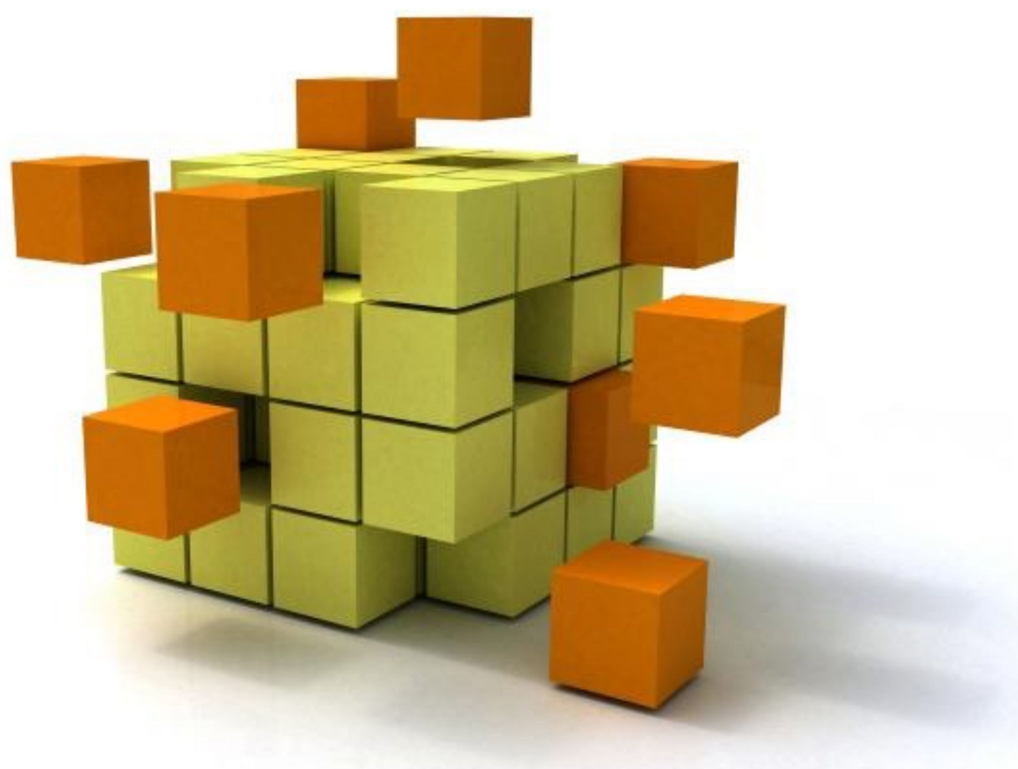


Applications concurrentes, mobiles et réparties en java

S7 Appro – Inf A3 EG

Stéphane Derrode, Alexandre Saïdi, Bât E6, 2ième étage
stephane.derrode@ec-lyon.fr

Organisation de l'AF

- **Grands chapitres**
  1. **Apprentissage de Java** : 4h de cours, 4h de TP, 2h d'autonomie – Stéphane Derrode
  2. **IHM en Java** : 4h de cours, 8h de TP, 4h autonomie (BE noté #1) – Stéphane Derrode
  3. **Prog. concurrente et distribuée** : 4h de cours, 4h de TP, 4h d'autonomie (BE noté #2) – Alexandre Saïdi
  4. **Prog. mobile** : 4h de cours, 4h de TP, 2h d'autonomie (BE noté #3) – Stéphane Derrode
- **Liens vers les énoncés :**

  http://perso.ec-lyon.fr/derrode.stephane/Teaching/ECL2A3A/ECL2A_Java/
- **Évaluation**
  - Examen papier : 50 %
  - Moyenne de 3 BEs : 50%

Nous utiliserons l'IDE VSCodium

Distribution des créneaux

ActivitÈ	Intervenant	Date	Heure		
Cours	S. Derrode	06/09/2023	10:15-12:15		
BE	S. Derrode	07/09/2023	08:00-10:00		
Cours	S. Derrode	07/09/2023	10:15-12:15		
BE	S. Derrode	08/09/2023	08:00-10:00	Module 1 (D	Apprentissage Java
BE	S. Derrode	11/09/2023	08:00-10:00	Module 2 (D	IHM en Java
Autonomie 25%	S. Derrode	14/09/2023	08:00-10:00	Module 3 (Sa	Prog. Concurrente et distribuée
Cours	S. Derrode	14/09/2023	10:15-12:15	Module 4 (D	Prog. Mobile
BE	S. Derrode	18/09/2023	08:00-10:00		
BE	S. Derrode	21/09/2023	08:00-10:00		
Cours	S. Derrode	21/09/2023	10:15-12:15		
Autonomie 25%	S. Derrode	25/09/2023	08:00-10:00		
BE	S. Derrode	28/09/2023	08:00-10:00		
BE	S. Derrode	28/09/2023	10:15-12:15		
Cours	A. Saidi	02/10/2023	08:00-10:00		
BE	A. Saidi	05/10/2023	08:00-10:00		
BE	A. Saidi	05/10/2023	10:15-12:15		
Cours	A. Saidi	09/10/2023	08:00-10:00		
BE	A. Saidi	12/10/2023	08:00-10:00		
Autonomie 75%	A. Saidi	12/10/2023	10:15-12:15		
Cours	S. Derrode	16/10/2023	08:00-10:00		
BE	S. Derrode	19/10/2023	08:00-10:00		
Cours	S. Derrode	19/10/2023	10:15-12:15		
BE	S. Derrode	26/10/2023	08:00-10:00		
Autonomie 25%	S. Derrode	26/10/2023	10:15-12:15		

