

BURKINA FASO

UNITÉ-PROGRÈS-JUSTICE

MASTER2: RÉSEAUX INFORMATIQUES ET MULTIMÉDIA

PROGRAMMATION MOBILE

PROGRAMMATION ORIENTEE OBJET

COMPAORE MOCTAR

20 September 2022

Agenda

1. La POO
2. Pourquoi la POO ?
3. Objet & Classe
4. Classe et classe abstraite
5. Classe et Interface
6. Héritage simple de classe
7. Encapsulation
8. Surcharge d'une méthode
9. Polymorphisme
10. Associations, cardinalités et navigabilité

La POO

- ❖ À la différence de la programmation procédurale, un programme écrit dans un langage objet répartit l'effort de résolution de problèmes sur **un ensemble d'objets collaborant par envoi de messages.**

Pourquoi le POO ?

- ❖ Le code est plus sûr
- ❖ Les programmes sont plus clairs
- ❖ La maintenance des applications est facilitée
- ❖ Le code est facilement réutilisable
- ❖ Il est facile de créer de nouveaux algorithmes légèrement différents par clonage d'un algorithme existant
- ❖ Il est facile de faire évoluer des programmes

Objet & Classe

La 1ère étape consiste à déterminer

- les entités que l'on souhaite manipuler
- la description générique qui relie toutes ses entités.

Objet & Classe

Objet (Etat + Comportement) :

- ❖ Personnes, lieux, ...
- ❖ Composés d'un état (propriété, données) et dotés de comportements (opérations, méthodes ...)
- ❖ Peuvent opérer directement sur leurs données
- ❖ Peuvent envoyer des messages les uns aux autres

Objet & Classe

Prenons ces informations :

- ❖ 5 Août 1960 (Proclamation de l'indépendance)
- ❖ 11 Décembre 1960 (Fête de l'indépendance)

Ce sont des dates et chaque date se caractérise par :

- ❖ un jour
- ❖ un mois
- ❖ une année.

Objet & Classe

Date

- Jour
- Mois
- Année

ProclamationIndependance

- 5
- aout
- 1960

DateDuJour

- 06
- avril
- 2022

FeteIndependance

- 11
- Decembre
- 1960

Objet & Classe

- Le *Jour*, le *Mois* et l'*Année* sont les **attributs** d'une Date.
- Cet ensemble d'attributs est appelé une **Classe**.

Objet & Classe

Classe :

- ❖ Une structure contenant les données et les comportements communs à un ensemble d'objets qu'elle décrit
- ❖ Chaque objet est une instance d'une classe
 - ❖ Le 5 aout 1960 et le 11 décembre 1960 sont chacune des instances de la classe **Date**.

Objet & Classe

Exemple1 : les planètes

- ❖ Saturne : planète gazeuse. Son diamètre est de 120.536 km. Elle est à une distance moyenne de 1.426.725.400 km du Soleil.
- ❖ Mars : planète rocheuse. Son diamètre est de 6794 km. Elle est à une distance moyenne de 227.936.640 km du Soleil.
- ❖ Jupiter : planète gazeuse. Son diamètre est de 142.984 km. Elle est à une distance moyenne de 779 millions de km du Soleil.
- ❖ Terre : planète rocheuse. Son diamètre est de 12.756,28 km. Elle est à une distance moyenne de 150.000.000 km du Soleil.

Objet & Classe

Exemple1 : les planètes

Planete

- *Type*
- *DistanceAuSoleil*
- *Diametre*

Terre

- Rocheuse
- 150000000
- 12756,28

Saturne

- Gazeuse
- 1426725400
- 120536

Jupiter

- Gazeuse
- 779 millions
- 142984

Mars

- Rocheuse
- 227936640
- 6794

Objet & Classe

Exemple1 : les planètes

```
public class Planete {  
    public String type;  
    public String distanceAuSoleil;  
    public String diametre;  
}
```

Planete

- *Type*
- *DistanceAuSoleil*
- *Diametre*

Objet & Classe

Exemple1 : les planètes

```
Planete terre = new Planete();
```

```
terre.type = 'Rocheuse';
```

```
terre.distance = '150000000';
```

```
terre.diametre = '12756,28';
```

Objet & Classe

Exemple2 : les étudiants

❖ Goama,

- ✓ 19 ans, étudiant en Comptabilité. groupe 1, sous-groupe 2.

❖ Noaga,

- ✓ étudiante en informatique, a 18 ans. Elle est dans le sous-groupe 1 du Groupe 2.

