PROGRAMMATION ORIENTÉE OBJET JAVA

LICENCE 2 INGÉNIERIE - INFORMATIQUE

2020– 2021

Marie NDIAYE

CLASSES ET OBJETS

L'APPROCHE OBJET

- **Programmation dirigé par les données et non par les traitements.**
  - Les procédures existent toujours mais on se concentre d'abord sur les entités que l'on va manipuler avant de se concentrer sur la façon dont on va les manipuler.
- **Notion d'encapsulation**
  - les données et les procédures qui les manipulent (on parle de méthodes) sont regroupés dans une même entité (la classe).
  - L'accès aux données est règlementé (***public, protected, private***)

LES CLASSES

REGROUPEMENT DES OBJETS

- Beaucoup d'objets pouvant collaborer dans une application
- Mais on peut le plus souvent dégager des types d'objets (structures et comportements identiques)
  - Ex : tous les livres dans une application de gestion de bibliothèque
- La notion de classe correspond à cette notion de types d'objets.

```
class Livre {
    String titre;
    Lecteur emprunteur;

    void setTitre(String t) {
        titre = t;
    }

    String getTitre() {
        return titre;
    }

    Date emprunte(Lecteur lec) {
        if (emprunteur = = null) {
            emprunteur = lec;
            return new Date();
        } else return null;
    }
}
```

Attributs

Méthodes

LES COMPOSANTS D'UNE CLASSE

- Les **constructeurs** (il peut y en avoir plusieurs) servent à créer des objets.
- Les **méthodes** déterminent le comportement des instances de la classe quand elles reçoivent un message.
- Les **attributs** (**variables**) représentent l'état de l'objet.

CONSTRUCTEUR

- Un constructeur est une méthode qui a le même nom que la classe.
- Un constructeur n'a pas de valeur de retour (même pas `void`).
- Plusieurs constructeurs peuvent exister dans une même classe (avec des arguments différents).
- Il faut au moins un constructeur dans une classe pour en instancier des objets.

CONSTRUCTEUR DE LA CLASSE LIVRE

```
class Livre {
    String titre;
    Lecteur emprunteur;

    Livre(String t) {
        titre = t;
        emprunteur = null;
    }

    void setTitre(String t) {
        titre = t;}

    String getTitre() {
        return titre;}

    Date emprunte(Lecteur lec) {
        …    }
}
```

Constructeur

CONSTRUCTEUR PAR DÉFAUT

- Lorsque le code d'une classe ne comporte pas de constructeur, un constructeur sera automatiquement ajouté par Java.
- Pour une classe Livre, ce constructeur par défaut est :

```
Livre( ) { }
```

MÉTHODES

- Les **accesseurs** : getXXX (), setXXX()
  - Deux types de méthodes servent à donner accès aux variables depuis l'extérieur de la classe :
    - Les ***getters*** (accesseurs en lecture) pour lire les variables

      String getTitre()
    - Les ***setters*** (accesseurs en écriture) pour modifier leurs valeurs.

      void setTitre(String t)
- Les **autres types** de méthodes
  - Offrent des services plus complexes aux autres instances
- Les méthodes *private* servent de sous-programmes utilitaires aux autres méthodes de la classe.

EXEMPLE

```
public class Employe {
   private double salaire;
   . . .
   public void setSalaire(double unSalaire) {
            if (unSalaire >= 0.0)
            salaire = unSalaire;
   }
   public double getSalaire() {
            return salaire;
   }
   . . .
}
```

Setter

Getter

PARAMÈTRES D'UNE MÉTHODE

- Indiquer le type des paramètres dans la déclaration des méthodes ou des constructeurs

  Date emprunte(Lecteur lec)

- Quand il n'y a pas de paramètre, on ne met rien entre les parenthèses :

  void afficheLivre( )

TYPE DE RETOUR D'UNE MÉTHODE

- Quand une méthode renvoie une valeur, on doit indiquer le type de la valeur renvoyée dans la déclaration :

  Date emprunte(Lecteur lec) { …}

- Le pseudo-type `void` indique qu'aucune valeur n'est renvoyée :

  void afficheLivre() { … }

LES VARIABLES

- Les **variables d'instances** ("globales à la classe")
  - sont déclarées en dehors de toute méthode
  - conservent l'état d'un objet, instance de la classe
  - sont accessibles et partagées par toutes les méthodes de la classe
- **Les variables locales**
  - sont déclarées à l'intérieur d'une méthode ou d'un bloc
  - ne sont accessible que dans le bloc dans lequel elles sont déclarées
- **Les variables de classe** (à voir plus loin)