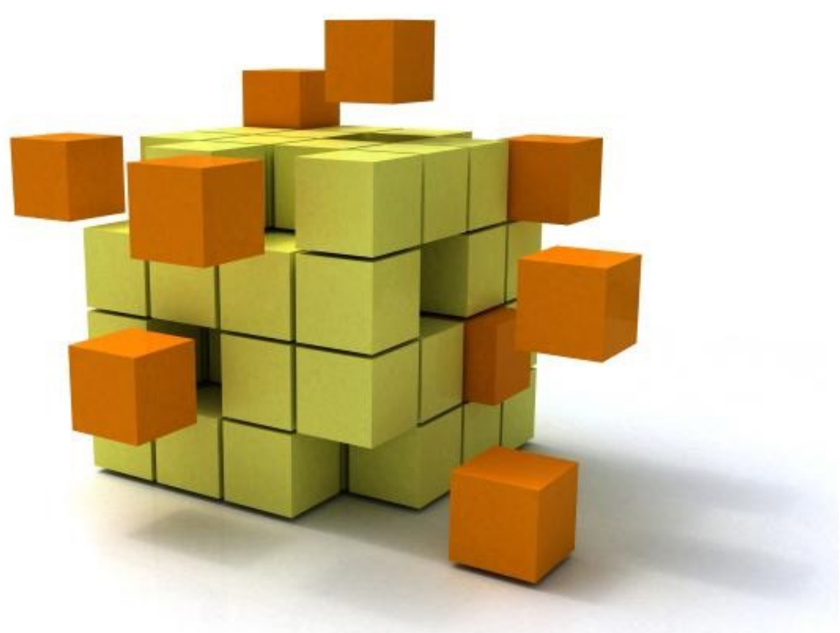
1- Apprentissage de Java

1. Paradigme objet
2. Java en quelques planches
3. Autour du *main() de Java*
4. Classes et objets
5. (Les énumérations)

Cours #1

6. Composition
7. Interfaces
8. Héritage
9. Les exceptions
10. Ce que l'on n'a pas vu…

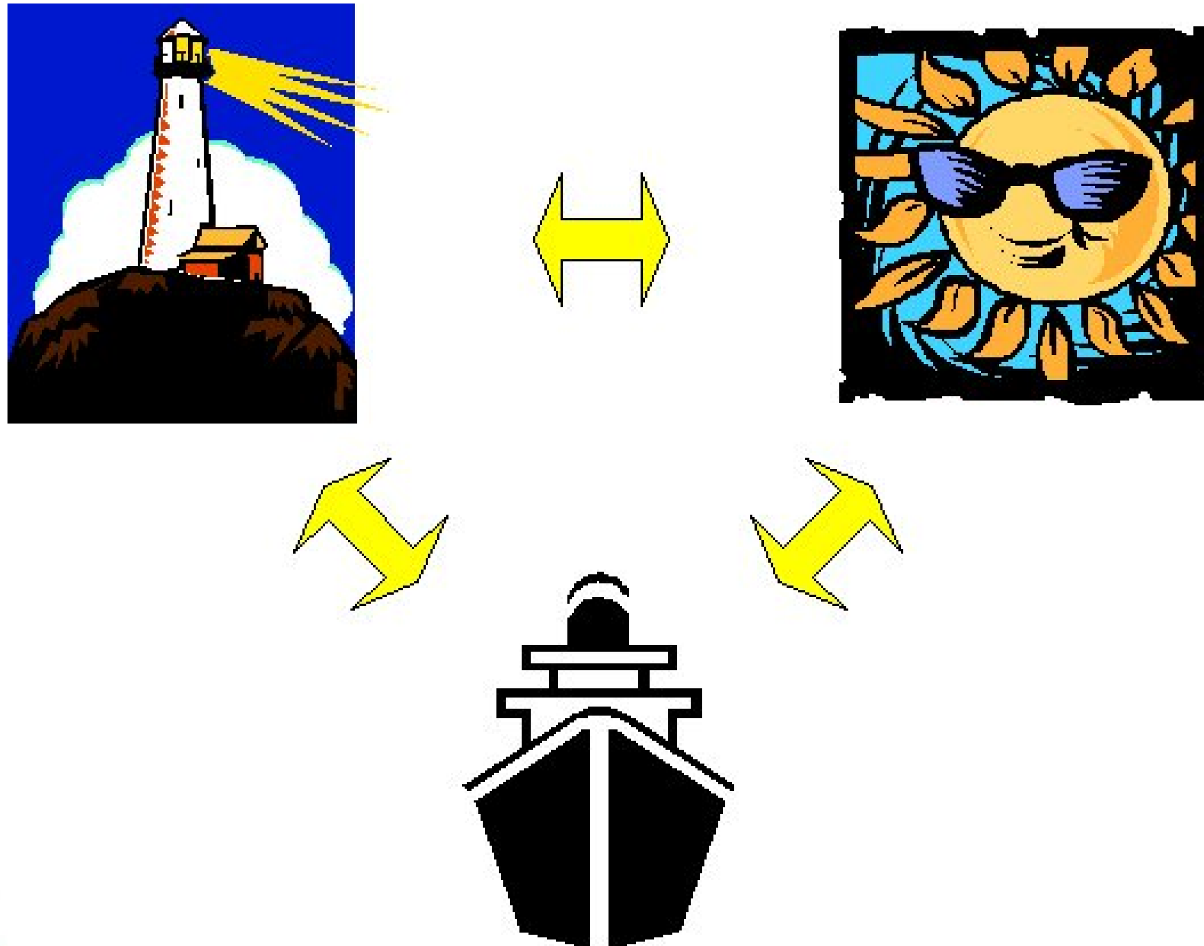
Cours #2



1- Paradigme objet

1- Paradigme objet

Le monde réel est composé d'objets autonomes qui sont en relation (communication) et coopèrent.



1- Paradigme objet

Ports PCI (6)
Port AGP
Ports parallèle et série
Ports USB
Port ISA(1)
Ports PS2
Socket (Processeur)
Chipset (ici sous ventilo)
Slot mémoire DIMM (3)
Ports IDE (4)

1- Paradigme objet

puissance
N
couleur
R
R
consommation
Klaxonner
Voiture
demarrer
R
N
places assises
mettre du carburant
R
volume carburant

1- Paradigme objet

Classe Produit

Attributs	Ref : chaine PrixHT : reel Quantite : entier
Methodes	reel GetPrix() SetPrix(reel) AddQuantite(entier) SubQuantite(entier)