❖ Tiiga,

- ❖ un garçon de 21 ans, est en Electronique. Il est dans le groupe 3, sous-groupe 2.

Objet & Classe

Exemple2 : les étudiants

Ici, un étudiant se caractérise par:

- age
- département
- groupe
- sous-groupe
- sexe

Objet & Classe

Exemple2 : les étudiants

Etudiant
<ul style="list-style-type: none">• Age• Departement• Groupe• SousGroupe• Sexe

Goama
<ul style="list-style-type: none">• 19• compta• 1• 2• Garçon

Tiiga
<ul style="list-style-type: none">• 21• electro• 3• 2• Garçon

Noaga
<ul style="list-style-type: none">• 18• INFO• 2• 1• Fille

Objet & Classe

Exemple2 : les étudiants

```
public class Etudiant {  
    ...  
}
```

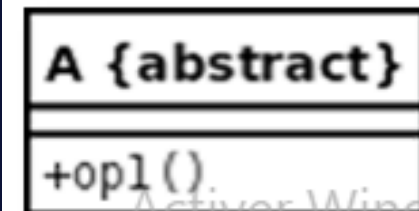
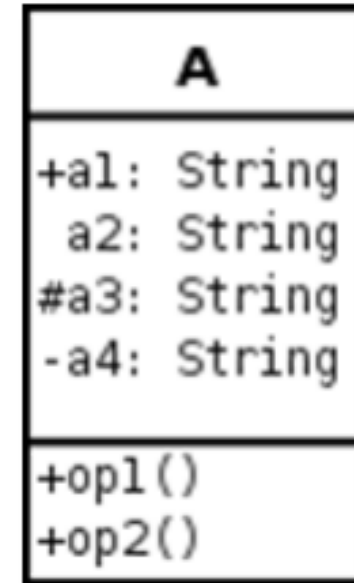
Etudiant

- Age
- Departement
- Groupe
- SousGroupe
- Sexe

Classe et classe abstraite

```
public class A {  
    public String a1;  
    package String a2;  
    protected String a3;  
    private String a4;  
    public void op1() { ... }  
    public void op2() { ... }  
}
```

```
public abstract class A {  
    ...  
    public abstract void op1();  
}
```



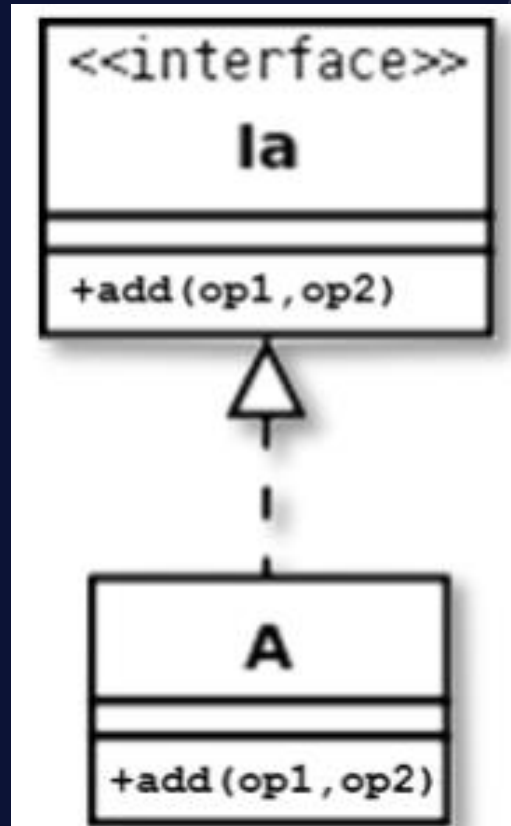
Classe et Interface

Interface :

Définitions de méthodes (sans implémentation) et de valeurs constantes (classe abstraite/héritage multiple ...)

```
public interface Ia {  
    ...  
    public int add(int op1, int op2);  
}
```

```
public class A implements Ia {  
    ...  
    public int add(int op1, int op2) { ... }  
}
```



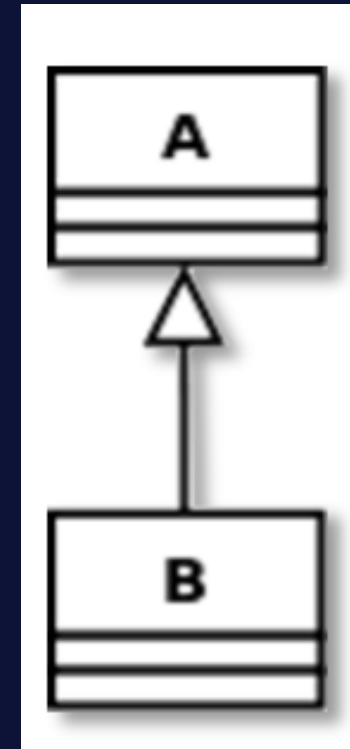
Héritage simple de classe

Interface :

