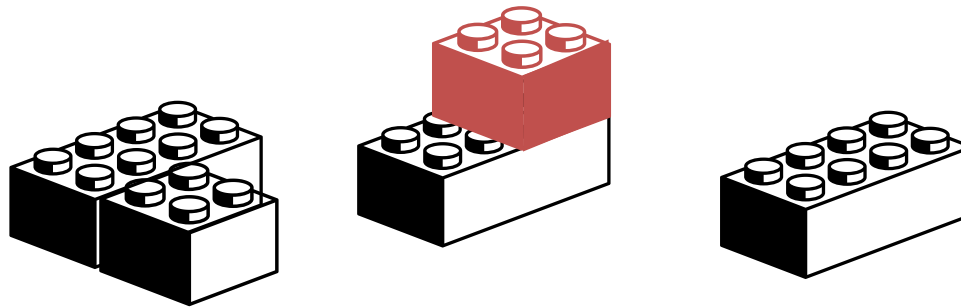


# POO

# Programmation Orientée Objets

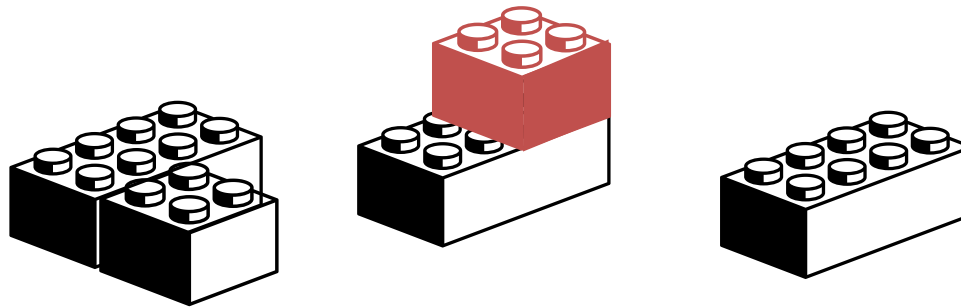
Mise en œuvre par le langage JAVA

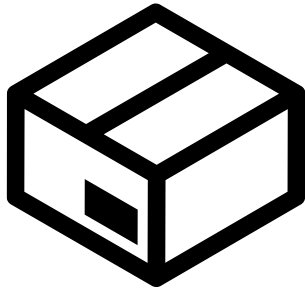
J. Saraydaryan



# Langage Java et Poo

J. Saraydaryan





# Organisation d'un programme JAVA



# Notion de Package

- ❑ Groupe de classes
- ❑ Les packages sont hiérarchisés créant une arborescence
- ❑ Cette arborescence (logique) correspond à arborescence physique des répertoires
- ❑ Classes d'un package doivent inclure la déclaration de ce package afin d'y être inclut

```
package com.course.examples.simplePoo;

class A {
    int a;
    int b;
    static String msg= "MY ONE INSTANCE VARIABLE";
    void A(){}
    void setAB(int a_tmp, int b_tmp){
        a=a_tmp;
        b=b_tmp;
    }
}
```



# Notion de Package

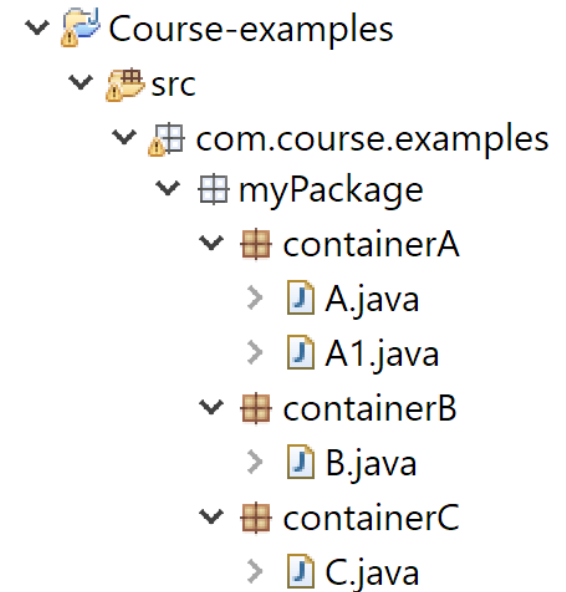
❑ Comment utiliser une classe d'un package ?

```
package com.course.examples.myPackage.containerA;

import com.course.examples.myPackage.containerB.B;

public class A {
    String name;
    int id;
    A1 a1;
    B b;
    com.course.examples.myPackage.containerC.C c;

    public A() {
    }
}
```





# Notion de Package

❑ Comment utiliser une classe d'un package ?

```
package com.course.examples.myPackage.containerA;
```

Déclaration de l'appartenance au package

```
import com.course.examples.myPackage.containerB.B;
```

Importation de l'objet B depuis un autre package

```
public class A {
```

```
    String name;
```

```
    int id;
```

```
    A1 a1;
```

```
    B b;
```

```
    com.course.examples.myPackage.containerC.C c;
```

Class du même package, pas besoin d'import

```
    public A() {
```

```
    }
```

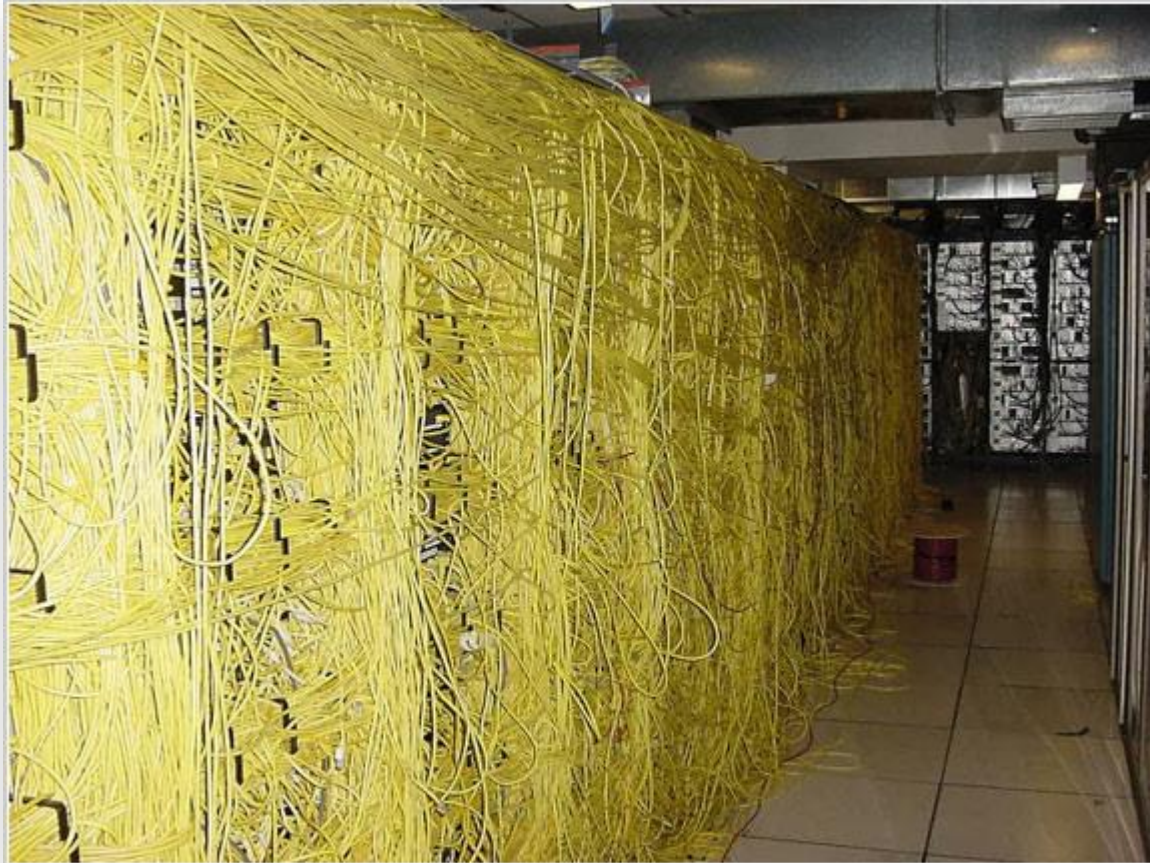
```
}
```

Déclaration de la classe C par l'adresse complète (incluant le nom du package)



# Notion de Package

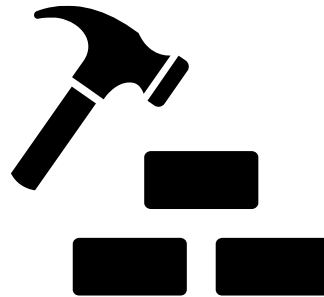
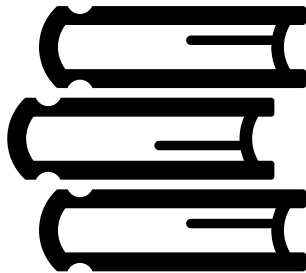
- ❑ Pourquoi utilise-t-on des packages ?





# Notion de Package

- ❑ Pourquoi utilise-t-on des packages ?
  - Meilleur organisation
  - Meilleur réutilisabilité (ensemble d'une package peut être auto-suffisant)
  - Evite les conflits de nommage

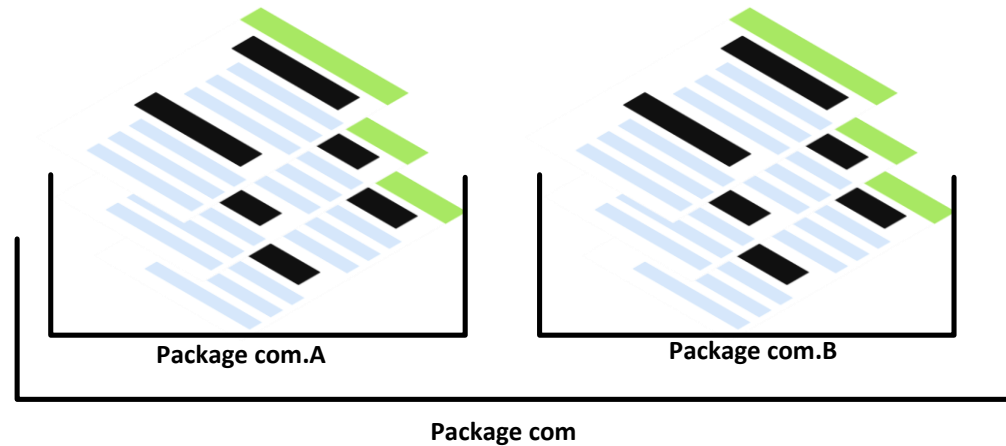
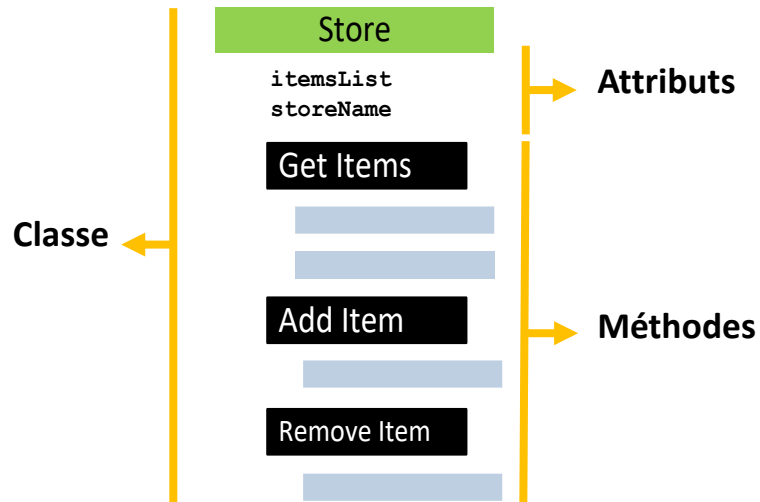






# Notion de Package

- ❑ Organisation d'un programme java en package

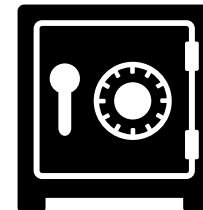




# Encapsulation et autorisation

## ❑ Encapsulation

- Objectif :
  - limiter / contrôler l'accès aux attributs d'une classe
  - Contrôle des modifications des attributs
  - Masquer les attributs aux personnes (objets) non autorisés





# Encapsulation et autorisation

## ☐ Niveau d'accès

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗



# Encapsulation et autorisation

## □ Niveau d'accès

```
package
com.course.examples.myPackage.crt.containerA;

public class A {
    private String secret;
    protected String name;
    int id;
    public String helloMessage;

    public A() {}

    private void myProcess(){}

    public String getSecret() {
        return secret;
    }
    public void setSecret(String secret) {
        this.secret = secret;
    }
}
```

Déclaration de  
l'appartenance au package

Définition de la portée des  
attributs

Définition de la portée des  
méthodes

Définition des accesseurs de  
l'attribut secret



# Encapsulation et autorisation

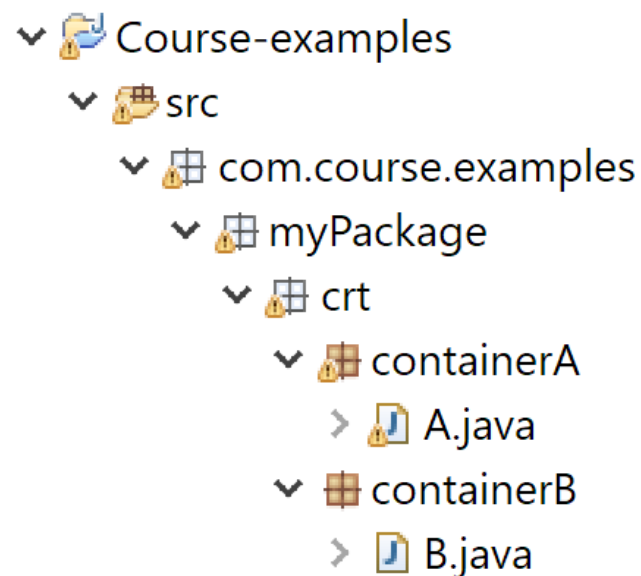
## ❑ Niveau d'accès

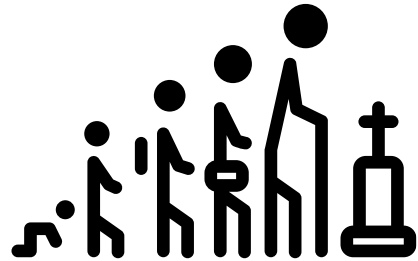
```
package
com.course.examples.myPackage.crt.containerB;

import
com.course.examples.myPackage.crt.containerA.
A;

public class B {
    A a;

    public B() {
        a = new A();
        a.secret = "my value";
        a.setSecret("my value");
        a.helloMessage = "HELLO";
        a.name = "testA";
    }
}
```