Produit XX

Attributs
- Ref : UC 8954X
- PrixHT : 235.8
- Quantite : 212

Methodes
- GetPrix
- SetPrix
- AddQuantite
- SubQuantite

Produit YY

Attributs
- Ref : UX 751V
- PrixHT : 20.8
- Quantite : 2

Methodes
- GetPrix
- SetPrix
- AddQuantite
- SubQuantite

Produit ZZ

Attributs
- Ref : TC 124T
- PrixHT : 451.2
- Quantite : 214

Methodes
- GetPrix
- SetPrix
- AddQuantite
- SubQuantite

1- Paradigme objet

Classe A : objet O1

Methodes internes

Methodes

Methodes

Attributs

Classe B : objet x

Methodes internes

Attributs

Methodes

Methodes

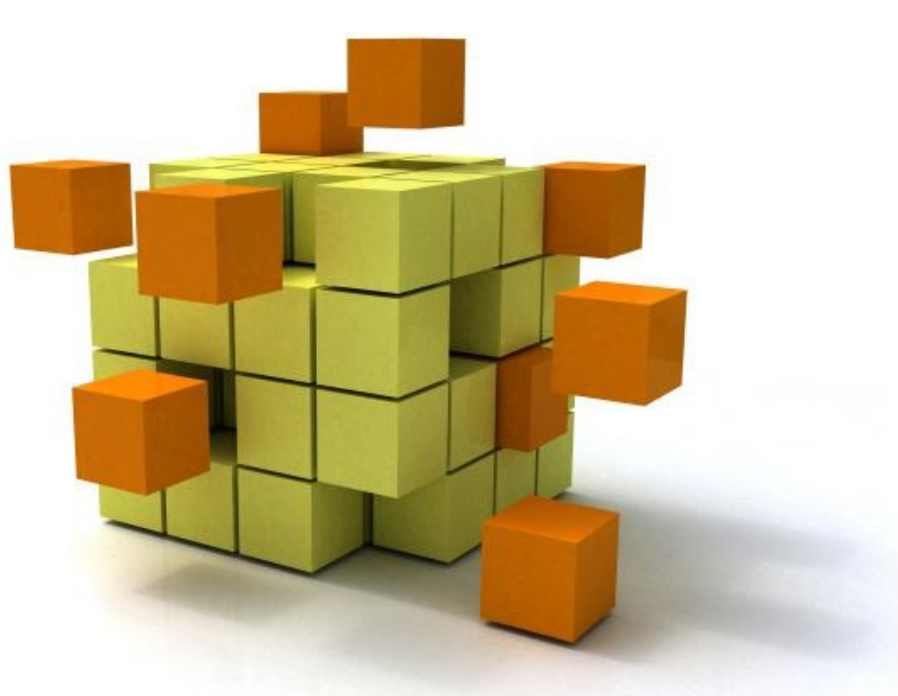
Methodes internes

Methodes

Methodes

Attributs

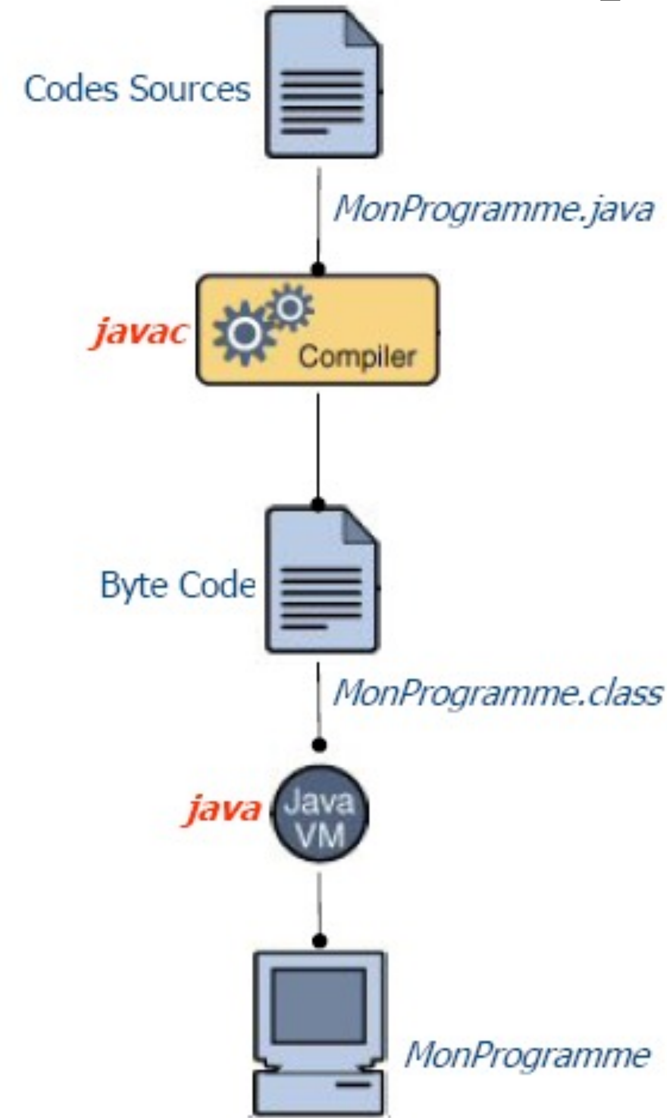
Classe A : Objet O2



2- Java en quelques planches

2- Java en quelques planches

- Création du code source
  - Outil : éditeur de texte, IDE
  - A partir des spécifications (par exemple en UML)
- Compilation en Byte-Code
  - A partir du code source
  - Outil : compilateur Java
- Diffusion sur l'architecture cible
  - Transfert du Byte-Code seul
  - Outils : réseau, disque, etc
- Exécution sur la machine cible
  - Exécution du Byte-Code
  - Outil : Machine Virtuelle Java

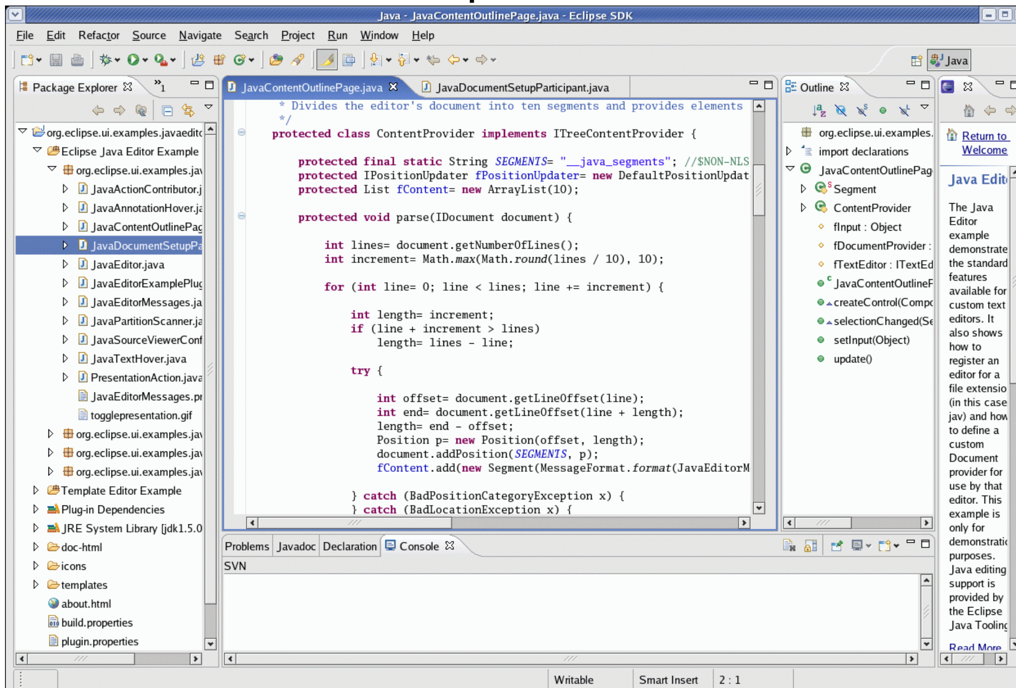


2- Java en quelques planches

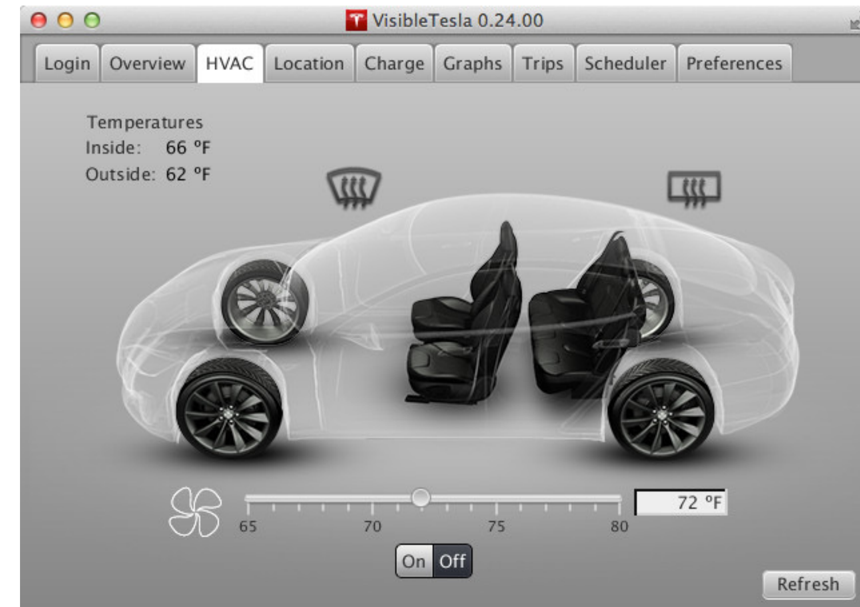
Voici la liste complète des 25 outils Java retenus par Oracle

Maestro Mars Rover Controller, JavaFX Deep Space Trajectory Explorer, NASA WorldWind, JMars, SBMT, Lucene, Hadoop, Parallel Graph AnalytiX, H2O, Minecraft, Jitter Robot & leJOS, Java applets, Netbeans & Eclipse, IntelliJ IDEA, Byte Buddy, Jenkins, GraalVM, Micronaut, Weblogic Tengah, Eclipse Collections, NSA Ghidra, Integrated Genome Browser, BioJava, VisibleTesla, SmartThings.