Définitions de méthodes (sans implémentation) et de valeurs constantes (classe abstraite/héritage multiple ...)

```
public class A {  
    public int add(int op1, int op2) { ... }  
}
```

```
public class B extends A {  
    public int subtract(int op1, int op2) { ... }  
}
```



Encapsulation

Accesseurs :

- ❖ protéger l'information contenue dans un objet
- ❖ ne proposer que des méthodes de manipulation de cet objet (getters et setters)

Modificateur	Classe	Package	Sous-classe	Partout
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
<i>no modifier</i>	✓	✓	✗	✗
private	✓	✗	✗	✗

Surcharge d'une méthode

Accesseurs :

- ❖ même nom de méthodes avec différents paramètres : type et/ou nombre
- ❖ la surcharge des operateurs n'existe pas sous java

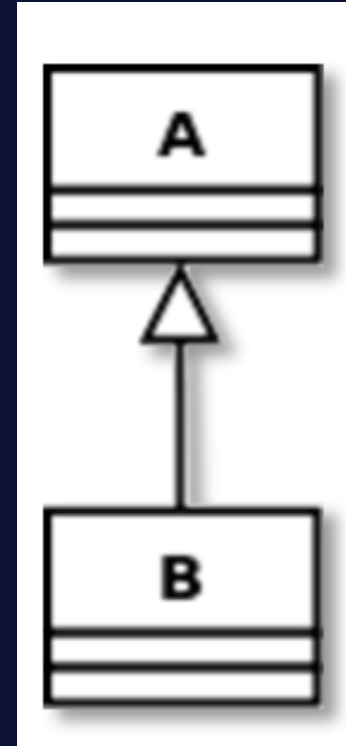
```
public class A {  
    ...  
    public int add(int op1, int op2) { ... }  
    public float add(float op1, float op2) { ... }  
    ...  
}
```

Redéfinition d'une méthode

- ❖ Ecraser dans la sous classe la définition d'une méthode de la superclasse
- ❖ L'annotation (mot clé spécial) : `@Override`

```
public class A {  
    public int add(int op1, int op2) { ... }  
}
```

```
public class B extends A {  
    @Override  
    public int add(int op1, int op2) { ... }  
}
```

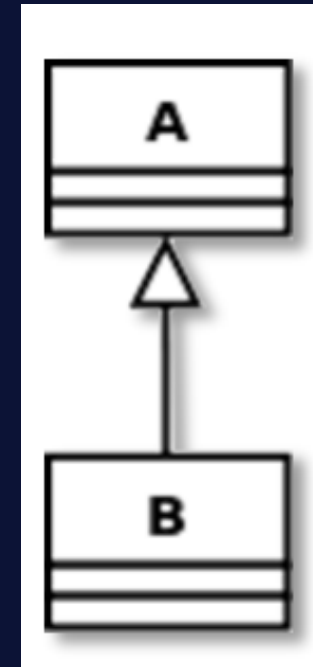


Polymorphisme (1/3)

- ❖ Attribuer à un objet d'une super-classe A une instance de la sous-classe B
- ❖ Cela permet de manipuler des objets sans vraiment connaître leur type
- ❖ Appel des méthodes polymorphiques (les méthodes redéfinis)

```
public class A {  
    public int add(int op1, int op2) { ... }  
}
```

```
public class B extends A {  
    public int subtract(int op1, int op2) { ... }  
}
```



A a = new B(); ok (upcasting)

Polymorphisme (2/3)

- ❖ Attribuer à un objet d'une super-classe A une instance de la sous-classe B
- ❖ Cela permet de manipuler des objets sans vraiment connaître leur type
- ❖ Appel des méthodes polymorphiques (les méthodes redéfinis)

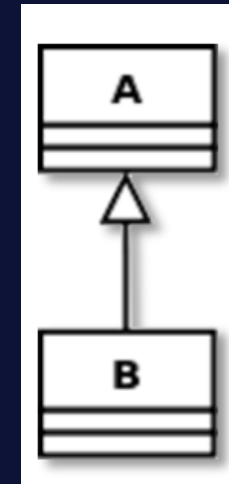
```
public class A {  
    public int add(int op1, int op2) { ... }  
}
```

```
public class B extends A {  
    public int subtract(int op1, int op2) { ... }  
}
```

```
A a = new B();  
int res1 = a.add(2,3);  
int res2 = a.subtract(5,4);
```

ok (upcasting)