VARIABLE D'INSTANCE

```
public class Portee {
    int a;

    public void test(int b) {
        if (a > b) {
            int c = b;
            b = a;
            a = b;
        }
        afficher(b);
    }
}
```

l'entier ***a*** est visible dans toute la classe

VARIABLE LOCALE (MÉTHODE)

```
public class Portee {
    int a;

    public void test(int b) {
        if (a > b) {
            int c = b;
            b = a;
            a = b;
        }
        afficher(b);
    }
}
```

l'entier ***b*** est visible à l'intérieur de la méthode test

VARIABLE LOCALE (BLOC)

```
public class Portee {
    int a;

    public void test(int b) {
        if (a > b) {
            int c = b;
            b = a;
            a = c;
        }
        afficher(b);
    }
}
```

l'entier ***c*** est visible dans le bloc du *if*

LES OBJETS

NOTION D'OBJET EN JAVA

- Un objet a
  - Une **adresse** en mémoire (identifie l'objet).
  - Un comportement (ou interface), donné par les procédures ou fonctions, appelées **méthodes**.
  - Un état interne (donné par les valeurs des **variables**).

CRÉATION D'OBJETS

- La création d'objet à partir d'une classe est appelée **instanciation**. L'objet créé est une instance de la classe.
- L'instanciation se décompose en 3 phases :
  - Obtention de l'espace mémoire nécessaire à la partie dynamique de l'objet et initialisation des attributs en mémoire (à l'image d'une structure).
  - Appel de méthodes particulières, les constructeurs, définies dans la classe.
  - Renvoi d'une référence sur l'objet (son identité) maintenant créé et initialisé.

INSTANCIATION D'UNE CLASSE

- L'instanciation est l'opération qui consiste à créer un objet à partir d'une classe.
- En Java, le mot-clé **new** provoque une instanciation en faisant appel à un **constructeur** de la classe instanciée.
- Une fois qu'elle est créée, l'instance
  - a son propre état interne (les valeurs des variables)
  - partage le code qui détermine son comportement (méthodes) avec les autres instances de classe.

INSTANCIATION DE LA CLASSE LIVRE

```
class InterfaceBibliothecaire {

        void enregistreLivre(String nom) {
                Livre nouveauLivre;
                nouveauLivre = new Livre(nom);
                ajouteEnBibliotheque(nouveauLivre);
        }

        void ajouteEnBibliotheque(Livre li) {
                …
        }
}
```

INVOCATION D'UNE MÉTHODE

- En Java, une méthode ne peut pas être invoquée seule, elle est toujours appelée sur un objet.
- Un point (.) sépare le nom de la méthode de l'objet sur lequel elle est invoquée.

```
Livre monLivre = new Livre("Germinal");
String titreDuLivre = monLivre.getTitre();
```

- Le mot-clé **this** désigne, en cours d'exécution d'une méthode, l'objet sur lequel elle est appelée.
- La syntaxe pour accéder aux attributs d'un objet est la même.

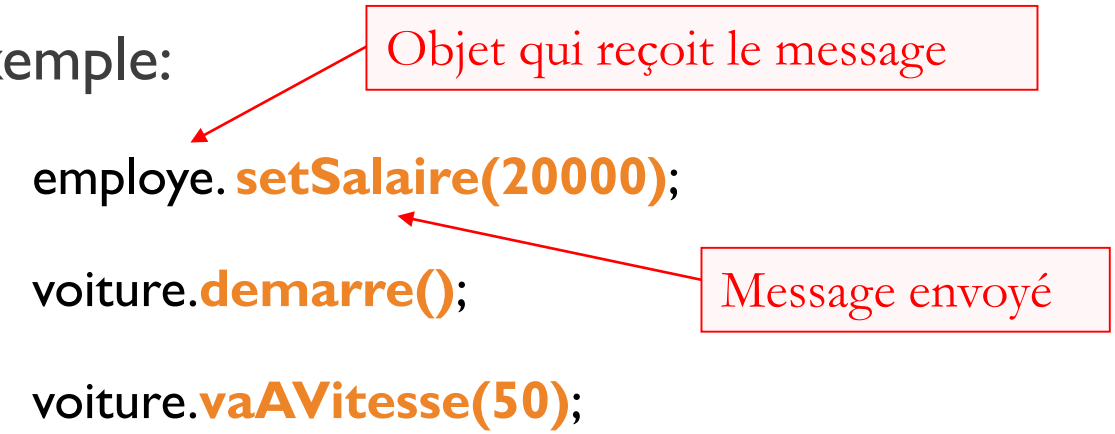
EXEMPLE

```
class Livre {
    String titre;
    Lecteur emprunteur;
    …
    boolean estEmprunte() {
        if (emprunteur == null) return false;
        else return true;
    }

    Date emprunte(Lecteur lec) {
        if ( this.estEmprunte() == true)
            return null;
        if ( lec.empruntPossible() ) {
            emprunteur = lec;
            lec.ajouteEmprunt(this);
            return new Date();
        } else return null;
    }
}
```

```
class Lecteur {
    Livre[ ] emprunts;
    int nbEmprunts;
    …
    boolean empruntePossible() {
        if (nbEmprunts < 5) return
    true;
        else return false;
    }

    void ajouteEmprunt(Livre liv) {
        emprunts[nbEmprunts] = liv;
        nbEmprunts ++;
    }
}
```

