JAVA

Encapsulation =

- Regrouper dans une seule et même entité (ex: les objets informatique) les données et les traitements (qui agissent sur ces données) en cachant l'implémentation de l'objet
    - Regroupement des caractéristiques d'un objet
      => données(=attributs= + traitements (=méthodes)
    - dissimuler les détails de l'implémentation
      => abstraction
- meilleure : visibilité + cohérence + plus grande modularité
- cadre plus rigoureux : les modifs de la structure interne restent invisibles à l'extérieur

Héritage =

- Définir une classe qui regroupe les points communs des 2 classes au lieu de définir 2 classes qui n'ont rien en commun
- Définir des sous-classes pour gére des les détails/spécificités
  Avantages :
    - organiser le code en évitant des duplications
    - créer une hiérarchie de classes (réutiliser le code dans les sousClasses)
    - faciliter la maintenance en apportant des modifications uniquement à la super classe

Re-définition - Overriding =

- Méthode pour laquelle les paramètres et leur types sont identiques, et les types de retour compatibles (types de bases identiques ou relation d'héritage)

Surcharge (Overload) =

- le nom de la méthode est aussi identique mais les paramètres ne le sont pas

---

Méthodes

```
tye_de_retour nom_methode(type_param1 nom_param1, ..., type_param? nom_paramN) {
    /* Corps de la méthode*/
};
```

Accesseurs ou Getters

retourne la valeur d'une variable

```
public typeAttribut getNomAttribut() {
    return attribut;
};
```

Mutateurs ou Setters ou Manipulateur

modifie l'état d'un attribut

```
public void setNomAttribut(typeAttribut a) {
    attribut = a;
};
```

Constructeur

méthode spécifique en charge de l'initialisation des attributs

```
public NomClasse(typeAttribut1 Attribut1, ..., typeAttributN AttributN) {
    hauteur=h;
    /*exemple d'initialisation*/
};
```

Constructeur par défaut

```
Patient peter = new Patient();
```

Méthode init

```
public void init(double h, double m) {
    hauteur = h;
    masse = m;
};
```

Méthode toString

```
public String toString() {
    return "chaîne de caractère" + "une autre chaîne de caractère";
};
```

Méthode équals

égalité de chaines de caractères

```
public boolean equals(ClasseBook b) {
    if(//condition){
        return false;
    } else {
        return title.equals(b.title)&& author.equals(b.author);
        }
    }
```

ArrayList

- maintient le nombre d'éléments, et récupérer cette valeur grace à size()
- elle cache les détails d'implémentation

- elle respecte l'ordre des objets selon l'insertion

```
import java.util.ArrayList;

//Déclaration
ArrayList<nomdelaClasse> nomVariable;

//Initialisation, instance
nomVariable = new ArrayList<nomdelaClasse>();

//Ajouter un élément à la fin d'une liste
nomVariable.add(b1);

//afficher un élément d'une ArrayList
for(int i=0; i<nomVariable.size(); i++) {
    System.out.println("le nb d'élément"+i+" est "+nomVariable.get(i).toString());
};
```

Héritage

- première ligne quand il y a une extend : **super(param1, …, paramN);**

  class NomSousClasse extends NomSuperClasse {
  / *déclaration de nouveaux attributs* /

  ```
  public NomSousClasse(liste paramètres) {

  super(arguments);
  //ou
  super.uneMéthodelaplusproche();

  //initialisation des attributs de NomSousClasse
  }
  //déclaration de nouvelles méthodes
  ```

  };

Méthode abstraite

- méthode consituée d'une en-tête de méthode sans corps de méthode
- la présence d'une méthode abstraite rend la classe abstraite

```
abstract typeRetour nomMéthode(liste_paramètres);
```

Classe abstraite

- unique objectif : servir de super-classe
  - ses sous-classes sont aussi abstraites tant quelles ne définissent pas toutes les méthodes abstraites
- une classe concrète est une classe non-abstraite
- une classe peut être abstraite sans contenir de méthodes abstraites

```
abstract class NomClasseAbstraite {
    //corps de la classe
};
```

Modificateur final

- **variable final** ne peut être modifiée (constante)
- **méthode final** ne peut pas être redéfine
- **classe final** ne peut être étendue

Modificateur static

- pour une méthode, peut être appelée sans instancier sa classe
- pour un attribut, la valeur est partagée entre les différentes instance de la classe

syntaxe

```
static int x = 10;
```

Interface

- attribue des composants communs à des composants non liées par une relation d'héritage
- impose à certaines classes d'avoir un contenu particulier sans que ce contenu ne fasse partie de la classe

interface ≠ classe

Une interface contient uniquement des :
constantes (`public final static <constante>`)
méthodes abstraites visible (`public abstract`)
méthodes static visibles(`public static`)
définitions de méthodes par défaut

Une interface ne peut pas être instanciée : elle ne contient pas de constructeurs
Une classe qui implémente une interface reçoit son type

Règles :

- plusieurs classes oeuvent implémenter une même interface
- une classe peut hériter d'une seule classe
- une classe peut implémenter plusieurs interfaces

Une classe qui :

- implémente toutes les méthodes d'une inteface est **concrète**
- n'implémente pas toutes les méthodes de l'interface est **abstraite**