## **Cycle de vie des Objets JAVA**

# Notion d'objet et d'instance

## Class



## Instance



# Notion d'objet et d'instance

## ❑ Création d'instance

- Appel du constructeur de la classe
- Propriété
  - Toujours public
  - Même nom que la classe
  - Ne retourne rien (pas de return)
  - Pas de void nécessaire
  - Plusieurs constructeurs possibles (surcharge)

```
package com.course.examples.misc;

public class AObject {
    private String name;
    private int id;

    public AObject(String name,int id) {
        this.name=name;
        this.id=id;
    }

    public AObject() {
        this.name="MyName1";
        this.id=0;
    }
    ...
}
```



# Notion d'objet et d'instance

## ❑ Création d'instance

```
package com.course.examples.misc;

public class AObject {
    private String name;
    private int id;


    public AObject(String name,int id) {
        this.name=name;
        this.id=id;
    }

    public AObject() {
        this.name="MyName1";
        this.id=0;
    }
    ...
}
```

```
AObject a1;
    a1=new AObject("AName", 100);

AObject a2;
    a1=new AObject();
```

# Notion d'objet et d'instance




```
package com.course.examples.misc;

public class AObject {
    private String name;
    private int id;

    public AObject(String name,int id) {
        this.name=name;
        this.id=id;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```



```
package com.course.examples.misc;

public class BObject {
    private AObject a;

    public BObject() {

    }

    public void createA(String name, int id){
        this.a=new AObject(name, id);
    }
}
```

# Notion d'objet et d'instance

```
public static void main(String[] args) {  
    BObject b;  
  
    b=new BObject();  
  
    b.createA("MyName1", 400);  
  
    AObject a2;  
  
    a2=new AObject("MyName2", 100);  
}
```



Déclaration de la classe BObject



Instance BObject



Instance AObject (cf Bobject)



Déclaration de la classe AObject



Instance AObject (cf Bobject)



AObject



BObject



b



b.a



a2

# Notion d'objet et d'instance

## ❑ Destruction d'instance (1/2)

- Pas de gestion directe de la libération des ressources (Garbage collector)
- Toute la mémoire d'objets non référencés (pas de référence pour y accéder) est susceptible d'être libérée par le Garbage Collector

```
AObject a1;  
a1=new AObject("AName", 100);  
a1=null;
```

# Notion d'objet et d'instance

## ❑ Destruction d'instance (2/2)

- Lors de la libération des ressources (Garabage collector) la méthode **finalize** d'un objet est appelée
- Permet de nettoyer proprement les éléments gérés par la classe courante (fermeture de flux fichier, réseaux, destruction de Thread...)

```
public class Fichier {  
    FileInputStream f;  
    File f=new File("AName");  
    ...  
    public void close() {  
        if (f != null) {  
            f.close();  
            f = null;  
        }  
    }  
}  
  
protected void finalize() throws Throwable {  
    super.finalize();  
    close();  
}
```

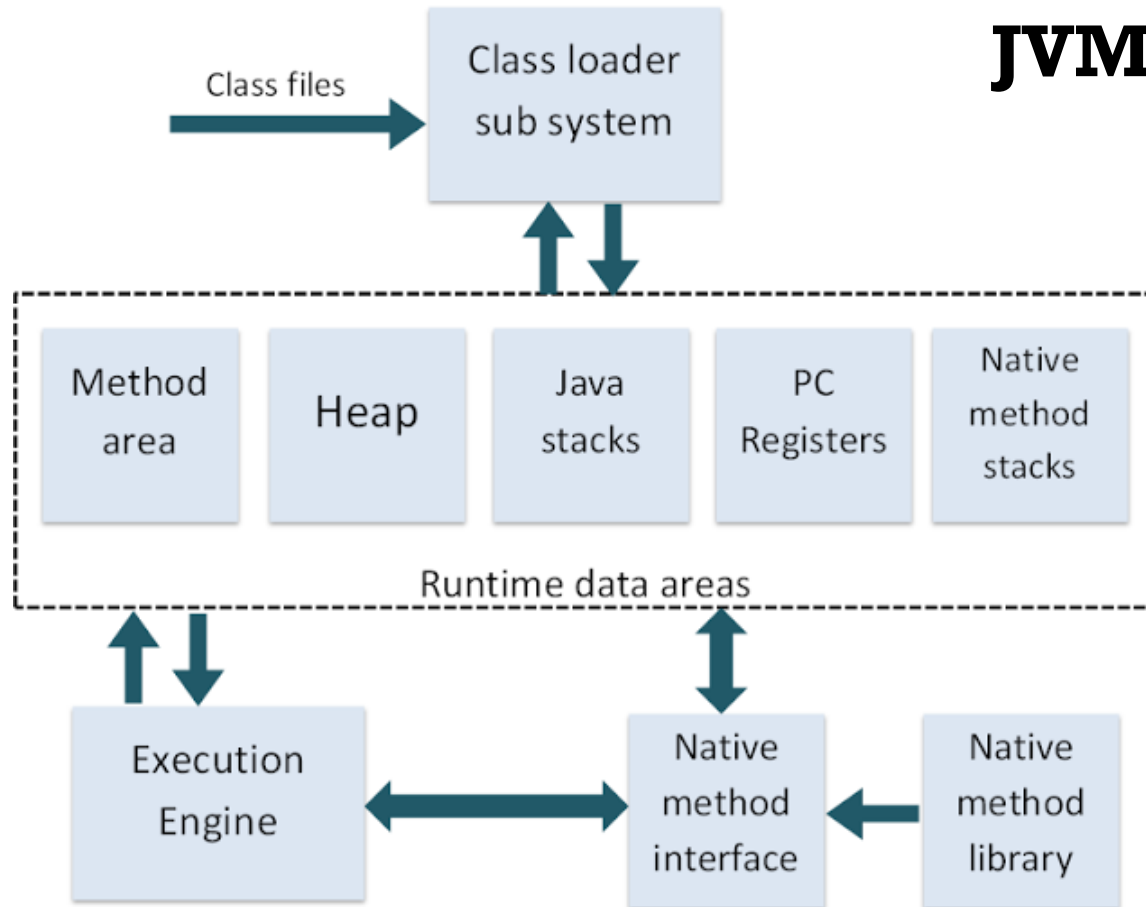
# Notion d'objet et d'instance

## ❑ Initialisation par default

- Tous les attributs d'un objet sont initialisés par default:
  - **0** pour tous les nombres (int, float ...)
  - **Null** pour toutes les références à un objet
- Possibilité de déclarer la valeur par défaut d'une attribut lors de sa déclaration