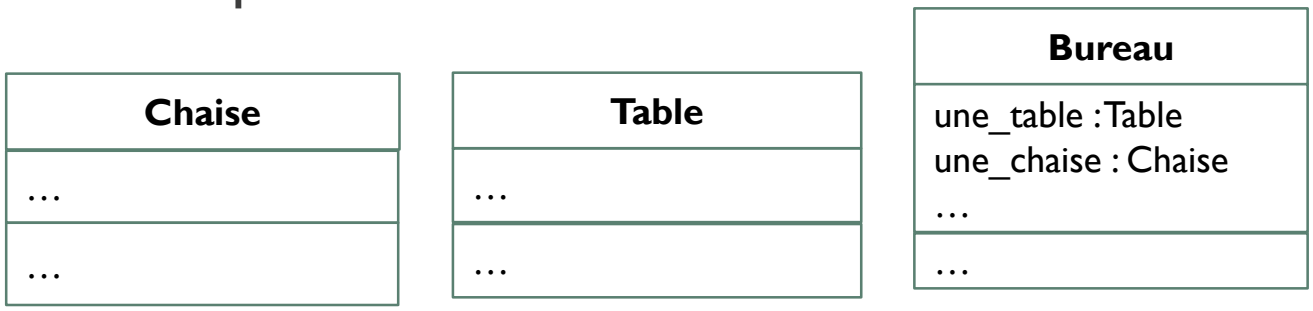
LES INTERACTIONS ENTRE OBJETS

- Les objets interagissent en s'envoyant des messages synchrones
- Les méthodes d'un objet correspondent aux messages qu'on peut lui envoyer : quand un objet reçoit un message, il exécute la méthode correspondante.
- Exemple:



AGRÉGATION : RAPPEL

- L'agrégation est une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe.
- L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes.
- Exemple :



AGRÉGATION : MISE EN ŒUVRE EN JAVA

```
class Point{
    int x ;
    int y ;
    … }
```

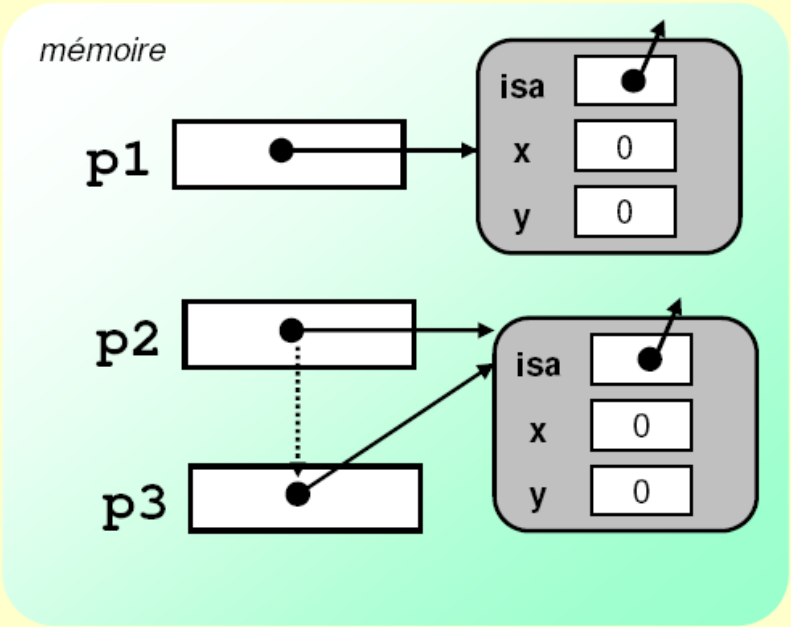
```
class Triangle {
    Point sommet1;
    Point sommet2;
    Point sommet3;
    …
}
```

```
class Carre{
    Point sommet ;
    int cote ;
    …
}
```