Eclipse

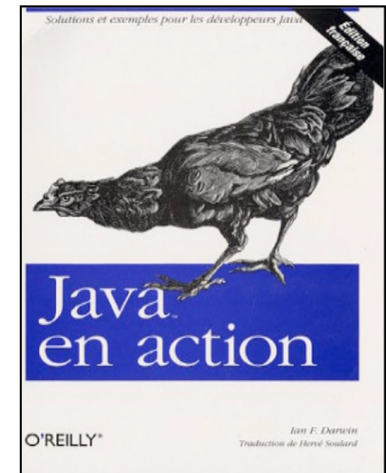


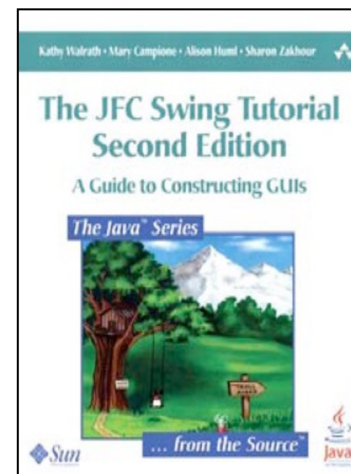
VisibleTesla

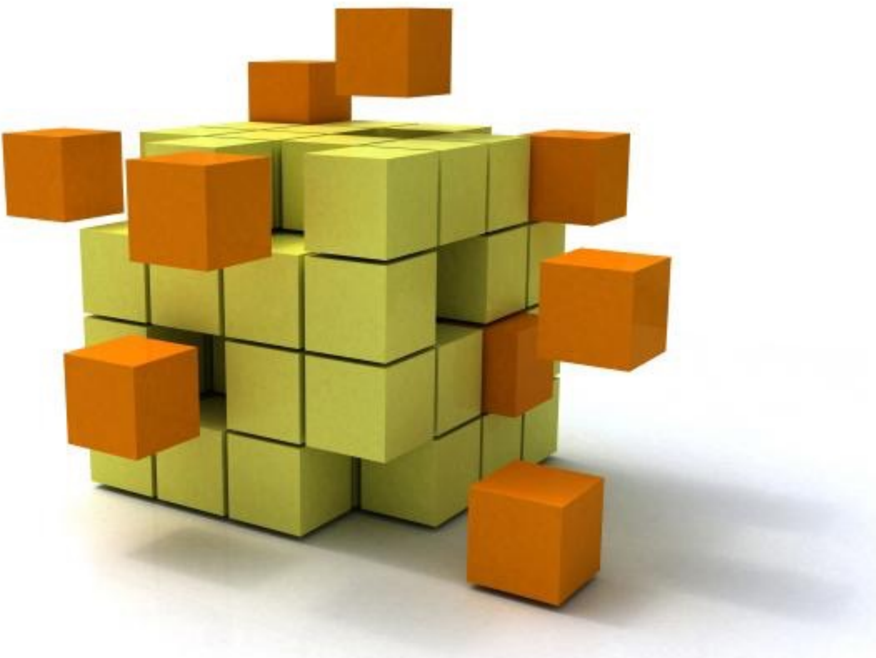


2- Java en quelques planches

- Programmer en Java
  - Auteur : Claude Delannoy
  - Éditeur : Eyrolles
- Java en action
  - Auteur : Ian F. Darwin
  - Éditeur : O'Reilly
- Packages Java :



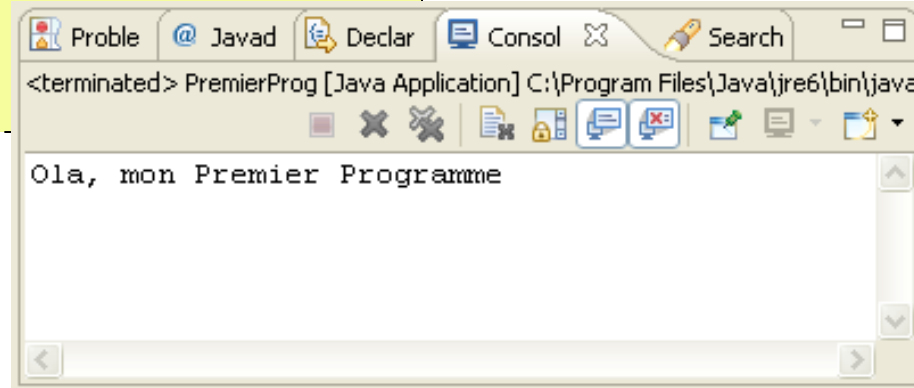




3- Autour du *main()* de Java

3- Autour du *main()* de Java

```
public class PremierProg {
    public static void main(String[] args) {
        System.out.println("Ola, mon Premier Programme");
    }
}
```



- public class PremierProg
  - Nom de la classe
- public static void main
  - La fonction principale
- String[] argv
  - Permet de récupérer des arguments transmis au programme au moment de son lancement
- System.out.println("Ola … ")
  - Méthode d'affichage dans la fenêtre console

3- Autour du *main()* de Java

- Fichiers
  - Un seul fichier « NomDeClasse.java »
  - Nom de la classe = Nom du fichier java
- Compilation
  - javac NomDeClasse.java
  - Génération d'un fichier Byte-Code « NomDeClasse.class »
  - Pas d'édition de liens (seulement une vérification)
- Exécution
  - java NomDeClasse (ne pas mettre l'extension .class pour l'exécution)
  - Choisir la classe principale à exécuter

3- Autour du *main()* de Java

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

                int      i = 1+3;
                byte     b = -3*-5;
                short    s = 5%2;
                long     l = 600851475143L;
                boolean o = true && (false || true);

                System.out.println(i+";"+ b+";"+ s+";"+ l+";"+ o);
        }
}
```

3- Autour du *main()* de Java

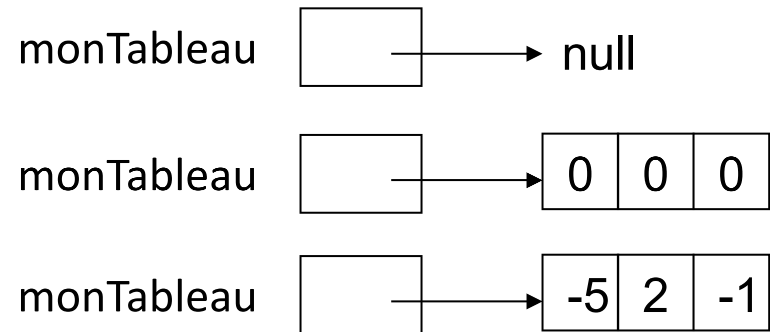
```
public class Test {
        public static void main(String[] args) {

                char    c = ‘o';
                float   f = 3.45678f;
                double d = -5e-10;
                String   s = "Bonjour";

                System.out.println(c+";"+ f+";"+ d+";"+ s);
        }
}
```

3- Autour du *main()* de Java

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

                int p = -8;
                p = p - 6;
                p++; p--;
                p += 2;

                float f = p;

                final double v = 5.4;
                short s = (short) v;
        }
}
```

3- Autour du *main()* de Java

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

		// 1- Declaration
		float[] monTableau = null;
		// 2- Allocation
		monTableau = new float[3];
		// 3- Initialisation
		monTableau[0] = -5;
		monTableau[1] = 2;
		monTableau[2] = -1;

		// Tout d'un coup !
		int[] Tab = {4, 7, 11};
	}
}
```



3- Autour du *main()* de Java

- Choix
  - Si alors sinon : « if condition {…} else {…} »

```
public class PremierProg {
  public static void main(String[] args) {
    int i = 1;
    if (i == 1)
      System.out.println("i vaut 1");
    else
      System.out.println("i ne vaut pas 1");
  }
}
```

3- Autour du *main()* de Java

– Switch case:

```
public class Test {
        public static void main(String[] args) {
                int month = 2;
                String monthString;
                switch (month) {
                  case 1:  monthString = "January";
                           break;
                  case 2:  monthString = "February";
                           break;
                  case 3:  monthString = "March";
                           break;
                  case 4:  monthString = "April";
                           break;
                  default: monthString = "Invalid month";
                           break;
                }
                System.out.println(monthString)
}}
```

3- Autour du *main()* de Java

- Itérations
  - Boucle : « for (initialisation ; condition ; modification) { … } »

```
for (int i=0; i<5; i++)
        System.out.println("i vaut " + i);

double[] Tab = {4.2, -7.1, 11.1};
for (int i=0; i<Tab.length; i++)
        System.out.println(Tab[i]);
```

  - Boucle (for each) : « for (Type var : Collection) { … } »

```
int[] Tab = {4, 7, 11};
for(int v : Tab)
        System.out.println(v);

String[] data = { "Toronto", "Stockholm" };
for (String s : data) {
        System.out.println(s);
}
```

Pr | Ja | De | Co

<terminated> Test [Java Application] /System/Li

```
4
7
11
Toronto
Stockholm
```

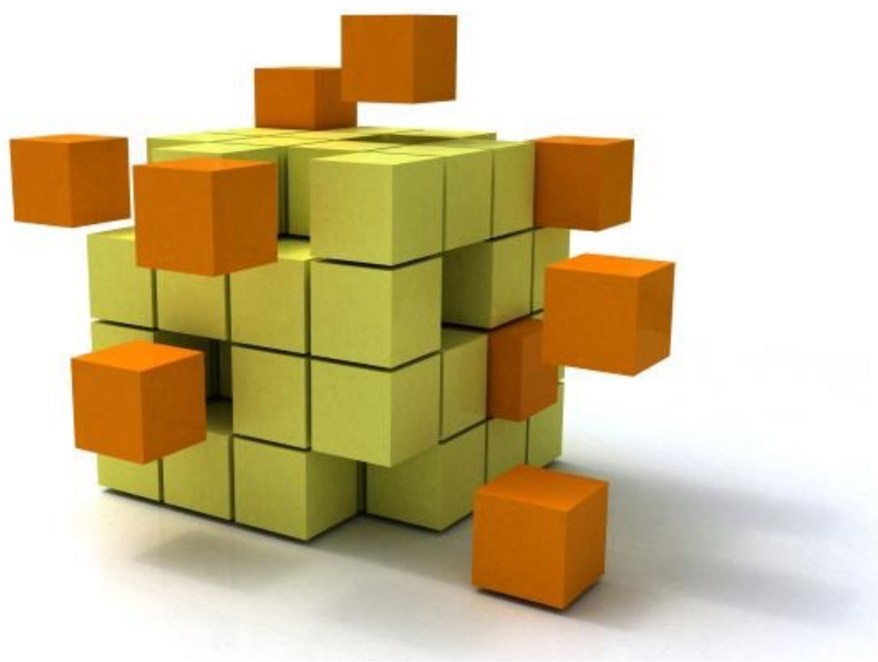
3- Autour du *main()* de Java

- Itérations (suite)
  - Tant que : « while (condition) { … } »