```
private String name="DefaultName" ;  
private int id=0;
```

# Garbage Collector (ramasse miettes)

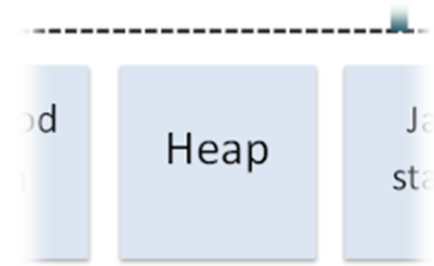


<https://www.quora.com/What-is-the-architecture-of-JVM-and-responsibility-of-each-component-in-JVM-Java-virtual-machine>

# Garbage Collector (ramasse miettes)

## ☐ Usage

- Vérification du Heap Memory ->  
Identification des objets utilisés ou non
- Suppression de objets non utilisés (libération des ressources)
- Objet non utilisé = aucun élément de notre programme n'a de pointeur vers cet objet

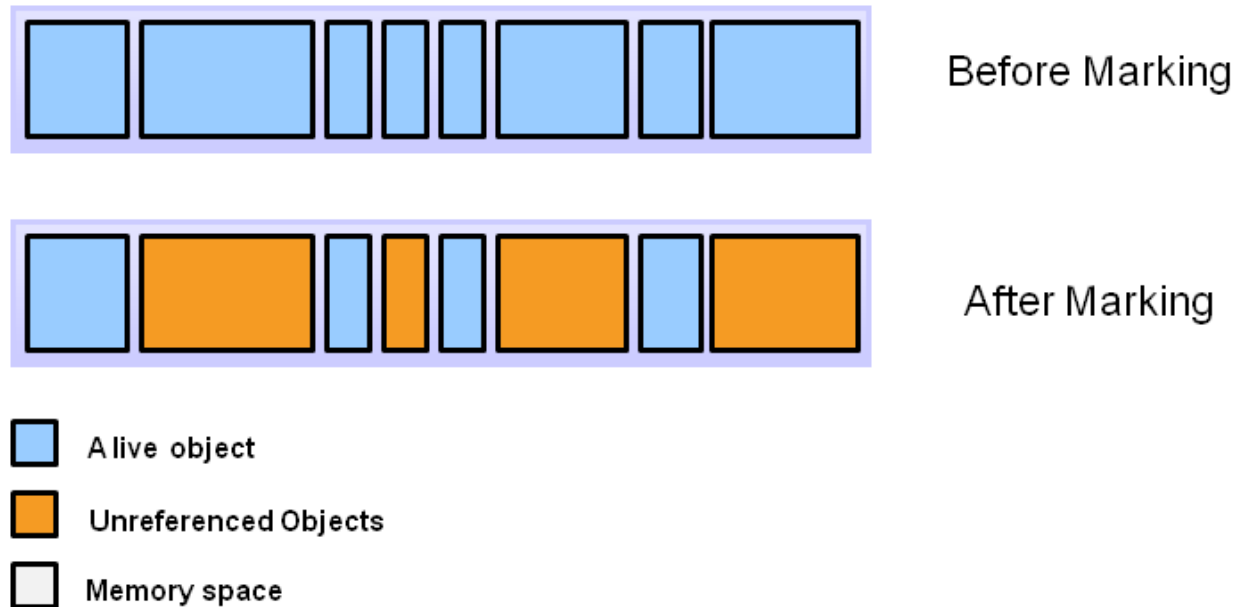


- **Heap:** Zone mémoire de création des objets



# Garbage Collector (ramasse miettes)

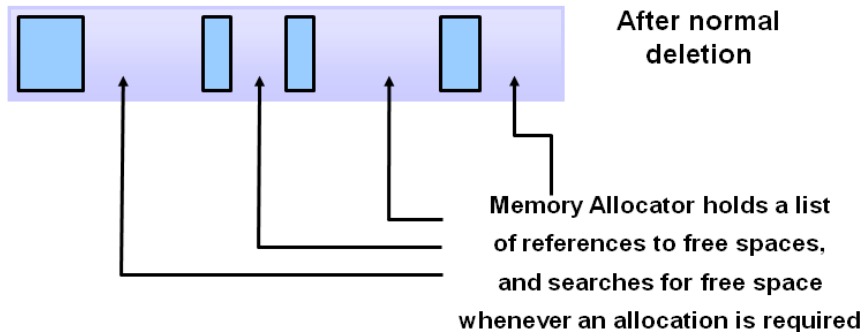
## Marking



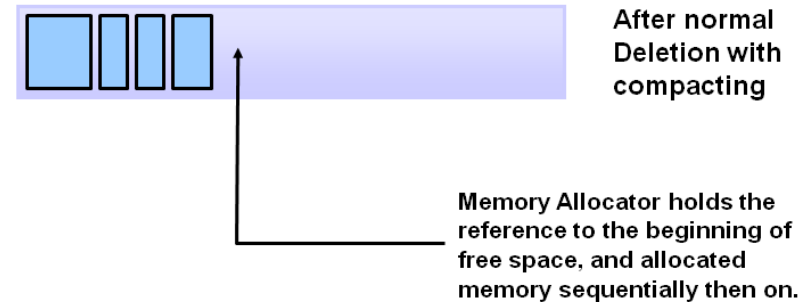
<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

# Garbage Collector (ramasse miettes)

## Normal Deletion



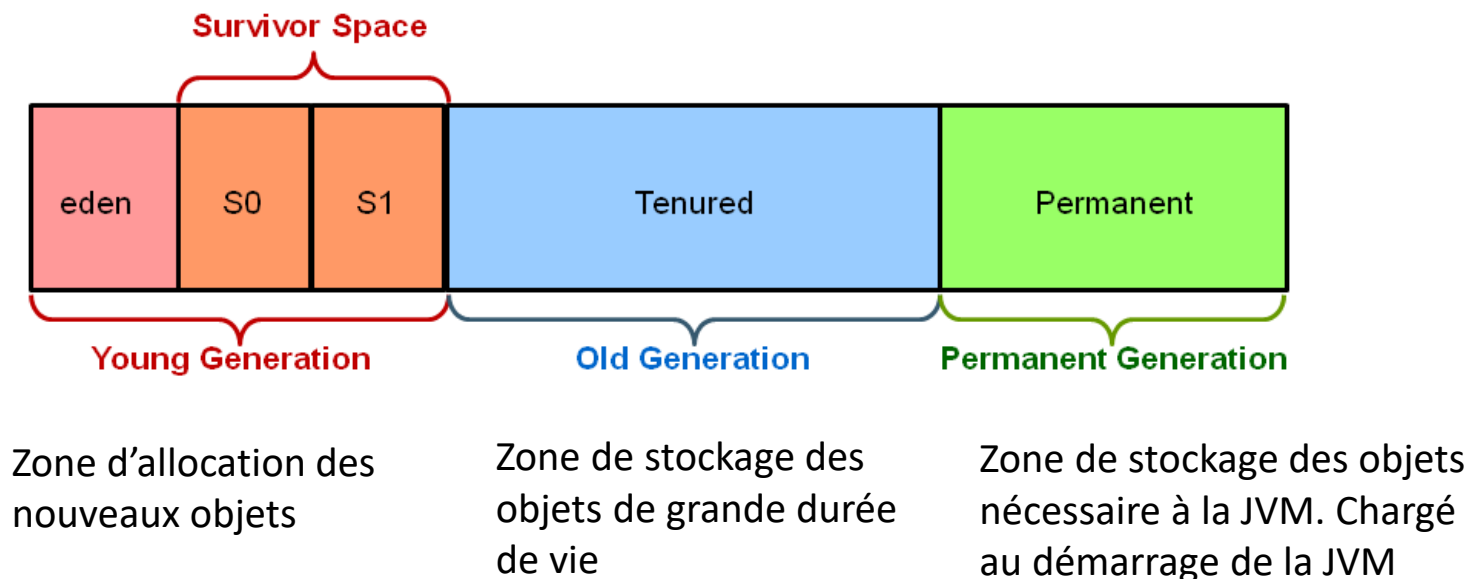
## Deletion with Compacting



<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

# Garbage Collector (ramasse miettes)

## ❑ JVM génération



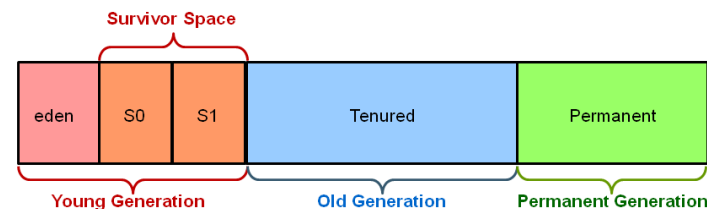
<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

# Garbage Collector (ramasse miettes)

## ❑ Minor/Major Garbage collection

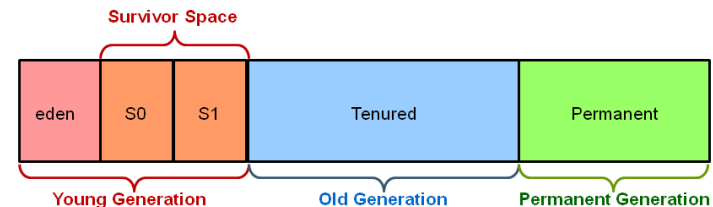
## ❑ Minor garbage collection

- Déclencher lorsque la zone « **Young Generation** » est remplie (impossible d'ajouter de nouveaux objets)
- Libérations des ressources associées
- Collecte des nouveaux objets « **Young Generation** » non utilisés
- Déplacement de certains objets dans la zone « **Old Generation** »
- Suspension de l'ensemble des evts de l'application (**Stop the World**)
  - tous les threads sont suspendus jusqu'à la fin de l'opération



# Garbage Collector (ramasse miettes)

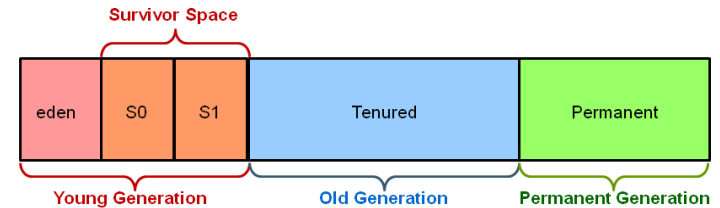
- ❑ Minor/Major Garbage collection
- ❑ Major garbage collection
  - Collecte des nouveaux objets « **Old Generation** » non utilisés
  - Libérations des ressources associées
  - Suspension de l'ensemble des evts de l'application (**Stop the World**)
    - tous les threads sont suspendus jusqu'à la fin de l'opération



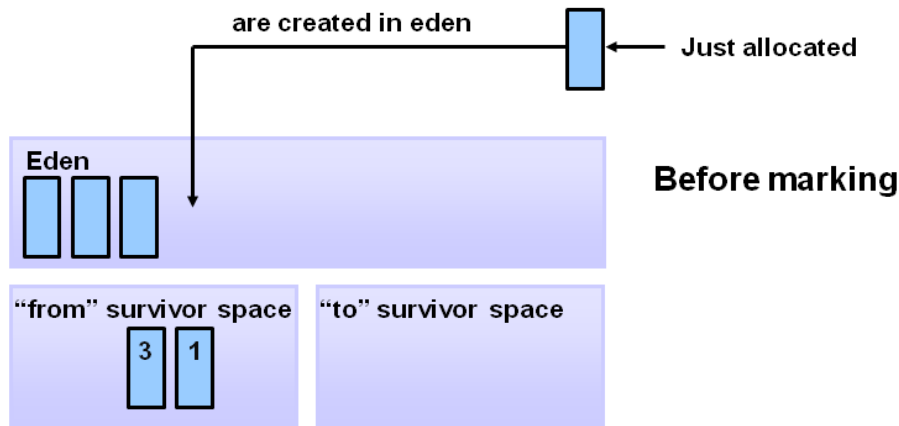
<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

# Garbage Collector (ramasse miettes)

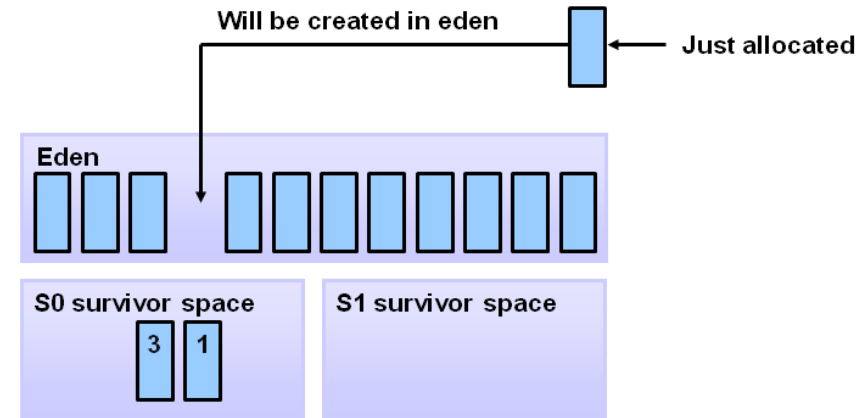
## Minor Garbage collection



### Object Allocation



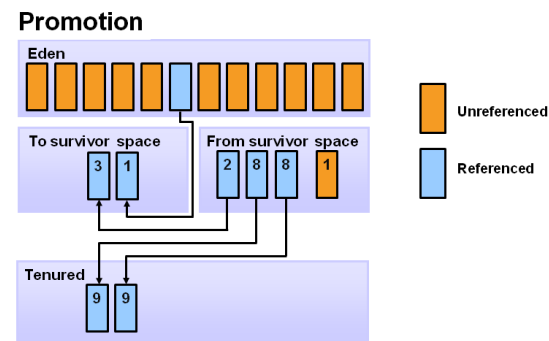
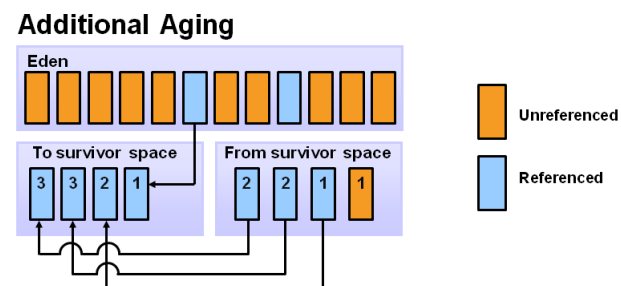
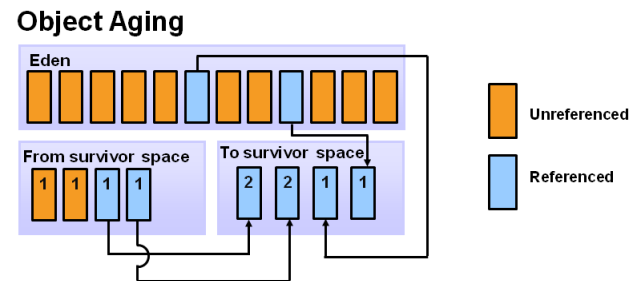
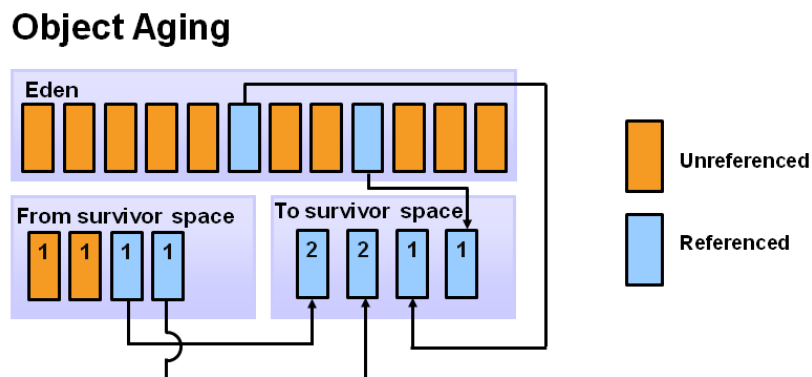
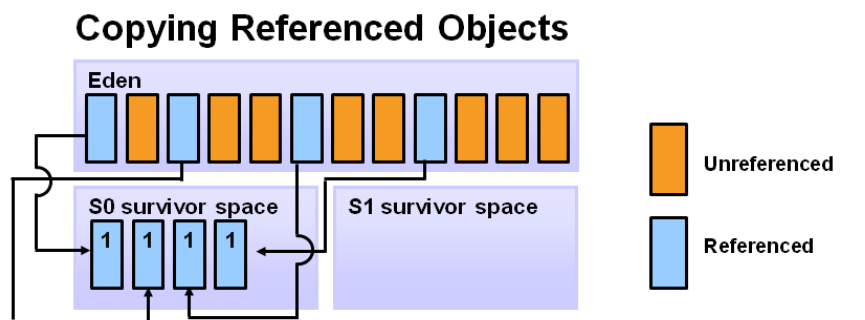
### Filling the Eden Space



<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

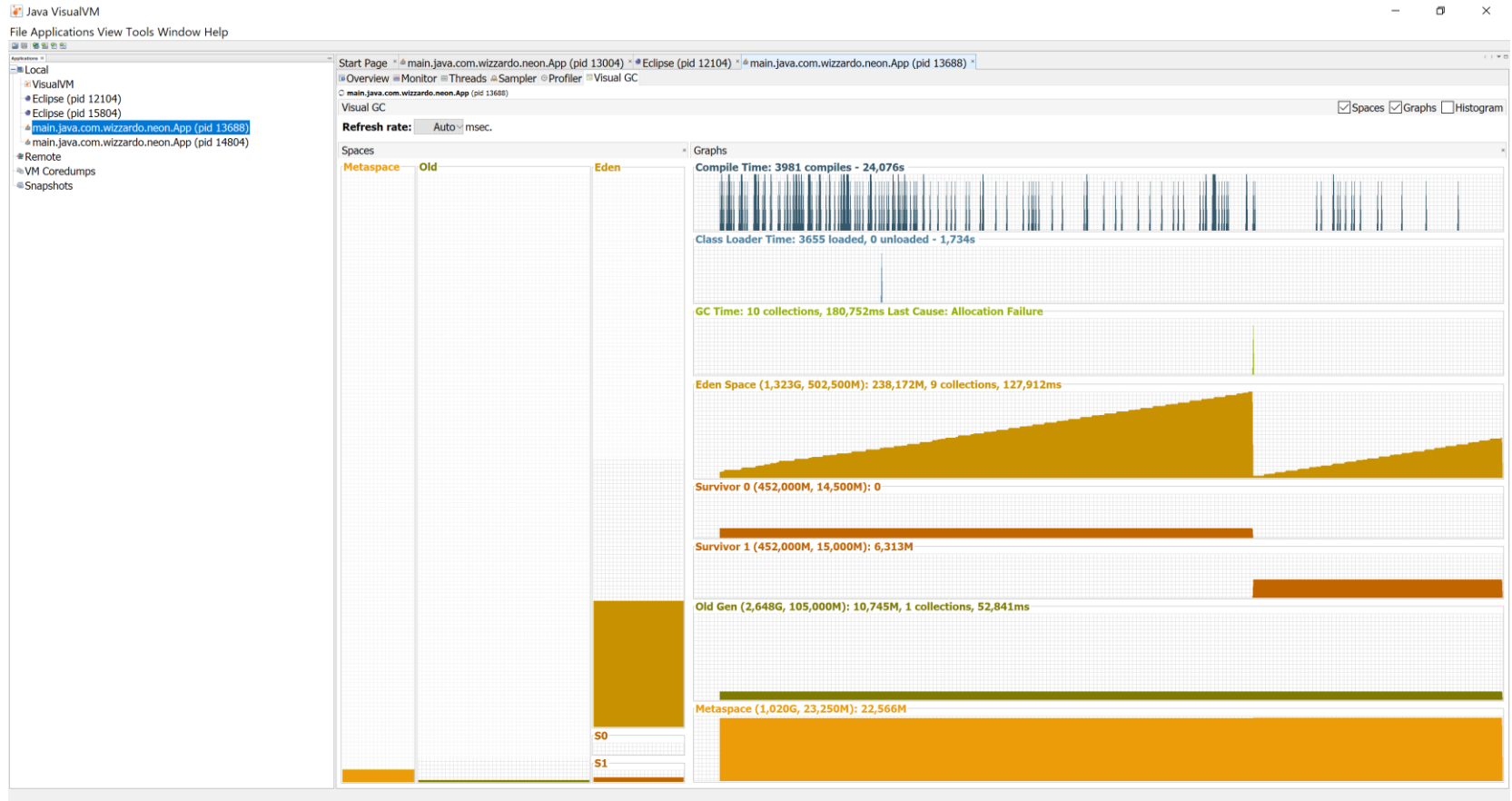
# Garbage Collector (ramasse miettes)

## Minor Garbage collection



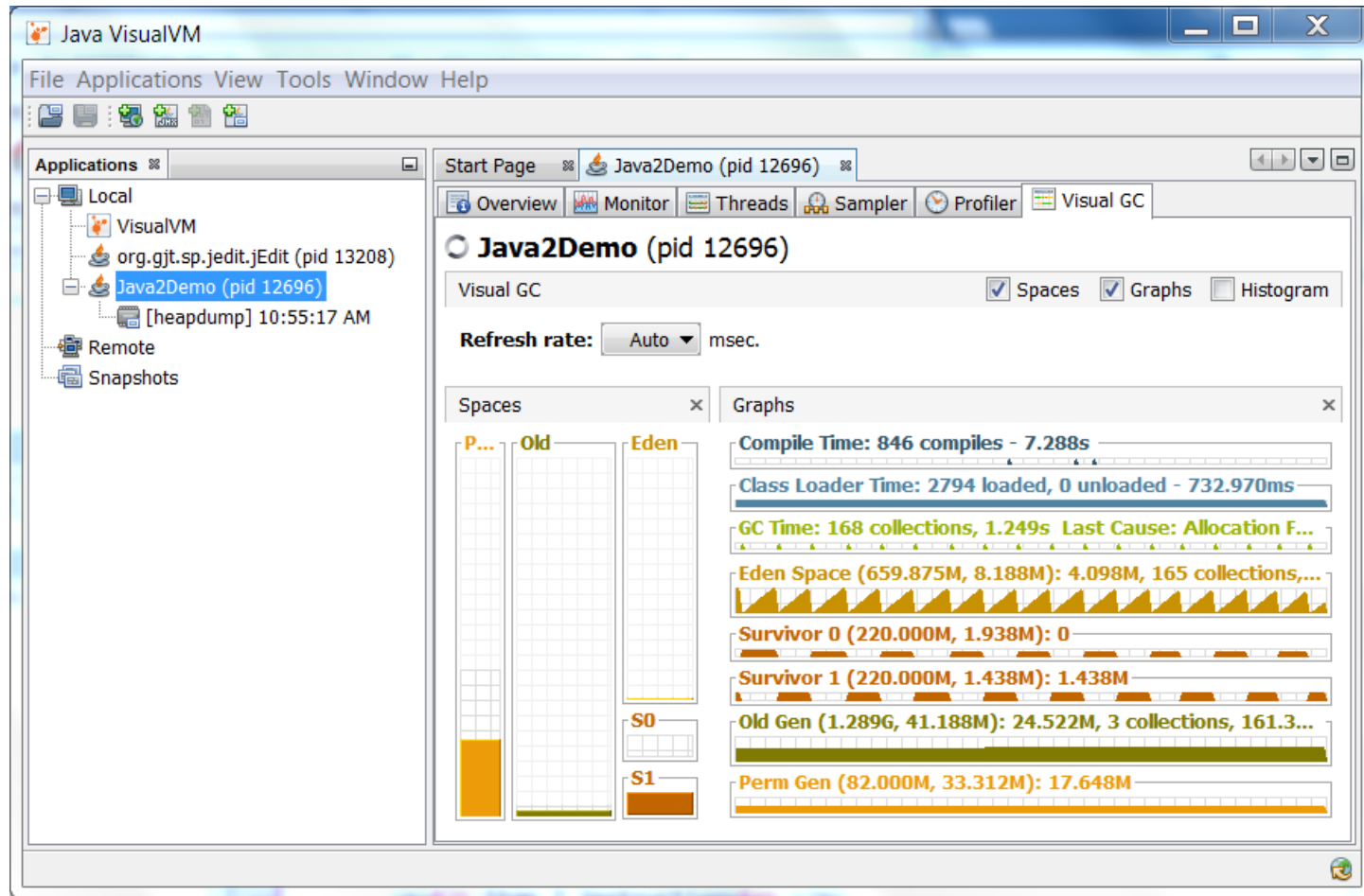
# Garbage Collector (ramasse miettes)

Jvisualvm + VisualGC plugin





# Garbage Collector (ramasse miettes)



<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

# Class Members

- ❑ Attributs / méthodes appartenant à la classe et non à l'instance
- ❑ Usage du mot clé « Static »
- ❑ Permet de créer un attribut une méthode commune à toute les instances de la classe



Static String form= "CookiesMan "



Name -> "A"  
Id -> 100



Name -> "A2"  
Id -> 400



Name -> "A3"  
Id -> 500

# Class Members