GESTION DE LA MÉMOIRE 1/3

- L'instanciation provoque une allocation dynamique de la mémoire

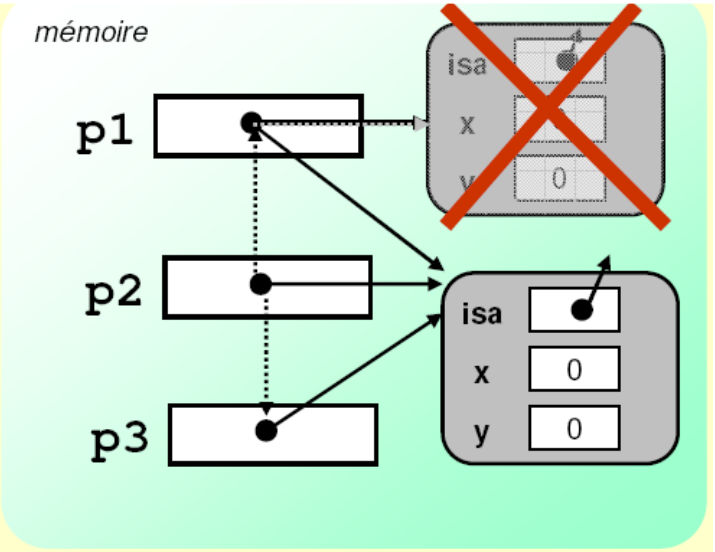
```
Point p1;
p1 = new Point();
Point p2 = new Point();
Point p3 = p2;
```



GESTION DE LA MÉMOIRE 2/3

- Objet non référencé : Le GC (***Garbage collector***) ou ramasse-miettes se charge de libérer la mémoire occupée (destruction asynchrone ou appel explicite: System.gc())

```
Point p1;
p1 = new Point();
Point p2 = new Point();
Point p3 = p2;
p1 = p2;
```



GESTION DE LA MÉMOIRE 3/3

- Le GC (ramasse-miettes) est une tâche qui
  - travaille en arrière plan
  - libère de la place occupée par les instances non référencées
- Il intervient
  - quand le système a besoin de mémoire
  - ou de temps en temps, avec une priorité faible

EXERCICE D'APPLICATION

SEGMENT DE DROITE (1/4)

- Il s'agit de modéliser un segment de droite dont les valeurs des abscisses des deux extrémités sont entières. Les opérations que l'on souhaite faire sur ce segment sont :
  - calculer sa longueur ;
  - savoir si un point d'abscisse donné se trouve sur le segment (c'est-à-dire si son abscisse est comprise entre la plus petite et la plus grande valeurs des abscisses des extrémités du segment).

SEGMENT DE DROITE (2/4)

- Écrire le code d'une classe publique Segment se trouvant dans un paquetage de nom segment comportant :
  - deux attributs privés de type int, `extr1` et `extr2`, représentant les abscisses (entières) des extrémités d'un segment sur un axe ; la classe fera en sorte que extr1 soit toujours au plus égal à extr2 ;
  - un constructeur de ce segment recevant en arguments les deux valeurs entières des abscisses des extrémités du segment que l'on veut construire ;
  - une méthode privée nommée `ordonne` échangeant éventuellement les valeurs des extrémités du segment de telle sorte que la valeur de extr1 soit au plus égale à la valeur de extr2. Cette méthode sera appelée par le constructeur après l'initialisation des deux extrémités.
  - une méthode `calculeLongueur` publique retournant la longueur du segment ;

SEGMENT DE DROITE (3/4)

- une méthode dont le prototype est : `public boolean appartient(int x);` indiquant si le point de coordonnée x appartient ou non au segment ;
- le getter `public int getExtr1();`
- le setter `public void setExtr1(int a);`
- le getter `public int getExtr2();`
- le setter `public void setExtr2(int a);`
- une méthode d'en-tête `public String toString()` qui redéfinit la méthode toString de la classe Object ; cela sera sans doute vu plus tard. Cette méthode décrira une instance de Segment sous la forme d'une chaîne de caractères, c'est-à-dire d'un objet de type String ; pour le segment d'extrémités -35 et 44, cette chaîne pourrait être : `"segment [-35, 44]"`. La méthode "retournera" (return...) cette chaîne.

SEGMENT DE DROITE (4/4)

- Vous définirez aussi dans le paquetage segment une classe `TestSegment` pour tester la classe Segment. Cette classe comportera une méthode `main` à laquelle vous devrez fournir trois paramètres entiers par la ligne de commande : abcisses des deux extrémités d'un segment et abscisse d'un point dont on voudra savoir s'il appartient ou non au segment.
- Dans la méthode main :
  - créez une instance de la classe `Segment` à partir des deux premiers entiers donnés sur la ligne de commande;
  - Afficher le segment en utilisant explicitement puis implicitement la méthode `toString()`.
  - Tester si le troisième argument appartient au segment;
  - Invoquez les autres méthodes de la classe `TestSegment`.