```
int i = 0;
while (i<5) {
    System.out.println("i vaut " + i);
    i++;
}
```

  - Faire jusqu'à : « do { … } while (condition) »

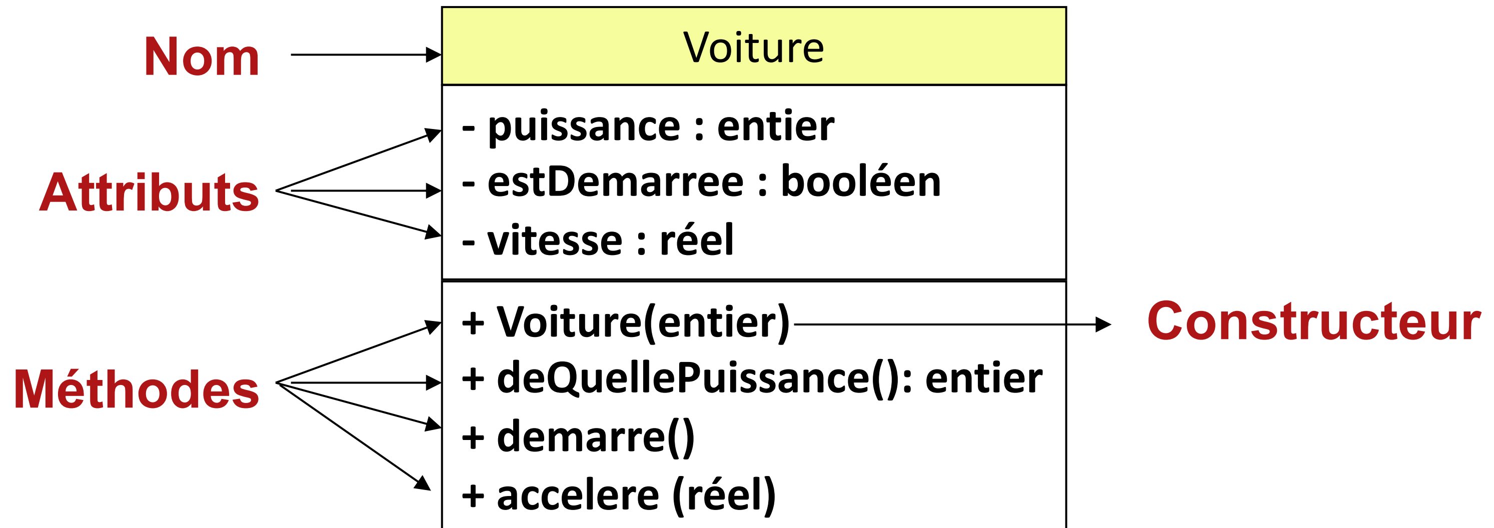
```
int i = 0;
do {
    System.out.println("i vaut " + i);
    i++;
} while (i!=5);
```



4- Classes et objets

4- Classes et objets

- Classe : représentation UML



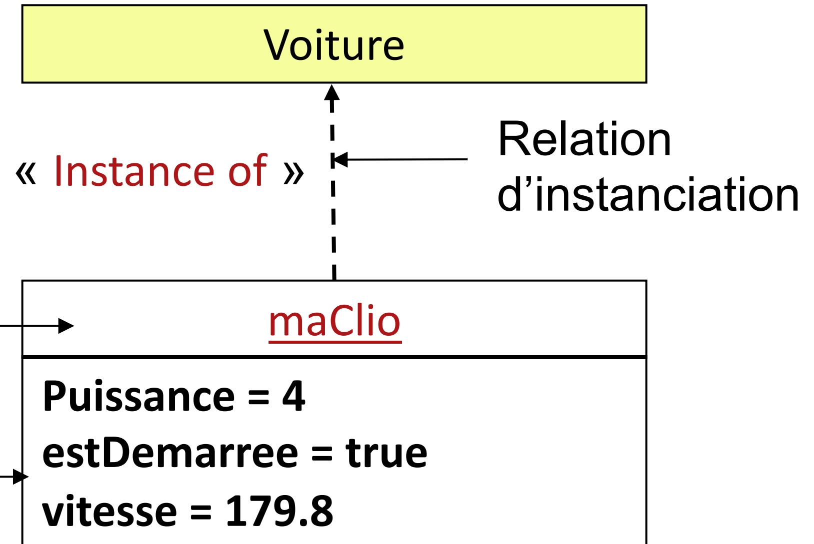
4- Classes et objets

```java
class Voiture {
  private int puissance;
  private boolean estDemarree;
  private double vitesse;

  public Voiture (int p) {
       if (p>0)
            puissance = p;
       else
            puissance = 5;
       estDemarree = false;
       vitesse = 0;
  }
  …
```

```java
  …
  public int deQuellePuissance() {
    return puissance;
  }
  public void demarre() {
    estDemarree = true;
  }
  public void accelere(double v) {
    if (estDemarree == true) {
      double avecTol;
      avecTol = v + 25;
       vitesse = vitesse + avecTol;
    }
  }
}
```

4- Classes et objets

- Un objet est **instance** d'une seule classe
  - Se conforme à la description que celle-ci fournit
  - Admet une valeur propre à l'objet pour chaque attribut de la classe
  - Les valeurs des attributs caractérisent l'**état** de l'objet
  - Possibilité de lui appliquer toute opération définie dans la classe
- Tout objet est manipulé et identifié par sa référence
  - On parle indifféremment d'**instance**, de **référence** ou d'**objet.**
  - maClio est une instance de Voiture



4- Classes et objets

- Chaque objet qui est une instance de la classe Voiture possède ses propres valeurs d'attributs

Voiture

« Instance of »

maClio : Voiture	saBerline : Voiture	sonKatKat : Voiture
Puissance = 4	**Puissance = 6**	**Puissance = 28**
estDemarree = true	**estDemarree = true**	**estDemarree = false**
Vitesse = 99.8	**Vitesse = 128.4**	**vitesse = 0**

4- Classes et objets

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

        Voiture maClio = null;
        maClio = new Voiture(4);
        maClio.demarre();
        maClio.accelere(99.8);

        Voiture saBerline = new Voiture(6);
        saBerline.demarre();
        saBerline.accelere(128.4);

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

~~Voiture sonKatKat = **new Voiture();**~~

```
    }
}
```

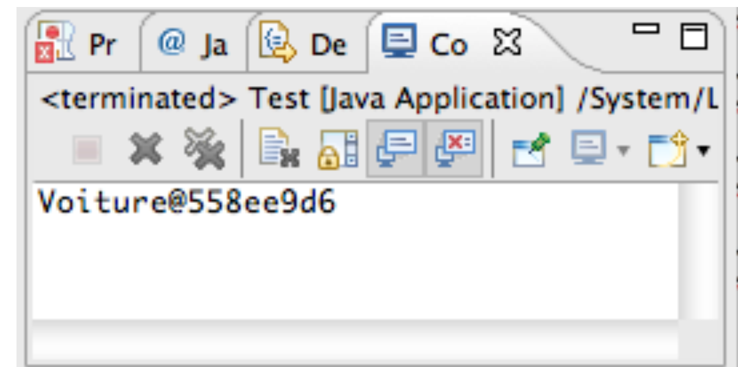
maClio → null

maClio → | 4 – false – 0 |

maClio → | 4 – true – 0 |

maClio → | 4 – true – 99.8 |

saBerline → | 6 – true – 128.4 |



4- Classes et objets