```
package com.course.examples.misc;
```

```
public class AClassMember {  
    private String name;  
    private int id;
```

```
    public static String form="CookieMan";
```

Déclaration d'un attribut de class

```
    public AClassMember(String name,int id) {  
        this.name=name;  
        this.id=id;  
    }
```

```
    public static int addition(int a, int b){  
        return a+b;  
    }
```

Déclaration d'une méthode

```
}
```

# Class Members

```
package com.course.examples.misc;

public class AClassMember {
    private String name;
    private int id;

    public static String form="CookieMan";

    public AClassMember(String name,int id) {
        this.name=name;
        this.id=id;
    }

    public static int addition(int a, int b){
        return a+b;
    }
}
```

```
public static void main(String[] args) {
    AClassMember a1,a2;
    a1=new AClassMember("A1", 10);
    a2=new AClassMember("A2", 20);
```

```
    System.out.println(AClassMember.form);
```

```
    System.out.println(a1.form);
```

```
    System.out.println(a2.form);
```

```
    int result=AClassMember.addition(45,10);
```

```
    int result2=a2.addition(4, 5);
```

```
}
```

# Exercice

Créer un programme respectant la hiérarchie suivante:

```

v pkg
  v manager
    > ConnectionMng.java
    > UserMng.java
  v model.user
    > UserLoginModel.java
    > UserModel.java
  > Launch.java
  
```

Réalisation les classes suivantes:

UserModel

- surname
- lastname
- login
- Pwd
- checkPwd(pwd)

UserLoginModel

- login
- pwd

UserMng

- Tab de UserModel
- checkUser (login,pwd)

ConnectionMng

- UserMng
- connection(login,pwd)

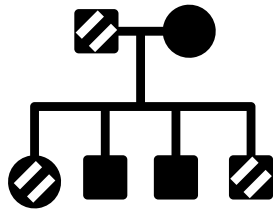


Attribuer les autorisations sur les attributs, accesseurs et les méthodes sachant que:

- Launch possède ConnectionMng, et UserMng, mais n'a accès qu'à connection(login,pwd) de ConnectionMng
- UserMng n'a accès qu'à checkPwd(pwd) de la classe UserModel
- ConnectionMng n'a accès qu'à checkUser(login,pwd) de UserMng
- Il est possible de changer surname et lastname après la création de UserModel

Réaliser le comportement de Launch

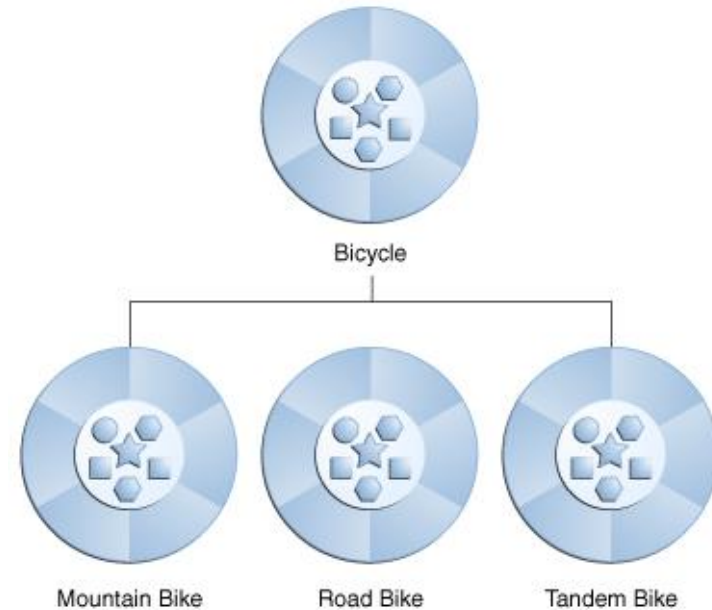
- Création d'un Tab d'UserModel (5 users)
- Création de ConnectionMng et UserMng (passage au constructeur du Tab d'UserModel)
- Vérification de connexion à l'aide de connection(login,pwd) de ConnectionMng d'utilisateurs légitimes et non-légitimes



# Héritage en JAVA

# Héritage

- ❑ Mettre en évidence des points communs entre plusieurs objets
  - Partage de caractéristiques → **attributs**
  - Partage de comportements → **méthodes**
- ❑ L'héritage de Java permet à chaque classe d'hériter des caractéristiques et des comportement d'une autre classe (1 seule)
- ❑ Cette classe parente sera appelée *superclass*
- ❑ Le mot clé *extend* permettra de réaliser l'héritage



<https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html>

# Héritage

```
public class Vehicule {
    private float speed_x, speed_y;
    private float coord_x, coord_y;
    private int nbrPlace;
    private int nbRoues;

    public Vehicule() {
        ...
    }
    public void accelerer(float x, float y){
        ...
    }
    public void freiner(float x, float y){
        ...
    }
    public float getSpeed_x() {
        return speed_x;
    }
    public void setSpeed_x(float speed_x) {
        this.speed_x = speed_x;
    }
    ...
}
```

```
public class Voiture extends Vehicule{
    public Voiture() {
        super();
    }
    ...
}
```

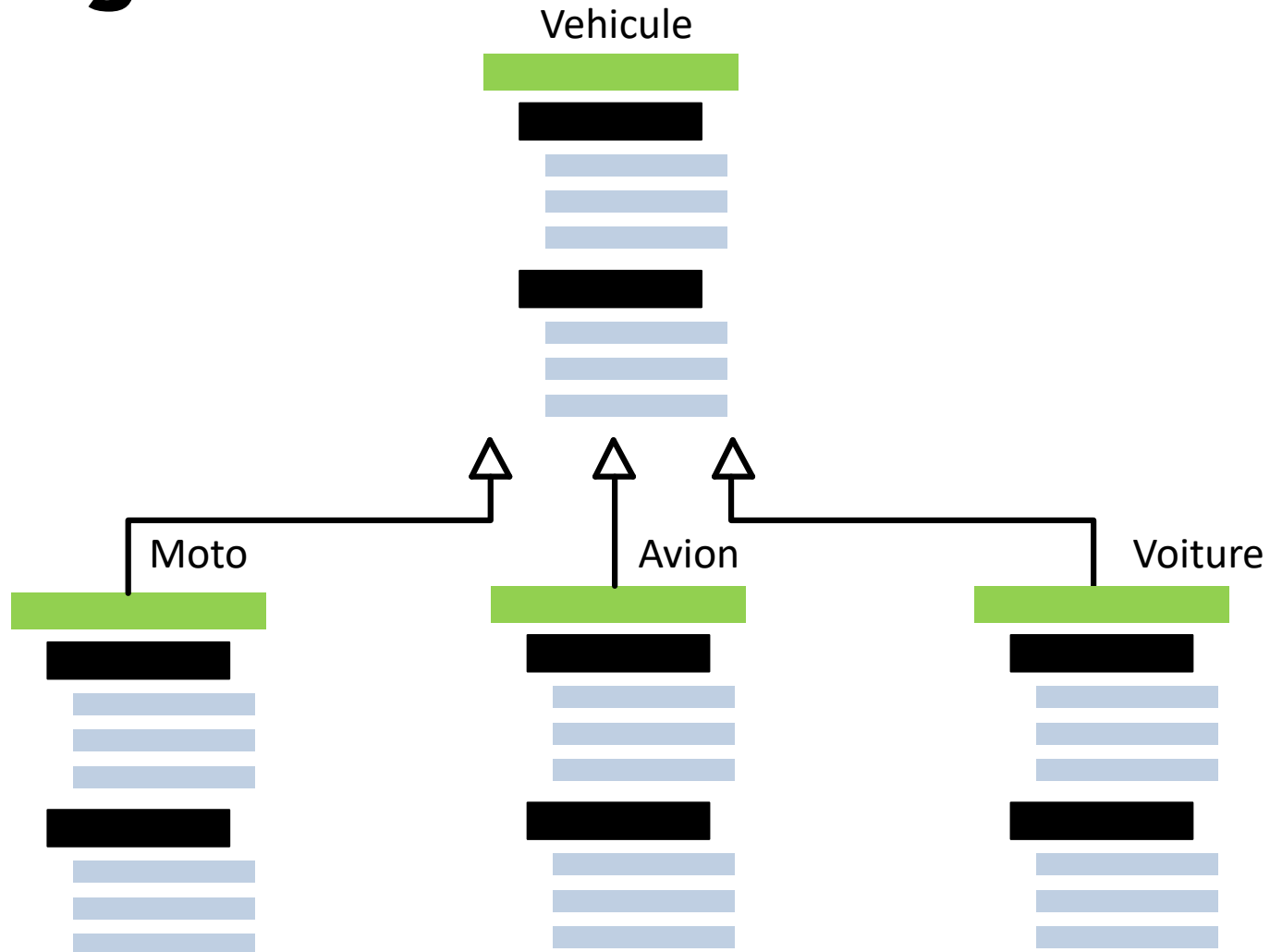
```
public class Moto extends Vehicule{

