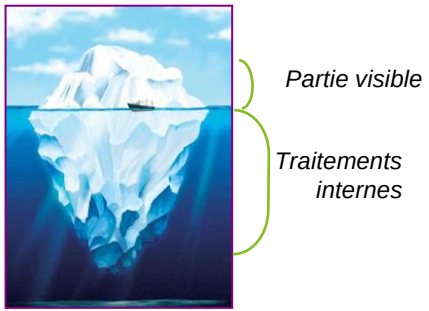
```
public class Test {
   // classe publique utilisable
   partout
}
```

```
class Test {
   // classe de visibilité package
   utilisable seulement par des
   classes situées dans le même
   package
}
```

Restriction d’accès à certaines méthodes

Une classe doit être

- Claire et simple d’utilisation
- Complexité masquée à l'intérieur



But de l’encapsulation:

- Restreindre la visibilité des traitements internes

Exemple

```java
public class RecetteAeroport {
  public long calculerRecetteTotale() {

      //appel à toutes les autres méthodes
  }

  public long calculerVentesBilletsAvion() {…}
  public long calculerTaxesAeroport() {…}
  public long calculerRecetteRestaurants() {…}
}
```

Classe de calcul de la recette

```java
public class Test {
  public static void main(String[] args) {
    RecetteAeroport r = new RecetteAeroport();

    // Toutes les méthodes sont accessibles même celles
    qui sont inutiles
    float recette = r.calculerRecetteTotale();
    float billets = r.calculerVentesBilletsAvion();
    float taxes = r.calculerTaxesAeroports();
  }
```

Appelant

calculerRecetteTotale() fait appel aux autres méthodes de la classe

Les autres méthodes sont à usage interne de la classe.

Problème : la classe Test a accès à ces autres méthodes…

Exemple

Solution : les méthodes à usage interne deviennent privées

Elles ne sont plus visibles de la classe Test

```
public class RecetteAeroport {
   public long calculerRecetteTotale() {
       //appel à toutes les autres méthodes
   }

   private long calculerVentesBilletsAvion() {…}
   private long calculerTaxesAeroport() {… }
   private long calculerRecetteRestaurants() {…}
}
```

Classe de calcul de la reccette

```
public class Test {
  public static void main(String[] args) {
    RecetteAeroport r = new RecetteAeroport();
    float recette = r.calculerRecetteTotale();
   }
}
```

Appelant

Chapitre 2
L'association

Définition et exemples

Définition

Il a été vu qu'une classe peut contenir des attributs de type primitif

```
public class Avion {
  private long matricule;
  //…
}
```

Une classe peut également porter des attributs typés par d'autres classes.

```
public class Avion {
  private long matricule;
  private Moteur moteur;

  public Moteur getMoteur() {
     return moteur;
  }
  public void setMoteur(Moteur moteur) {
     this.moteur = moteur;
  }
  //…
}
```

```
public class Moteur {
  private int identifiant;
  private String dateRevision;
  //…
}
```

Références multiples

Que produit le code suivant ?

```
Avion monAvion = new Avion();
Moteur monMoteur = new Moteur();
monAvion.setMoteur(monMoteur);

monMoteur.setDateRevision("21/02/2007");

System.out.println(monAvion.getMoteur().getDateRevision());
```

Les deux références pointent vers la même zone mémoire.

Atelier (TP)

OBJECTIFS : mettre en œuvre l'encapsulation

DESCRIPTION : Dans le TP suivant vous allez être amené à mettre en place les règles de l'encapsulation.