```
class Voiture {
  private int puissance;
  private boolean estDemarree;
  private double vitesse;
  public Voiture () {
      puissance = 5;
      estDemarree = false;
      vitesse = 0.;
  }
  public Voiture (int p) {
      puissance = p;
      estDemarree = false;
      vitesse = 0.;
  }
  public Voiture (int p, boolean d, double v) {
      puissance = p;
      estDemarree = d;
      vitesse = v;
  }
...}
```

Constructeurs multiples

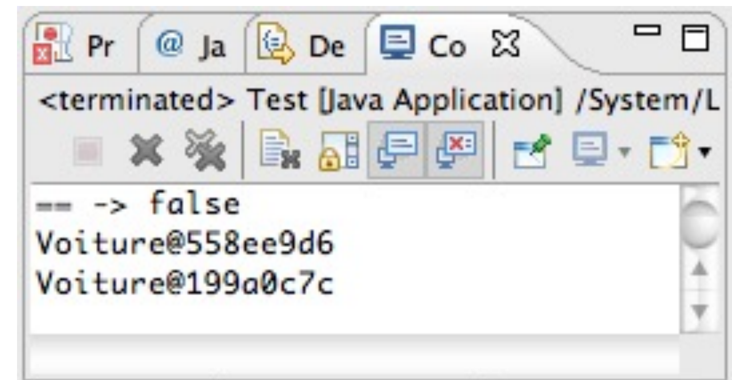
```
public class Test {
    public static void main(String[] args) {
        Voiture maClio    = new Voiture();
        Voiture saBerline = new Voiture(4);
        Voiture sonKatKat = new Voiture(28, true, 10.5);
    }
}
```

4- Classes et objets

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

        Voiture maClio = new Voiture(4);
        maClio.demarre();
        maClio.accelere(99.8);

        Voiture saPolo = new Voiture(4);
        saPolo.demarre();
        saPolo.accelere(99.8);

        if (maClio == saPolo)
            System.out.println("== -> true");
        else
            System.out.println("== -> false");

        System.out.println(maClio);
        System.out.println(saPolo);
    }
}
```

maClio → 4 – true – 99.8

saPolo → 4 – true – 99.8



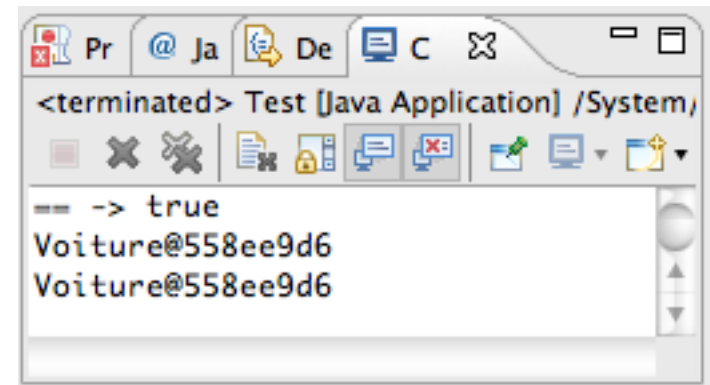
4- Classes et objets

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

        Voiture maClio = new Voiture(4);
        maClio.demarre();
        maClio.accelere(99.8);

        Voiture saBerline = new Voiture(6);

        saBerline  = maClio;
        if (maClio == saBerline)
            System.out.println("== -> true");
        else
            System.out.println("== -> false");

        System.out.println(maClio);
        System.out.println(saBerline);
    }
}
```

maClio → 4 – true – 99.8

saBerline → 6 – false – 0

maClio, saBerline → 4 – true – 99.8



4- Classes et objets

- Variables et constantes de classe par l'exemple

```
public class Voiture {
  private int poids;
  public static final int PTAC_MAX = 3500;
  private static int nbVoitCreees = 0;

  public Voiture(int p, ...) {
    poids = p;
    nbVoitCrees++;
  }
}
```

- Méthodes de classe par l'exemple

```
public class Voiture {
  private static int nbVoitCreees = 0;

  public static int getNbVoitCreees(){
    return Voiture.nbVoitCreees;
  }
}
```

```
public class TestMaVoiture {
  public static void main (String[] argv) {
    Voiture V1 = new Voiture(2500);
    System.out.println("Nbre Instance :" +  Voiture.getNbVoitCreees());
  }
}
```

4- Classes et objets

Tableau d' objets

1. Déclaration

```
Voiture[] monTableau;
```

2. Dimensionnement

```
monTableau = new Voiture[3];
```

3. Initialisation

```
monTableau[0] = new Voiture();
monTableau[1] = new Voiture(7);
monTableau[2] = new Voiture(8, true, 10);
```

1. 2. et 3.

```
Voiture[] monTableau = {
  new Voiture(),
  new Voiture(7),
  new Voiture(8, true, 10)
};
```

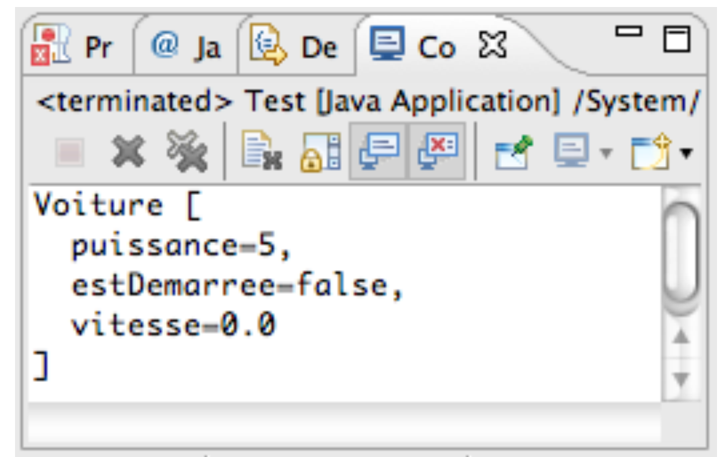
```
for (int i = 0; i < monTableau.length; i++)
  monTableau[i].demarre();
```

4- Classes et objets

```
class Voiture {
    private int puissance;
    private boolean estDemarree;
    private double vitesse;
    public Voiture (int p) {
        puissance = p;
        estDemarree = false;
        vitesse = 0.;
    }

    public String toString() {
        String S ="Voiture [\n"
          + "  puissance=" + puissance + ",\n"
          + "  estDemarree=" + estDemarree + ",\n"
          + "  vitesse=" + vitesse
          + "\n]";
        return S;
    }
...}
```

Méthode toString()

```
public class Test {
    public static void main(String[] args) {
        Voiture maClio = new Voiture (5);
        System.out.println(maClio);
    }
}
```

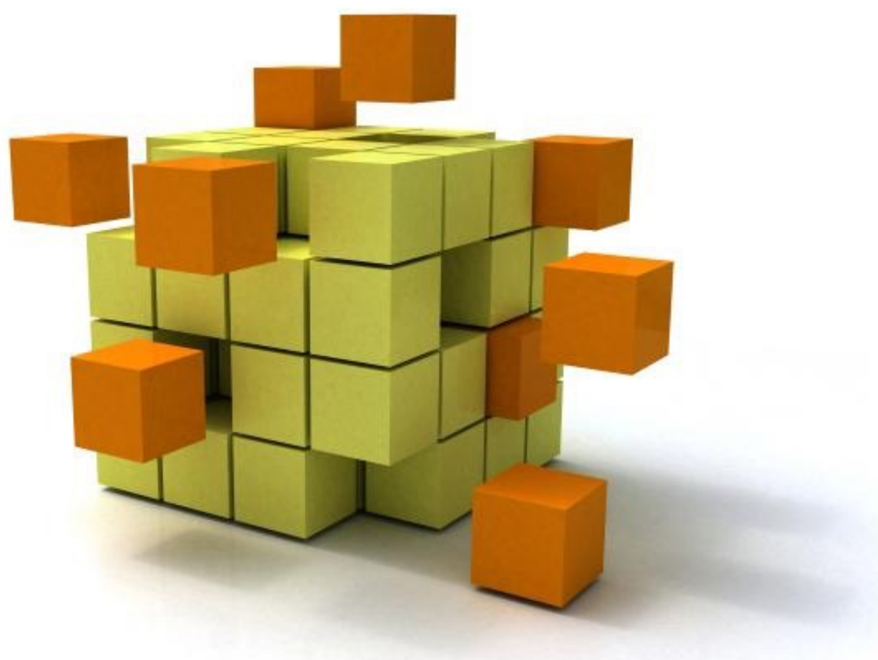


4- Classes et objets

- Encapsulation : visibilité des membres d'une classe

	+ public	- private
classe	La classe peut être utilisée par n'importe quelle classe	Utilisable uniquement par les classes définies à l'intérieur d'une autre classe. Une classe privée n'est utilisable que par sa classe englobante
attribut	Attribut accessible partout où sa classe est accessible. N'est pas recommandé du point de vue encapsulation	Attribut restreint à la classe où est faite la déclaration
méthode	Méthode accessible partout où sa classe est accessible.	Méthode accessible à l'intérieur de la définition de la classe

- Le statut « protected » sera vu dans l'héritage.



5- Enumérations

5- Les énumérations

– **Enumération** : ensemble de constantes