    public Moto() {
        ...
    }
    ...
}
```

```
public class Avion extends Vehicule {
    public Avion() {
        ...
    }
    ...
}
```



# Héritage



# Héritage

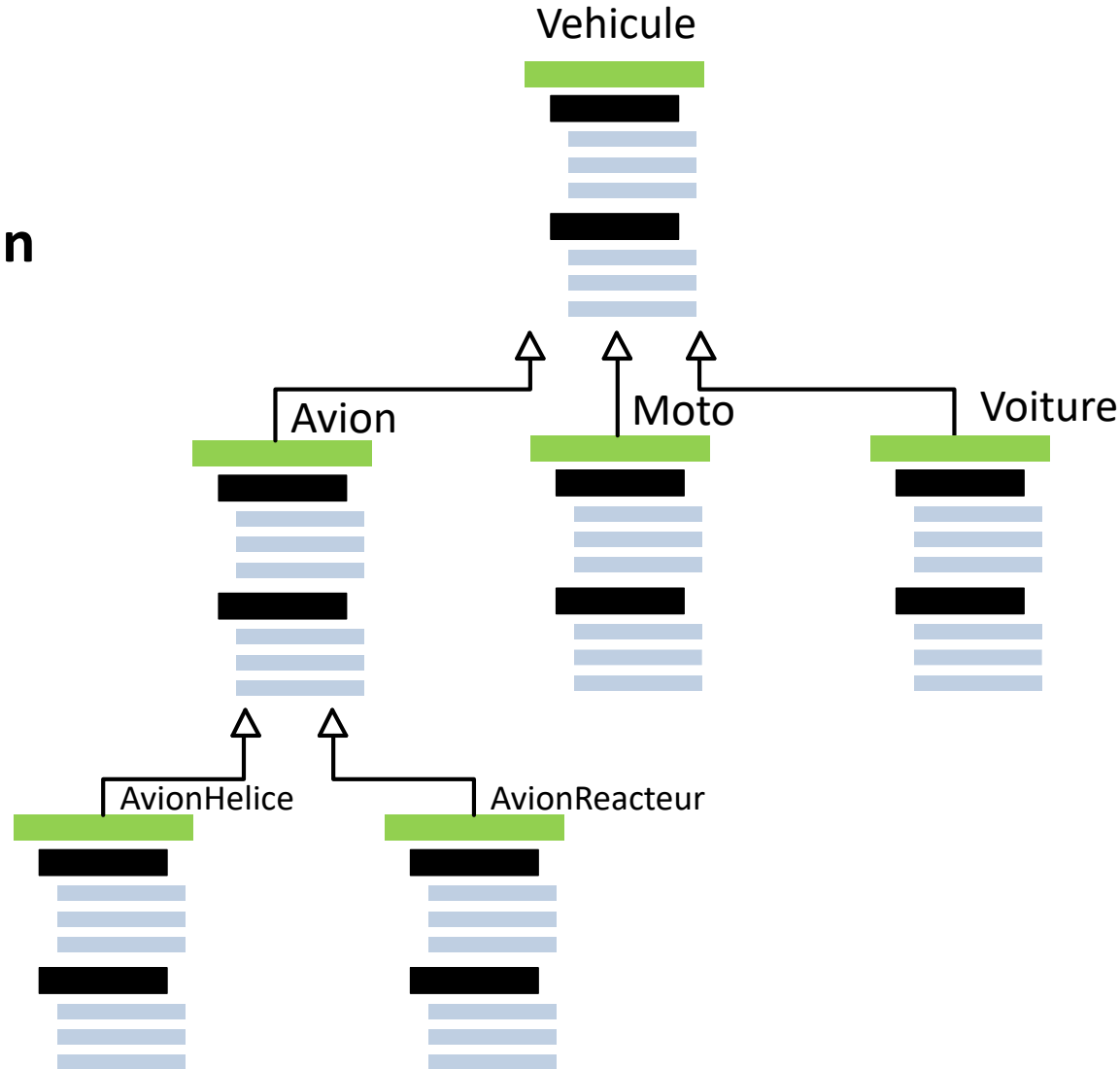


Généralisation

→ Abstraction

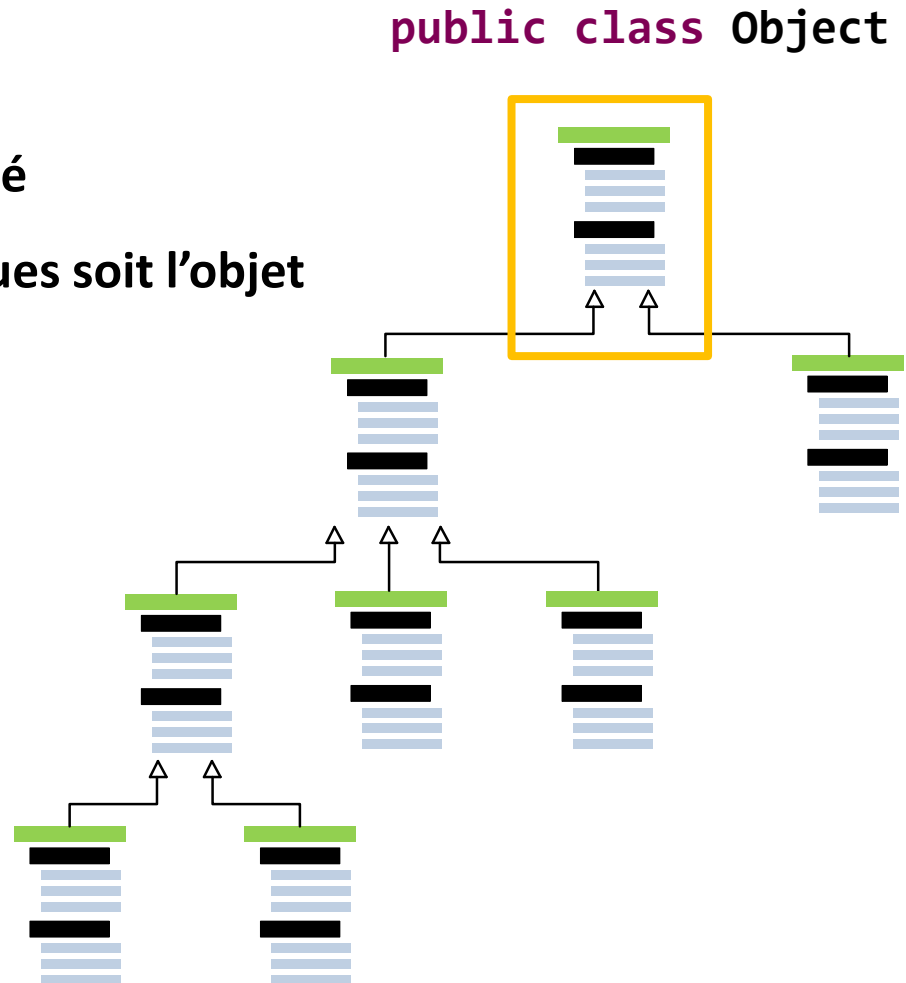
Spécialisation

→ Monde réel



# Héritage

- ❑ Toutes les classes héritent d'une même super class *Object*
- ❑ Ce process permet la généricité
- Effectuer un traitement quelques soit l'objet



# Héritage

## ❑ Méthodes principales héritées de Object

- **equals**

→ permet de définir l'égalité entre deux *Object*

- **hashCode**

→ retourne un identifiant de l'objet (int)

- **toString**

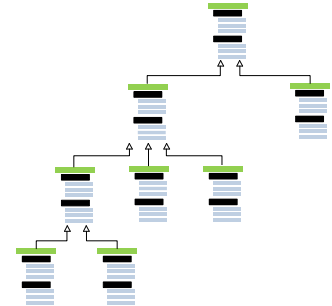
→ retourne une chaine de caractères représentant l'objet

- **Finalize**

→ appelé par le Garbage Collector

- **Clone**

→ créer et retourne une copie de l'objet



# Héritage

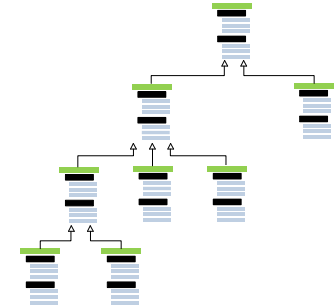
## Methods

Modifier and Type	Method and Description
protected <b>Object</b>	<b>clone()</b> Creates and returns a copy of this object.
boolean	<b>equals(Object obj)</b> Indicates whether some other object is "equal to" this one.
protected void	<b>finalize()</b> Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
<b>Class&lt;?&gt;</b>	<b>getClass()</b> Returns the runtime class of this <b>Object</b> .
int	<b>hashCode()</b> Returns a hash code value for the object.
void	<b>notify()</b> Wakes up a single thread that is waiting on this object's monitor.
void	<b>notifyAll()</b> Wakes up all threads that are waiting on this object's monitor.
<b>String</b>	<b>toString()</b> Returns a string representation of the object.
void	<b>wait()</b> Causes the current thread to wait until another thread invokes the <b>notify()</b> method or the <b>notifyAll()</b> method for this object.
void	<b>wait(long timeout)</b> Causes the current thread to wait until either another thread invokes the <b>notify()</b> method or the <b>notifyAll()</b> method for this object, or a specified amount of time has elapsed.
void	<b>wait(long timeout, int nanos)</b> Causes the current thread to wait until another thread invokes the <b>notify()</b> method or the <b>notifyAll()</b> method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

# Héritage

## ❑ Appel/Surcharge des méthodes héritées

- Mot clé **super** permet d'accéder aux méthodes parentes
- Annotation **@Override** permet de surcharger une méthode héritée



**Annotations:** metadata qui fournissent des informations sur le programme sans avoir d'effet direct sur ce dernier

## Usage des annotations possibles:

- **Information pour le compilateur:** détection d'erreurs, suppression de warning)
- **Compile-time et deployment-time processing :** des outils peuvent traiter ces informations pour générer du code, des fichiers, etc..
- **Runtime processing:** disponible et examiné au runtime

# Héritage

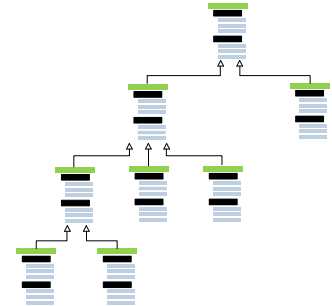
## ❑ Surcharge des méthodes héritées

```
public class Voiture extends Vehicule{
    public Voiture() {
        // appel du constructeur parent
        super();
    }

    //Surcharge de la méthode parente directe
    @Override
    public void setCoord_x(float coord_x) {
        coord_x=coord_x+0.5f;
        super.setCoord_x(coord_x);
    }

    //Surcharge de la méthode parente indirecte (Object)
    @Override
    public String toString() {
        String display="";
        // appelle des méthodes héritées
        float x= getCoord_x();
        float y= getCoord_y();

        display="Voiture["+x+", "+y+"]";
        return display;
    }
}
```



# Héritage

## ☐ Accès aux éléments hérités

Modifieur	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗



# Héritage

## ❑ Accès aux éléments hérités

```
public class Vehicule {
```

```
    private float speed_x, speed_y;  
    private float coord_x, coord_y;
```

```
    private int nbrPlace;  
    private int nbRoues;
```

```
    protected String licensePlate;  
    public String name;
```

```
    public Vehicule() {  
        ...  
    }
```

```
    public void accelerer(float x, float y){  
    }  
    public void freiner(float x, float y){  
        ...  
    }
```

```
    private void generateLicensePlate(){  
        ...  
    }
```

```
public class Voiture extends Vehicule{
```



...



# Héritage

## ❑ Connaitre l'objet spécialisé

```
public class CkcheckInstance {  
    public static void main(String[] args) {  
        Moto m1=new Moto();  
        Voiture v1=new Voiture();  
        Voiture v2=new Voiture();  
        Vehicule[] vehiculeList=new Vehicule[3];  
        vehiculeList[0]=m1;  
        vehiculeList[1]=v1;  
        vehiculeList[2]=v2;  
  
        for(int i=0;i<vehiculeList.length;i++){  
            if(vehiculeList[i] instanceof Moto){  
                Moto tmpM=(Moto)vehiculeList[i];  
                System.out.println("Elt at "+i+" is a Moto, "+tmpM.toString());  
            }  
  
            if(vehiculeList[i] instanceof Voiture){  
                Voiture tmpV=(Voiture)vehiculeList[i];  
                System.out.println("Elt at "+i+" is a Voiture, "+tmpV.toString());  
            }  
        }  
    }  
}
```

# Exercice

Créer une Classe UserModel

- SurName
- LastName
- Login
- Pwd

Redéfinissez la classe toString() afin d'afficher tous les paramètres de la classe sauf pwd (texte remplacé par des \*)

Redéfinissez la classe equal() afin retourner vrai si deux utilisateurs possèdent le même login et le même lastName



# Exercise

```
----- jdoe -----  
- surname: john  
- lastName: Doe  
- login: jdoe  
- pwd: *****
```

```
----- tsm -----  
- surname: ted  
- lastName: Smith  
- login: tsm  
- pwd: *****
```

SAME OBJ:

```
----- jdoe -----  
- surname: john  
- lastName: Doe  
- login: jdoe  
- pwd: *****
```

SAME OBJ:

```
----- jdoe -----  
- surname: eric  
- lastName: Doe  
- login: jdoe  
- pwd: *****
```



# Exercice

Modéliser un **Shop** possédant des **Items**.

Chaque **Item** possède:

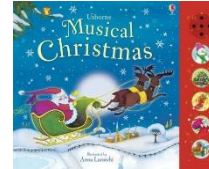
- Name, Id, price, nbrElt
- sell(), isInStorage()
- Il existe des catégories d'**Items** :
  - Fourniture (**Consumable**) (Quantity)  
2 types de **Consumable** existent:
    - **Paper** (quality, weight)
    - **Pen** (color,durability)
  - **Book** (nb of pages,author, publisher,year,age)  
4 Types de **Book**
    - **BookToTouch** (Material, durability)
    - **MusicalBook** (list of sound, lifetime, nbre of battery)
    - **PuzzleBook**(nbre of pieces)
    - **OriginalBook**(isNumeric)



Le **Shop** possède une liste d'items et les fonctionnalités suivantes:

- isItemAvailable(), getAgeForBook(), addItem()

# Exercice



Item[] itemTab

- isItemAvailable(),
- getAgeForBook()

# Exercice

Créer une Classe A (name)

- getName()

Créer une Classe B (price) qui hérite de A

- getPrice()

Créer une Classe C (address) qui hérite de A

- getAddress()

Créer une classe Launch qui permet  
De créer un tableau de A (3)

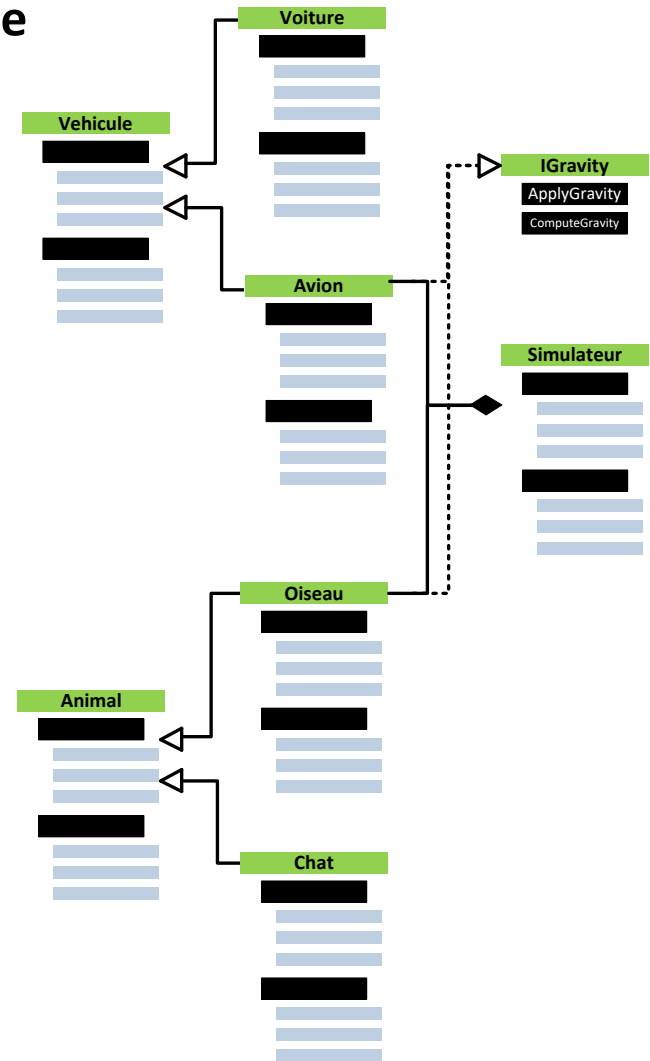
De parcourir le tableau et de :

- getPrice si objet est de type B
- getAddress si objet est de type C



# Interface

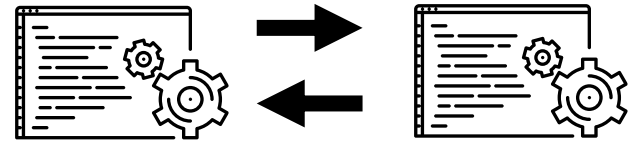
- ❑ Héritage en java → 1 seule Classe Parente
- ❑ Comment spécifier un comportement commun à des objets différents ?



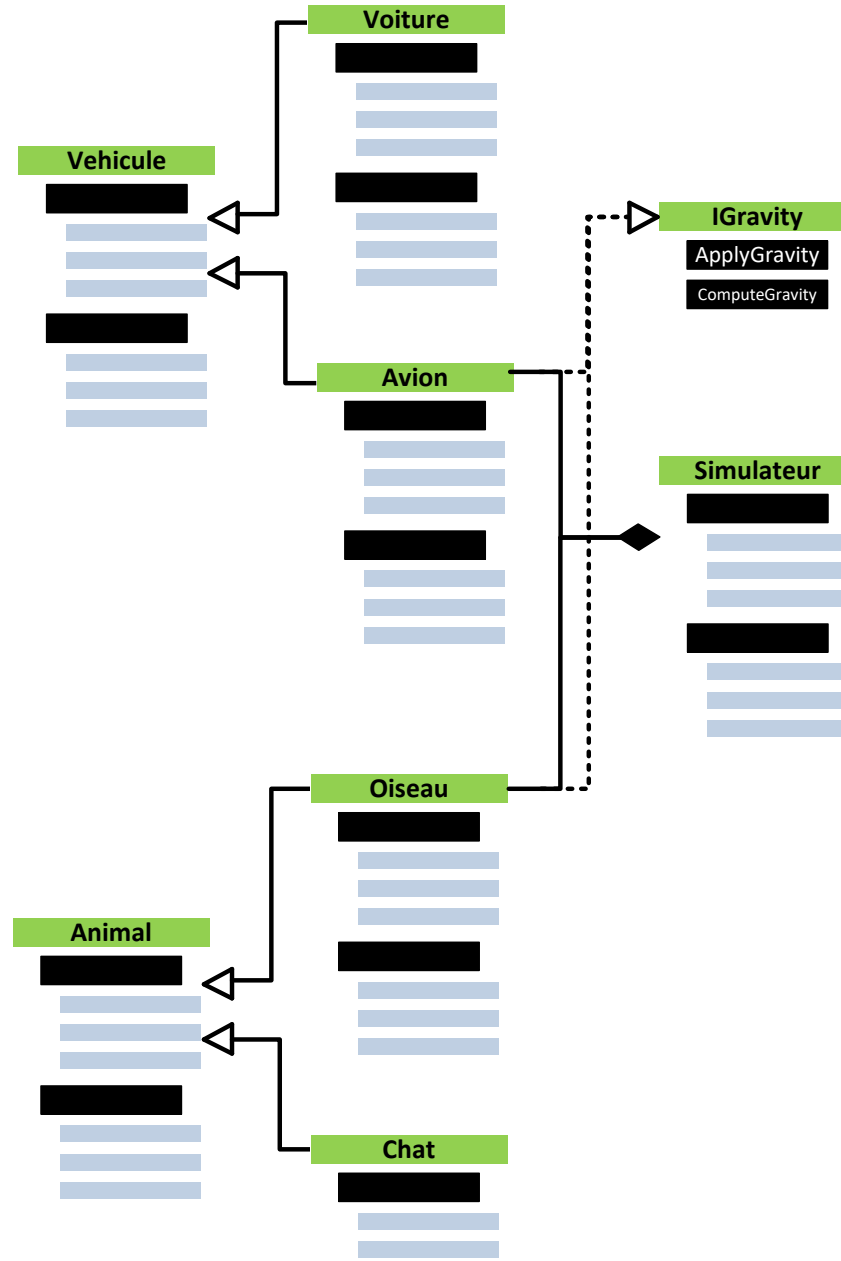


# Interface












- ❑ **Une interface** java est un groupe de méthodes (nom) sans comportement définit (body)
- ❑ **Les Objets (Class)** se référant à cette **interface (implements)** devront définir le comportement de ces méthodes
- ❑ **Les Objets** peuvent posséder **plusieurs interfaces**



# Interface














# Interface

- ▼  interfaces
  - ▼  animal
    - >  Animal.java
    - >  Chat.java
    - >  Oiseau.java
  - ▼  vehicule
    - >  Avion.java
    - >  Vehicule.java
    - >  Voiture.java
  - >  IGravity.java
  - >  Simulator.java












```
public interface IGravity {  
    public void applyGravity(float[] forces);  
    public void computeGravity(float[] forces);  
}
```

# Interface

- ▼  interfaces
  - ▼  animal
    - >  Animal.java
    - >  Chat.java
    - >  Oiseau.java
  - ▼  vehicule
    - >  Avion.java
    - >  Vehicule.java
    - >  Voiture.java
  - >  IGravity.java
  - >  Simulator.java

```
public class Avion extends Vehicule implements IGravity{  
    private String name;  
    public Avion(String name) {  
        this.name=name;  
    }  
    @Override  
    public void applyGravity(float[] forces) {  
        ...  
    }  
    @Override  
    public void computeGravity(float[] forces) {  
        ...  
    }  
}
```

# Interface

- ▼  interfaces
  - ▼  animal
    - >  Animal.java
    - >  Chat.java
    - >  Oiseau.java
  - ▼  vehicule
    - >  Avion.java
    - >  Vehicule.java
    - >  Voiture.java
  - >  IGravity.java
  - >  Simulator.java












```
public class Oiseau extends Animal implements IGravity{
    private String name;

    public Oiseau(String name) {
        this.name=name;
    }

    @Override
    public void applyGravity(float[] forces) {
        ...
    }

    @Override
    public void computeGravity(float[] forces) {
        ...
    }
}
```

# Interface

- ▼  interfaces
  - ▼  animal
    - >  Animal.java
    - >  Chat.java
    - >  Oiseau.java
  - ▼  vehicule
    - >  Avion.java
    - >  Vehicule.java
    - >  Voiture.java
  - >  IGravity.java
  - >  Simulator.java

```
public class Simulator {  
    IGravity[] gtab;  
  
    public Simulator() {  
        this.gtab=new IGravity[5];  
        this.gtab[0]= new Avion("Avion1");  
        this.gtab[1]= new Avion("Avion2");  
        this.gtab[2]= new Oiseau("Oiseau1");  
        this.gtab[3]= new Oiseau("Oiseau2");  
        this.gtab[4]= new Oiseau("Oiseau3");  
    }  
  
    public void process(){  
        for(int i=0; i<gtab.length;i++){  
            float[] forces={45,52,10.2f};  
            this.gtab[i].applyGravity(forces);  
        }  
    }  
}
```

# Exercice

Modéliser un **Simulateur** possédant des **Robot, Human, Animal**.

Chaque **Robot** possède:

- Id, name
- humanDetection()
- Il existe des catégories de **Robot(price)**:
  - HumanoidRobot (speed, automy)
  - FixPresentationRobot(voiceRange)
  - AutonomousVehicle (speed, automy)
- Il existe des catégories d'**Human(age)**:
  - BabyH (voiceRange)
  - MiddleAgeH (speed)
- Il existe des catégories d'**Animal (age)**:
  - Cat (speed)
  - Fish (speed)
  - Corals (isToxic,color)
- Les catégories Cat,Fish MiddleAgeH,AutonomousVehicule, HumanoidRobot doivent remplir les fonctions suivantes:
  - Accelerate(), break(), setOrientation(x,y,z)

Modéliser le Simulateur permettant d'interagir avec ces Entités



# Clone et égalité



## ❑ Comparaison

- Comment comparer 2 objets ?
  - Type de base operateur == suffisant
  - Que faire pour les objets ?
    - utiliser la méthode **equal** héritée de la classe Objet

```
AObject a1;  
AObject a2;  
  
if( a1.equals(a2)){  
    // a1 == a2  
}
```



# Clone et égalité



## ❑ Comparaison



Si la méthode n'est **equal** n'est pas redéfinie → test si les deux références pointent vers le même objet

```
AObject a1;  
AObject a2;  
  
if( a1.equals(a2)){  
    // a1 = a2  
}
```

```
@Override  
public boolean equals(Object obj) {  
    // vérification du type d'objet  
    if (obj instanceof AObject){  
        // Cast de l'objet  
        AObject aTmp=(AObject)obj;  
  
        if(this.id==aTmp.getId() &&  
            this.name == aTmp.getName()){  
            return true;  
        }  
    }  
    return false;  
}
```

Vérification du type d'Objet passé en paramètre

**Cast** d'un type d'objet vers un autre type: déclaration explicite du type d'objet générique vers un plus spécifique

# Clone et égalité



## ❑ Copie d'un objet

### ▪ Comment dupliquer un objet ?

- Définir une nouvelle instance d'un objet ayant les mêmes propriétés qu'une instance existante
- Transmettre une copie à une méthode et non une référence

→ S'appuyer la méthode **clone** héritée de la classe

**Objet**

→ Informer les autres objets de la possibilité de copie **implement Clonable**

# Clone et égalité



## ❑ Copie d'un objet

### ▪ Object.clone

- **Copie** de l'ensemble des éléments **attribut par attribut**
- Attention pour **les objets** → **même pointeur** sur l'objet en mémoire
- **Redéfinir la copie** pour les **attributs objets**

# Clone et égalité



## ❑ Copie d'un objet

```
public class AcObject implements Cloneable{
    private String name;
    private int id;
    private BObject b;
    private CObject c;

    public AcObject(String name,int id) {
        this.name=name;
        this.id=id;
        b=new BObject();
    }
    ...
}
```

```
...
@Override
public AcObject clone() throws
    CloneNotSupportedException {
    AcObject aCpoy=(AcObject)super.clone();

    //utilisation d'une class clonable
    aCpoy.setB(this.b.clone());

    //création explicite d'une nouvel objet
    CObject cCopy=new CObject();
    cCopy.setX(this.c.getX());
    cCopy.setY(this.c.getY());

    aCpoy.setC(cCopy);

    return aCpoy;
}
```

# Exercice

Une imprimerie possède un stock de Livres

Un Livre est composé de:

- Nbr\_page, prix, date
- Auteur (nom prenom)
- Editeur (nom, adress)

L'imprimerie va réaliser des copies du stock de livres en changeant l'éditeur par le sien et en majorant le prix de 10%

L'imprimerie est composée de:

- Livre[] livreOriginalTab
- Livre[] livreReeditionTab
- Editeur (nom, adress)

Réaliser le programme associé



# Classe Abstraite

## ❑ Définition

Class composée d'attributs et de comportements par défaut (cf héritage) et de comportements obligatoires à spécifier ultérieurement (par les classes filles)

## ❑ Propriétés:

- **Ne peut pas être instanciée directement**
- Peut hériter de propriétés d'autres classes concrètes ou abstraites
- Peut posséder des interfaces
- **Possède des méthodes sans body** (signature de méthode)  
représentant les méthodes obligatoires à définir ultérieurement:  
→ **Méthodes abstraites**

```
public abstract class MyClassAbstract {  
    ...  
    public abstract void myFnct(float volume);  
    ...  
}
```

# Classe Abstraite

```
public abstract class AnimalAbstract {  
    private String name;  
    private float weight;  
    private float height;  
    private int age;  
  
    public AnimalAbstract(String name, float  
                           weight, float heigh) {  
        this.name=name;  
        this.weight=weight;  
        this.height=heigh;  
    }  
    // méthode concrète  
    public void incAge(){  
        this.age++;  
    }  
    // méthode concrète  
    public int getAge(){  
        return this.age;  
    }  
    // méthode abstraite  
    public abstract void eat(float volume);  
}
```

```
public class Cat extends AnimalAbstract  
{  
  
    public Cat(String name, float  
               weight, float height) {  
        super(name, weight, height);  
    }  
  