Erreur syntaxique

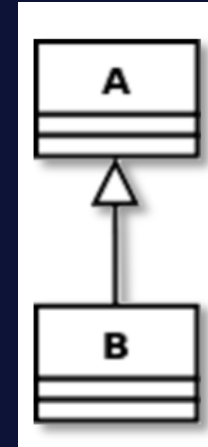


Polymorphisme (3/3)

- ❖ Attribuer à un objet d'une super-classe A une instance de la sous-classe B
- ❖ Cela permet de manipuler des objets sans vraiment connaître leur type
- ❖ Appel des méthodes polymorphiques (les méthodes redéfinis)

```
public class A {  
    public int add(int op1, int op2) { ... }  
}
```

```
public class B extends A {  
    public int subtract(int op1, int op2) { ... }  
}
```



```
A a = new B();  
int res1 = a.add(2,3);  
int res2 = a.subtract(5,4);
```

ok (upcasting)

Erreur syntaxique

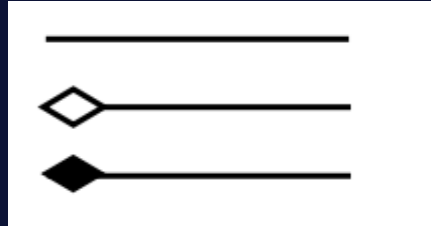
```
if (a instanceof B)  
    int res3 = ((B)a).subtract(5,4);
```

ok (downcasting)

Associations, cardinalités et navigabilité (1/2)

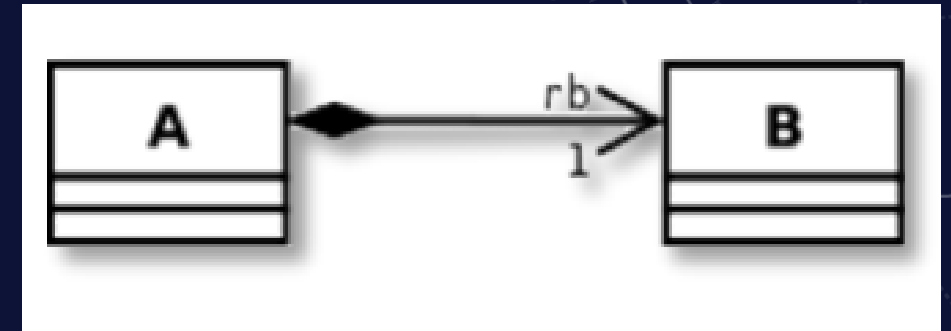
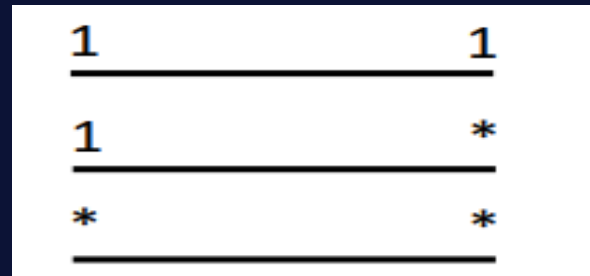
❖ Associations

- ✓ Simple Aggregation
- ✓ Composition



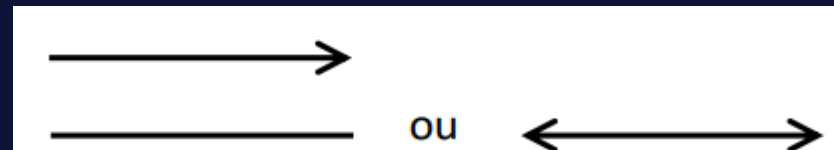
❖ Cardinalités

- ✓ one-to-one
- ✓ one-to-many
- ✓ many-to-many



❖ Navigabilité

- ✓ Unidirectionnelle
- ✓ Bidirectionnelle



Associations, cardinalités et navigabilité (2/2)

```
public class A {  
    private B rb;  
    public void addB(B b) {  
        this.setB(b);  
    }  
    public B getB() { return rb; }  
    public void setB(B b) { this.rb = b; }  
}
```



```
public class B {  
    ...  
    // La classe B ne connaît pas  
    l'existence de la classe A  
}
```

Conclusion

- Les notions de base de la programmation OO ont été révisées avec des exemples de codes en Java
- Ce qui faut retenir c'est la résolution de problèmes sur **un ensemble d'objets collaborant par envoi de messages.**