```
public enum Day {
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
   THURSDAY, FRIDAY, SATURDAY
}
```

– Autre exemple : les points cardinaux

```
public enum Pc {
  NORTH, EST, SOUTH, WEST
}
```

– Autre exemple : les planètes du système solaire

```
public enum Planet {
  MERCURY, EARTH, MARS, JUPITER, SATURN,
  URANUS, NEPTUNE
}
```

5- Les énumérations

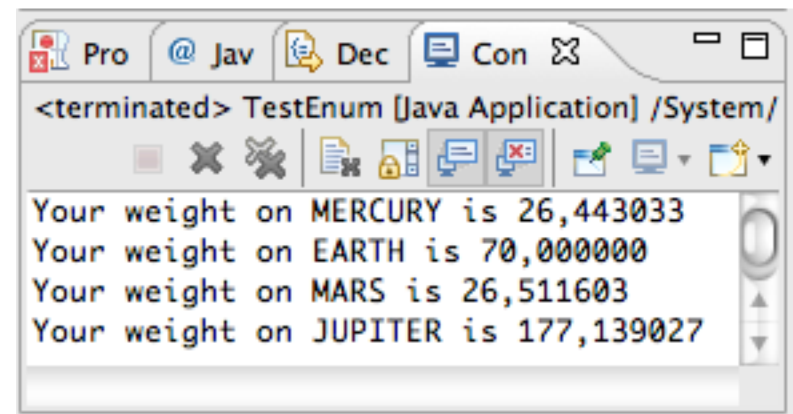
```
public class TellHowDayIs {
    Day day;
    public TellHowDayIs(Day d) {day = d; }
    public void tellItLikeItIs() {
        switch (day) {
            case MONDAY:
                System.out.println("Mondays are bad.");
                break;
            case SATURDAY:
            case SUNDAY:
                System.out.println("Weekends are best.");
                break;
            default:
                System.out.println("Midweek days
                                         are so-so.");
                break;
        }
}}
```

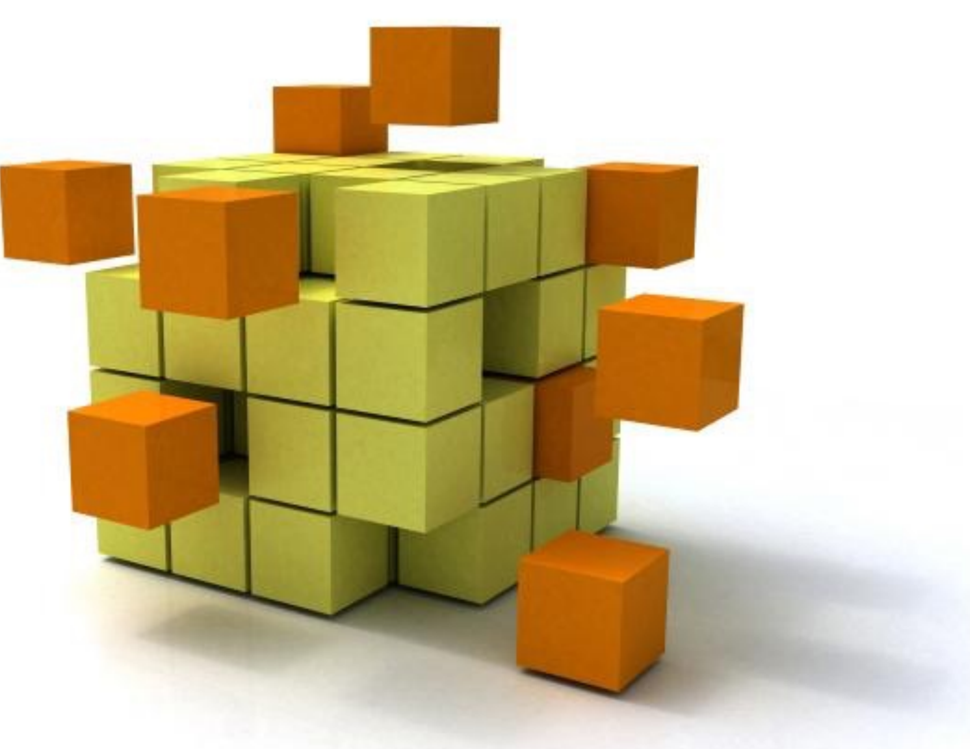
```
public class TestEnum {
   public static void main(String[] args) {
      TellHowDayIs firstDay =
         new TellHowDayIs(Day.MONDAY);
      firstDay.tellItLikeItIs();

      TellHowDayIs LastDay =
         new TellHowDayIs(Day.SUNDAY);
      LastDay.tellItLikeItIs();
   }
}
```

5- Les énumérations

```
public enum Planet {
   MERCURY (3.303e+23, 2.4397e6),
   EARTH   (5.976e+24, 6.37814e6),
   MARS    (6.421e+23, 3.3972e6),
   JUPITER (1.9e+27,   7.1492e7);

   private final double mass;
   private final double radius;
   Planet(double m, double r) {
      mass = m;
      radius = r;
   }
  // universal gravitational cst
  public static final double G = 6.673E-11;
  double surfaceGravity() {
      return G * mass / (radius * radius); }
  double surfaceWeight(double val) {
      return val* surfaceGravity(); }
}
```

```
public class TestEnum {
   public static void main(String[] args) {
      double earthWeight = 70;
      Planet Terre = Planet.EARTH;
      double mass = earthWeight
            /Terre.surfaceGravity();