    @Override  
    public void eat(float volume) {  
        ...  
    }  
  
    @Override  
    public void sleep(float volume) {  
        ...  
    }  
}
```

# Abstract vs Interface

- ☐ Définit des propriétés par défaut

(cf attributs)

- ☐ Définit des comportement par défaut

(cf méthodes concrètes)

- ☐ Définit des comportements

obligatoires à définir

(cf méthodes abstraites)

- ☐ 1 Seul héritage possible

- ☐ Définit des comportement par défaut

- ☐ Plusieurs interfaces possibles



# Exercice

Modéliser un **Contrôle de consommation de véhicule** possédant des **Véhicules**.

Chaque **Véhicule** possède:

- Poids, consommation/km
- Il existe des catégories d'**Véhicules**:
  - **Voiture**
    - Présence d'un turbo
    - Poids charge complémentaires (caravane)
  - **Moto**
    - Présence d'un sideCare
    - Poids du sideCare
  - **Camion**
    - Nbre de remorques
    - Poids des remorques



Chaque véhicule devra fournir sa consommation suivant un kilométrage (km) défini

# Exercice

Calcul consommation

## **Voiture**

$\text{Consommation} * \text{km} * 1,25 (\text{si turbo}) + \text{poidsSup} / \text{poids} * \text{consommation} * \text{km}$

## **Moto**

$\text{Consommation} * \text{km} + \text{poidsSup} / \text{poids} * \text{consommation} * 1,5 * \text{km}$

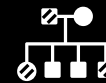
## **Camion**

$\text{Consommation} * \text{km} + \text{poidsSup} / \text{poids} * \text{consommation} * 1,5 * \text{nbrRemorque} * \text{km}$





# Les objets *Collections* de JAVA



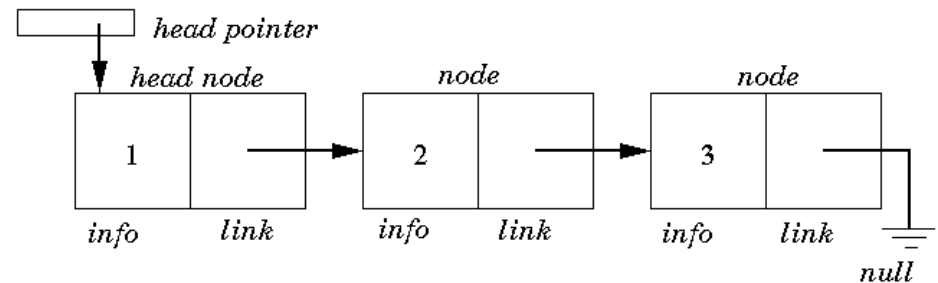
# Les Collections

## ❑ Définition

Ensemble d'objet fournit par JAVA permettant de **stocker/rechercher/restituer** des collections d'objets

## ❑ Exemple

- Enum
- List
- Queue
- Map



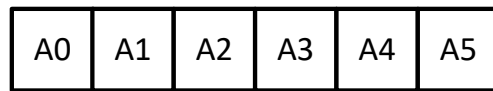
**A Linked List**

<https://people.engr.ncsu.edu/efg/210/s99/Notes/LinkedList.1.html>

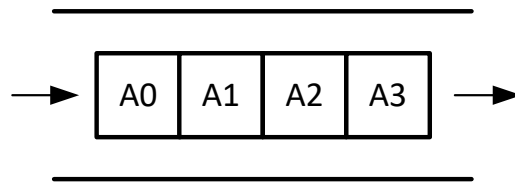


# Les Collections

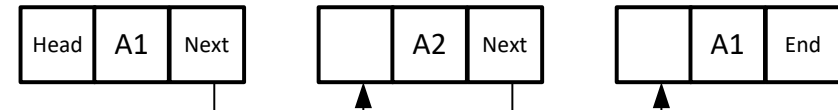
## ❑ Rappel (1/2)



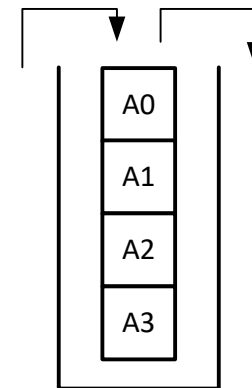
List



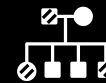
First In First Out  
FIFO



Chained List

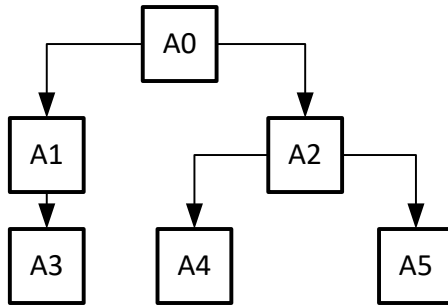


Last In First Out  
LIFO

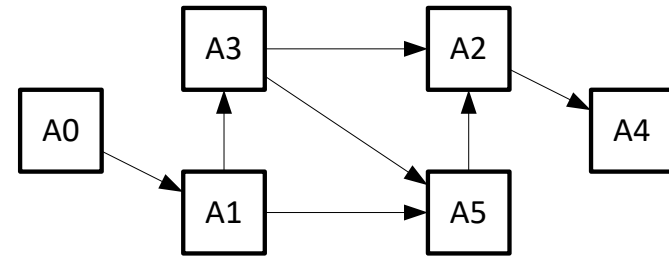


# Les Collections

## ❑ Rappel (2/2)



Tree



Graph

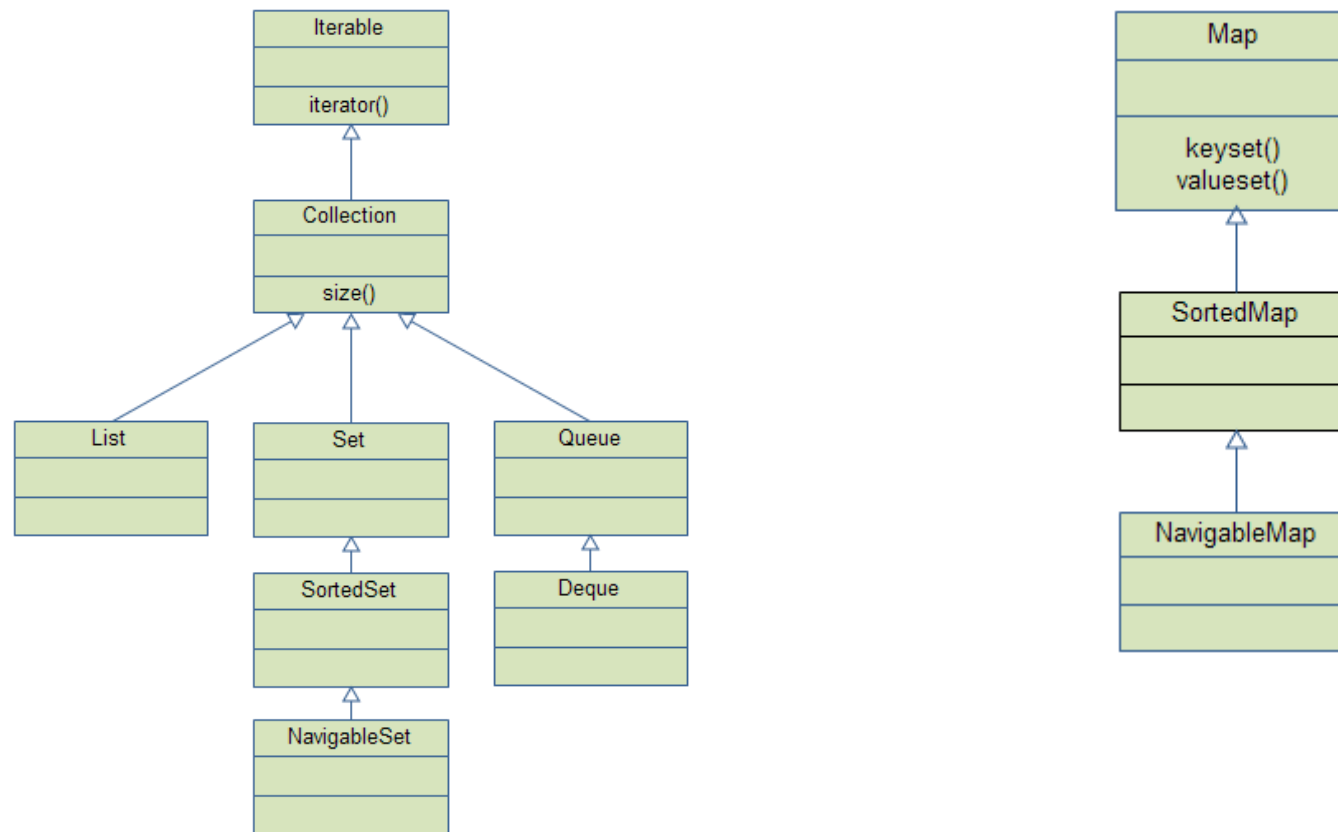
key0	A0
key1	A1
Key2	A2

Map



# Les Collections

## ❑ Les Collections de JAVA (les interfaces)



<http://tutorials.jenkov.com/java-collections/overview.html>

Copyright © Jacques Saraydaryan



# Les Collections

## ❑ Iterable

**Interface** permettant aux objets Collection d'être parcourus notamment dans les boucles for de java

## ❑ Collection

**Interface** représentant l'ensemble des opérations possibles communes sur un ensemble de données

L'accès aux données est principalement basé sur l'index

## ❑ Map

**Interface** permettant d'associer une valeur à une clé. Clés et valeurs sont des objets.

Le couple clé-valeur est inséré, et la valeur (l'objet) est retrouvé grâce à sa clé.





# Les Collections

## ☐ List

**Interface** représentant un ensemble **ordonné** d'objets (garanti l'ordre de parcours des objets avec un itérateur)

## ☐ Set

**Interface** représentant un ensemble **non-ordonné** d'objets (aucune garantie de l'ordre de parcours des objets avec un itérateur)

## ☐ Queue

**Interface** représentant une structure de queue classique (FIFO)

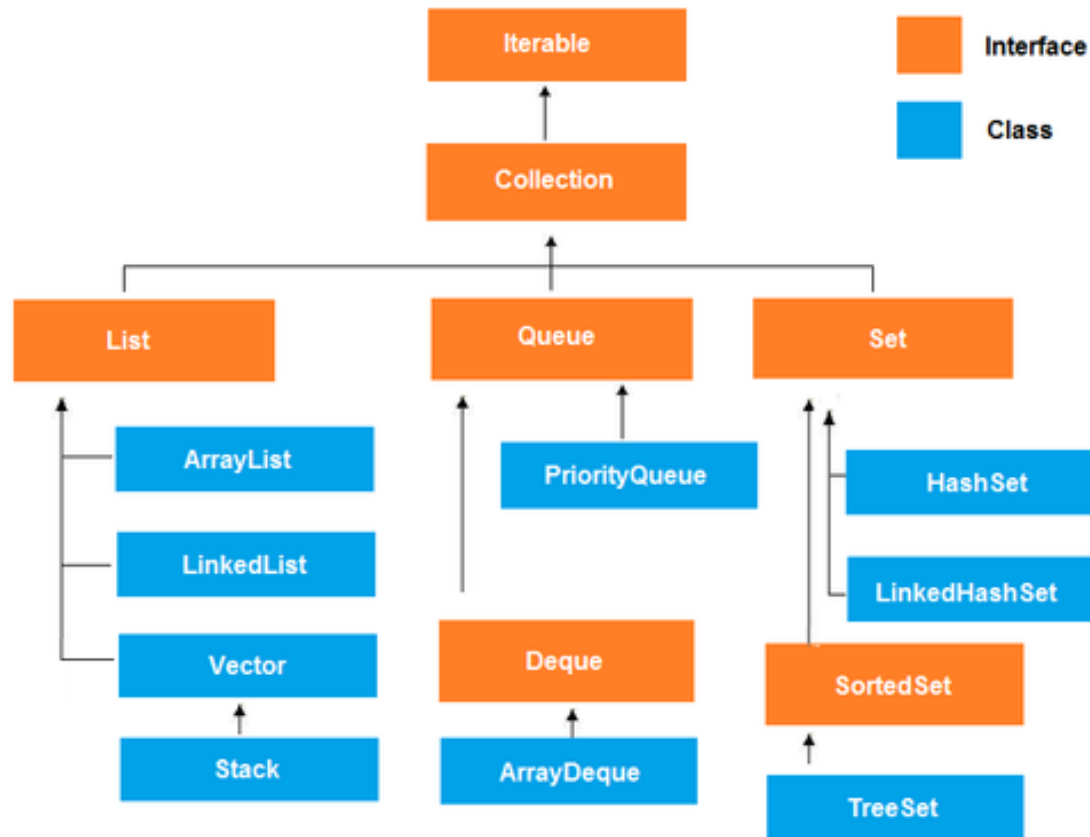
## ☐ Stack

**Interface** représentant une structure de pile classique (LIFO)



# Les Collections

## ❑ Les Collections de JAVA (les classes et les interfaces)



<http://www.beingjavaguys.com/2013/03/java-collection-framework.html>

Copyright © Jacques Saraydaryan



# Les Collections

## ❑ Les Collections de JAVA : les méthodes (communes)

Methods	
Modifier and Type	Method and Description
boolean	<b>add(E e)</b> Ensures that this collection contains the specified element (optional operation).
boolean	<b>addAll(Collection&lt;? extends E&gt; c)</b> Adds all of the elements in the specified collection to this collection (optional operation).
void	<b>clear()</b> Removes all of the elements from this collection (optional operation).
boolean	<b>contains(Object o)</b> Returns true if this collection contains the specified element.
boolean	<b>containsAll(Collection&lt;?&gt; c)</b> Returns true if this collection contains all of the elements in the specified collection.
boolean	<b>equals(Object o)</b> Compares the specified object with this collection for equality.
int	<b>hashCode()</b> Returns the hash code value for this collection.
boolean	<b>isEmpty()</b> Returns true if this collection contains no elements.
Iterator<E>	<b>iterator()</b> Returns an iterator over the elements in this collection.
boolean	<b>remove(Object o)</b> Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	<b>removeAll(Collection&lt;?&gt; c)</b> Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	<b>retainAll(Collection&lt;?&gt; c)</b> Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	<b>size()</b> Returns the number of elements in this collection.
Object[]	<b>toArray()</b> Returns an array containing all of the elements in this collection.
<T> T[]	<b>toArray(T[] a)</b> Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

<https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html>



# Les Collections

- ❑ Comment choisir ma collection ?





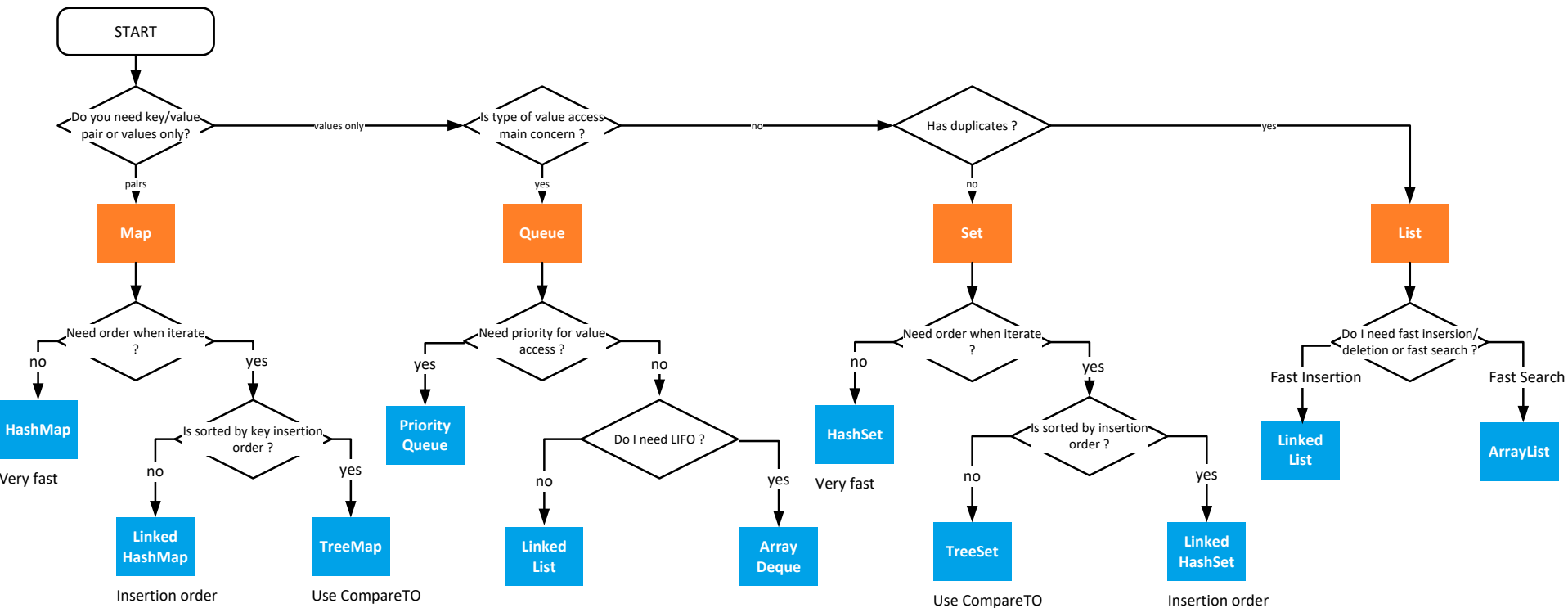
# Les Collections

- ❑ **Comment choisir ma collection ?**
- ❑ **Quelques questions à se poser:**
  - Peut-on avoir des doublons ?
  - L'ordre des objets est-il important ?
  - Quel type d'accès aux objets est nécessaire (séquentiel, par index, par clé) ?
  - Quelle est la performance d'accès aux données souhaitées ?



# Les Collections

## ❏ Comment choisir ma collection ?





# Les Collections

## ❑ Comment choisir ma collection ?

### HashMap

- Permet de stocker et retrouver des objets sous forme de clé-valeur (utilise un tableau de liste chaînées pour stocker les objets, index d'une clé = HASH(Objet) )
- Caractéristiques:
  - Utilisation de la valeur *null* possible pour clé et valeur
  - Pas d'accès concurrent garanti
  - Ne garantit aucun ordre lors du parcours de la collection

### Linked HashMap

- Liste doublement chaînée pour maintenir par default les éléments par ordre d'insertion
- Caractéristiques:
  - garantit l'ordre lors du parcours de la collection
  - Constructeur `accessOrder`: `true` : les éléments sont triés du moins accédés au plus accédés, `false` : les éléments sont triés dans leur ordre d'insertion



# Les Collections

## ❑ Comment choisir ma collection ?

### TreeMap

- Permet de stocker et retrouver des objets sous forme de clé-valeur de manière triée dans un arbre de type Red-black tree.
- Caractéristiques:
  - Trié par ordre naturel des clés ou en utilisant une instance **Comparator**
  - Pas d'accès concurrent garanti
  - Ne garantit aucun ordre lors du parcours de la collection





# Les Collections

## ❑ Comment choisir ma collection ? MAP

```
AObject a=new AObject("A", 0);  
AObject a1=new AObject("A1", 1);  
AObject a2=new AObject("A2", 2);
```

Création des objets

```
HashMap<String, AObject> myHashMap;  
//LinkedHashMap<String, AObject> myHashMap;  
//TreeMap<String, AObject> myHashMap;  
myHashMap=new HashMap<>();
```

Déclaration/Création de la Map

```
myHashMap.put("Ak", a);  
myHashMap.put(a1.getName(), a1);  
myHashMap.put(a2.getName(), a2);
```

Stockage des objets dans la Map

```
for (Entry<String, AObject> objSet : myHashMap.entrySet()) {  
    System.out.println("key:"+objSet.getKey()+",  
                        Obj:"+objSet.getValue());  
}
```

Parcours des clé-valeur de la Map

```
myHashMap.get("Ak")
```



# Les Collections

## ❑ Comment choisir ma collection ?

### ArrayList

- Forme de List la plus simple, tableau de taille dynamique (utilise un tableau de liste chaînées pour stocker les objets, index d'une clé = HASH(Objet) )
- Caractéristiques:
  - Tableau pour stocker les éléments (premier elt à index 0, null autorisé)
  - Pas d'accès concurrent garanti
  - Accès réalisé grâce à l'index
  - Recherche rapide

### Linked List

- Liste doublement chaînée , les éléments sont reliés par des pointeurs, utilisée pour réaliser des FIFO
- Caractéristiques:
  - Pas besoin d'être redimensionné
  - Insertion rapide



# Les Collections

## ❑ Comment choisir ma collection ? LIST

```
AObject a=new AObject("A", 0);  
AObject a1=new AObject("A1", 1);  
AObject a2=new AObject("A2", 2);
```

→ Création des objets

```
ArrayList<AObject> myArray=new ArrayList<>();  
//LinkedList<AObject> myArray=new LinkedList<>();
```

→ Création des listes

```
myArray.add(a);  
myArray.add(a1);  
myArray.add(a2);
```

→ Ajout des objets dans la liste

```
for (AObject obj : myArray) {  
    System.out.println("Obj: "+obj);  
}
```

→ Récupération des objets de la liste

```
System.out.println("Obj: "+myArray.get(2));
```

→ Récupération d'un objet unique

```
}
```



# Les Collections

## ❑ Comment choisir ma collection ?

### HashSet

- Set qui utilise une HashMap et comme clé le HASH de l'objet stocké (doublons interdits)
- Caractéristiques:
  - Aucune garantie sur l'ordre de parcours
  - Doublons interdits (null autorisé)
  - Objets insérés doivent définir les méthodes **equals** (et hashCode())

### Linked HashSet

- Cf HashList , FIFO sans doublons



# Les Collections

## ❑ Comment choisir ma collection ?

TreeSet

- Stocke ses éléments de manière ordonnée en les comparant entre-eux.
- Caractéristiques:
  - Restitution des éléments:
    - Ordre naturel (si implémente **Comparable**)
    - Ordre obtenu en utilisant une instance de type **Comparator**



# Les Collections

## ❑ Comment choisir ma collection ? SET

```
public class SetSample {
```

```
    public static void main(String[] args) {
```

```
        AObject a = new AObject("A", 0);
```

```
        AObject a1 = new AObject("A1", 1);
```

```
        AObject a2 = new AObject("A2", 2);
```

```
        AObjectComparator comparator= new AObjectComparator();
```

```
        TreeSet<AObject> myTree = new TreeSet<>(comparator);
```

Création d'un comparator  
Création d'un TreeSet

```
        myTree.add(a);
```

```
        myTree.add(a1);
```

```
        myTree.add(a2);
```

```
        for (AObject obj : myTree) {  
            System.out.println("Obj:"+obj);  
        }
```

Récupération des  
objets dans l'ordre  
respectant le  
comparator

```
        System.out.println("Obj:"+myTree.first());
```

```
    }
```

```
...}
```



# Les Collections

❑ Comment choisir ma collection ?    SET

```
...
static private class AObjectComparator implements Comparator<AObject> {
    // - means o2>o1, 0 means o2==o1, + means o2<o1
    @Override
    public int compare(AObject o1, AObject o2) {
        return o1.getName().compareTo(o2.getName());
    }
}

}
```



# Les Collections

## ❑ Comment choisir ma collection ?

### Array Deque

- Opérations accessibles aux 2 bouts de la queue
- Caractéristiques:
  - Pas de gestion de priorité possible
  - LIFO et FIFO Possible

### Priority Queue

- Queue dont les éléments sont ordonnés par l'ordre naturel des objets ou par l'usage d'un comparator.
- Caractéristiques:
  - Ne contient pas d'élément null
  - Operateurs (poll, remove, peek, element) renvoient le premier élément de la collection qui est le plus petit selon l'ordre de classement des éléments





# Les Iterators

## ❑ Définition:

Interface définissant des méthodes permettant de parcourir les données d'une collection

## ❑ Méthodes:

- **boolean hasNext()** Indiquer s'il reste au moins un élément à parcourir dans la collection
- **Object next()** Renvoyer le prochain élément dans la collection
- **void remove()** Supprimer le dernier élément parcouru



# Les Iterators

```
AObject a = new AObject("A", 0);  
AObject a1 = new AObject("A1", 1);  
AObject a2 = new AObject("A2", 2);
```

```
ArrayList<AObject> myArray = new ArrayList<>();
```

```
myArray.add(a);  
myArray.add(a1);  
myArray.add(a2);
```

```
Iterator<AObject> it=myArray.iterator();
```

→ Récupération d'un itérateur (collection)

```
while(it.hasNext()){
```

→ Vérification si d'autres éléments existent

```
    AObject current=it.next();
```

→ Récupération de l'élément suivant

```
    System.out.println(current);
```

```
}
```



# Exercice

Créer une Classe **Produit** (nom,prix)

Créer une Classe **Stock** possédant:

- une **HashMap** de **Produit** (nom produit = clé)
- **checkProduct()** retournant vrai si le produit existe

Créer une Classe **Launch** possédant

- Un **Stock** (à initialiser avec un ensemble de produits)
- une **Liste** de **String** (nom de Produit et autres noms)
- La classe **Launch** va vérifier l'ensemble des produits de sa List de String et reconstruire une autre Liste de Produits correspondant aux Produits valides



Option: Trier la liste de Produit par prix puis par nom

[https://docs.oracle.com/javase/6/docs/api/java/util/Collectionns.html#sort\(java.util.List,%20java.util.Comparator\)](https://docs.oracle.com/javase/6/docs/api/java/util/Collectionns.html#sort(java.util.List,%20java.util.Comparator))



# Exercice

Créer une classe **ProdutGenerator** créant des **Product** à la demande

Créer un **main** permettant d'appeler la génération de Product et passant ces produits dans une **FIFO**

Vider ensuite la **FIFO** en affichant l'objet traité

Réaliser la même opération à l'aide d'une **LIFO**.





# Les Génériques

## ❑ Définition

Un type générique est une classe ou interface permettant d'effectuer un traitement sur un objet dont le type est paramétrable (les objets sont typés au moment de la compilation).

## ❑ Usage

```
ArrayList<String> myListString=new ArrayList<>();  
ArrayList<Integer> myListInteger=new ArrayList<>();  
ArrayList<AObject> myListAObject=new ArrayList<>();
```

Definition du type souhaité à la déclaration de la classe

ArrayList<String> myListString

Nom de la classe générique



# Les Génériques

## ❑ Création d'une classe générique

```
public class Box {  
    private Object object;  
  
    public void set(Object object) {  
        this.object = object; }  
    public Object get() { return object; }  
}
```

```
public class Box<T> {  
    // T stands for "Type"  
    private T t;  
  
    public void set(T t) { this.t = t; }  
    public T get() { return t; }  
}
```

**class Box<T>** →

Définition de la classe générique usage des < > pour indiquer le type générique et du symbole **T** représentant le type d'objet générique

**private T t;** →

Usage du symbole **T** dès que le type doit être défini



Le symbole représentant le type (e.g T) ne peut pas être un type primitif (e.g int, float, double,...)



# Les Génériques

## ❑ Création d'une interface générique

```
public interface Pair<K, V> {  
    public K getKey();  
    public V getValue();  
}  
  
public class OrderedPair<K, V>  
    implements Pair<K, V> {  
  
    private K key;  
    private V value;  
  
    public OrderedPair(K key, V value) {  
this.key = key;  
this.value = value;  
    }  
  
    public K getKey(){ return key; }  
    public V getValue() { return value; }  
}
```

Convention de nommage des symboles :

- E - Element (used by the Java Collections)
- K - Key
- N - Number
- T - Type
- V - Value
- S,U,V etc. - 2nd, 3rd, 4th types

Déclaration de l'objet :

```
AObject a1=new AObject("a1", 1);  
OrderedPair<String,AObject> myPair=new  
    OrderedPair<String, AObject>("A1",a1);
```



# Questions ?





# References

# References

## ▪ Web references

- Classes package
  - <https://docs.oracle.com/javase/tutorial/java/javaOO/classdecl.html>
- Héritage / interface / classe abstraite
  - <https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html>
  - <https://docs.oracle.com/javase/tutorial/java/landl/index.html>
  - <https://programming.guide/java/clone-and-cloneable.html>
  - <https://dzone.com/articles/java-interface-vs-abstract-class>
  - <https://www.javaworld.com/article/2077421/learn-java/abstract-classes-vs-interfaces.html>
- Uml/Diagramme de classe
  - <http://users.teilar.gr/~gkakaran/oose/04.pdf>
  - <https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html>
- Collection
  - <https://www.mainjava.com/java/core-java/complete-collection-framework-in-java-with-programming-example/>
  - <http://tutorials.jenkov.com/java-collections/index.html>
  - <http://www.javapractices.com/topic/TopicAction.do?id=65>
- Générique
  - <https://docs.oracle.com/javase/tutorial/java/generics/types.html>
- Exception
  - <https://www.jmdoudoux.fr/java/dej/chap-exceptions.htm>
- Concurrency
  - <https://docs.oracle.com/javase/tutorial/essential/concurrency/>
- I/O
  - <https://docs.oracle.com/javase/tutorial/essential/io/charstreams.html>
- Annotation
  - <https://docs.oracle.com/javase/tutorial/java/annotations/index.html>
  - <https://www.jmdoudoux.fr/java/dej/chap-annotations.htm>

# References

- **Web references**

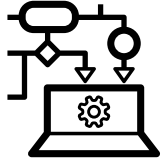
- Maven
  - <https://maven.apache.org/guides/getting-started/>
  - <https://java.developpez.com/tutoriels/java/maven-book/>
  - [http://igm.univ-mlv.fr/~dr/XPOSE2010/Apache\\_Maven/introduction.html](http://igm.univ-mlv.fr/~dr/XPOSE2010/Apache_Maven/introduction.html)
  - <https://mermet.users.greyc.fr/Enseignement/CoursPDF/maven.pdf>
- Tests
  - <http://www.test-recette.fr/tests-techniques/>
- JVM
  - <https://www.geeksforgeeks.org/jvm-works-jvm-architecture/> ->TB
  - <https://javatutorial.net/jvm-explained>
  - <https://www.cubrid.org/blog/understanding-jvm-internals/>

- **Livre / autres ressources**

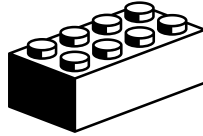
- Programmation Orientée Objet en Java, F. Perrin, CPE Lyon
- Kathy SIERRA et Bert BATES pour leur ouvrage Java -Tête la Première (O'Reilly edition)



Created by Denograph™  
from the Noun Project



Created by H Alberto Gongora  
from the Noun Project



Created by jon trillana



Created by lastspark  
from Noun Project



Created by Aha-Soft  
from Noun Project



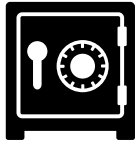
Created by AllredoCreates.com  
from Noun Project



Created by Trần Quang Hải  
from the Noun Project



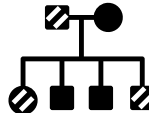
Created by Aldric Rodriguez  
from the Noun Project



Created by parkjeun  
from the Noun Project



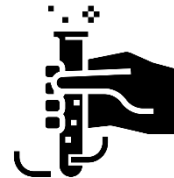
Created by parkjeun  
from the Noun Project



Created by dData



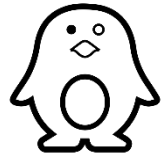
Created by Ogher Aloni  
from the Noun Project



Created by dData  
from Noun Project



Created by Krisada  
from Noun Project



Created by Ogher Aloni  
from the Noun Project



Created by Mohammed Tawqil Alam  
from Noun Project



Created by Delmar Hassan  
from Noun Project



**Jacques Saraydaryan**

**Jacques.saraydaryan@cpe.fr**