      for (Planet p : Planet.values())
         System.out.printf("  weight on "
            + p.name() + " is "
            + p.surfaceWeight(mass)
            + "\n");    }
}
```

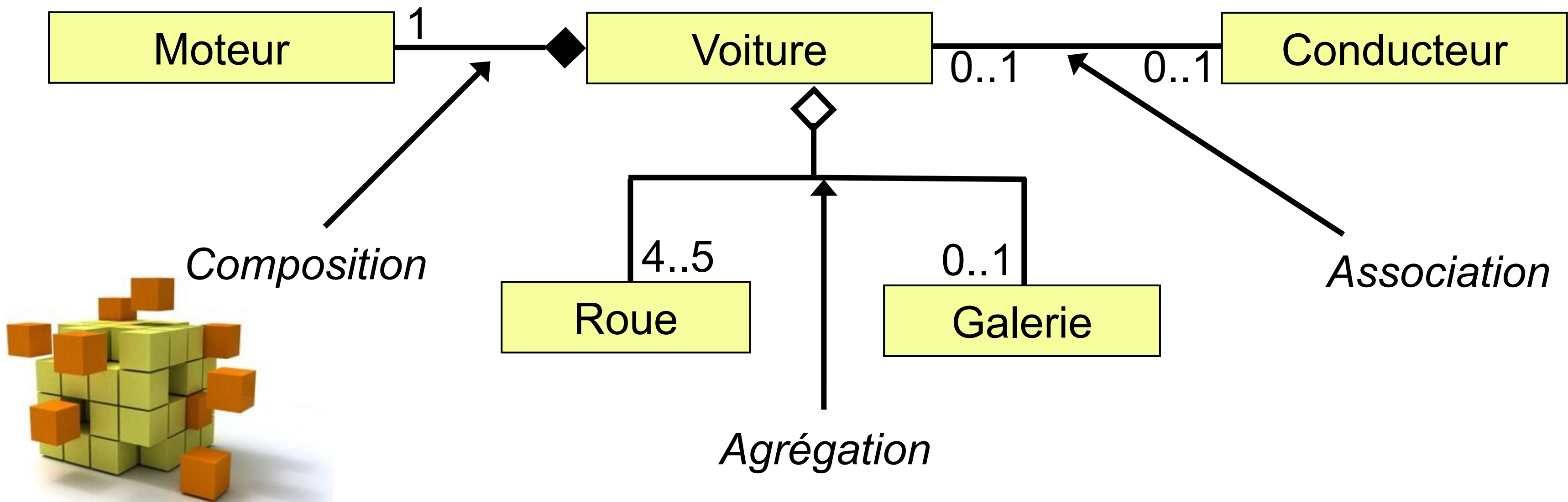




6- Composition

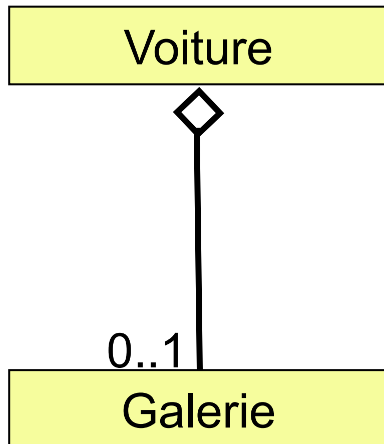
6- Composition

- Association : les objets sont sémantiquement liés
  - Exemple : une Voiture est conduite par un Conducteur
- Agrégation : cycles de vie indépendants
  - Exemple : une Voiture et une Galerie
- Composition : cycles de vie identiques
  - Exemple : voiture possède un moteur qui dure la vie de la voiture



6- Composition

- Agrégation : L'objet de la classe Galerie n’envoie pas de message à l’objet de classe Voiture




```
public class Voiture {
  private Galerie laGalerie;
  …
  public Voiture(Galerie g) {
    laGalerie = g;
    …
  }
  …
}
```

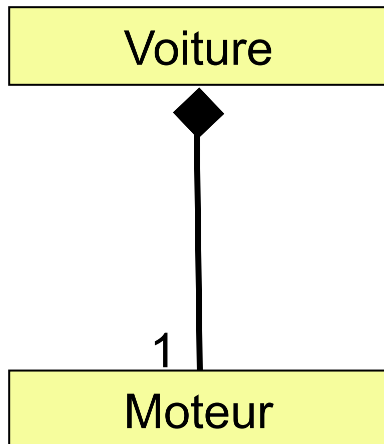
Attribut qui stocke une référence de galerie

Un objet Galerie transmis au moment de la construction

```
public class TestVoiture {
  public static void main (String[] argv) {
    Galerie maGalerie = new Galerie(…);
    Voiture maVoiture = new Voiture(maGalerie);
  }
}
```

6- Composition

- Composition (Solution 1) : Seule Voiture envoie des messages à Moteur




```
public class Voiture {
  private Moteur leMoteur;
  ...
  public Voiture(int p) {
    leMoteur = new Moteur(p);
    ...
  }
}
```

Attribut qui stocke une référence de moteur

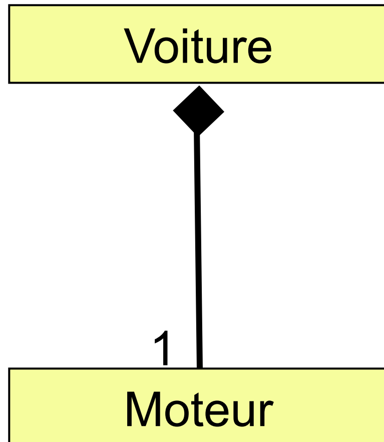
Création de l' objet moteur

```
public class Moteur {
  private int puissance;
  ...
  public Moteur(int p) {
    puissance = p;
    ...
  }
}
```

Attribut qui stocke la puissance

6- Composition

- **Composition (Solution 2)** : Les deux échangent des messages



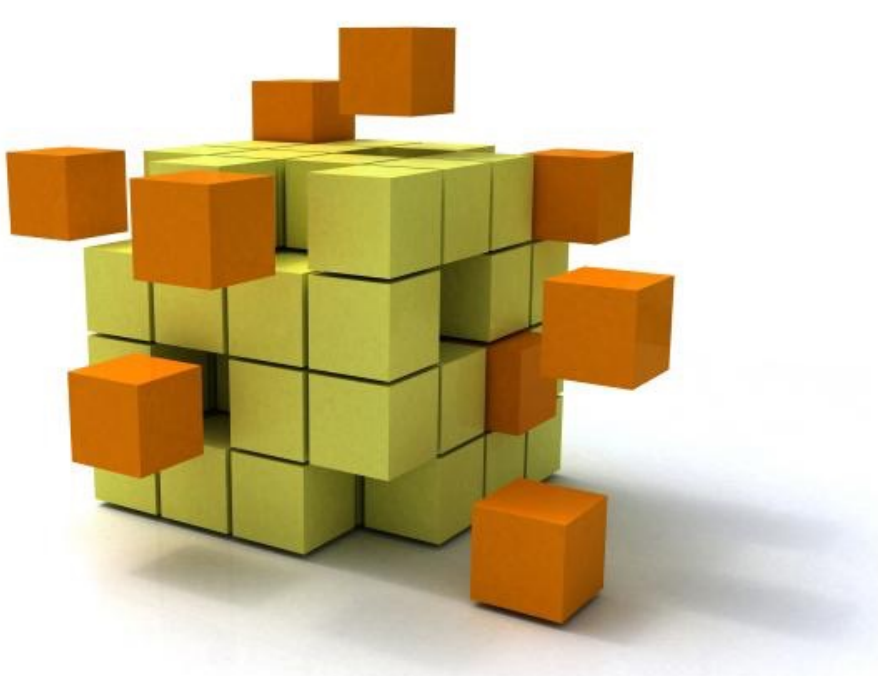

```java
public class Voiture {
   private Moteur leMoteur;
   public Voiture(int p) {
      leMoteur = new Moteur(p, this);
      ...
   }
}
```

Attribut qui stocke une référence de moteur

Transmission de la référence de l'objet courant

```java
public class Moteur {
   private int puissance;
   private Voiture laVoiture;
   public Moteur(int p, Voiture v) {
      puissance = p;
      laVoiture = v;
   }
}
```

Attribut qui stocke une référence de voiture.



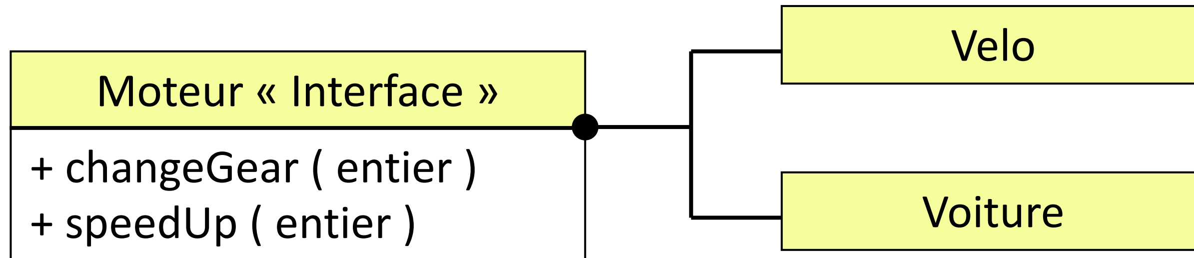
7- Les interfaces

7- Interface

- Les méthodes d’une classe forme son interface avec le monde extérieur, c’est à dire les autres classes et en particulier le programme principal.

- Une Interface forme un ensemble de méthodes vides (ce que l’on appelle la signature de la méthode). Elle est vue comme une contrat à respecter par la classe qui l’implémente.

Représentation UML



7- Interface

```
public interface Moteur{
    public void changeGear(int newValue);
    public void speedUp(int increment);
}
```

```
public class Voiture implements Moteur{
    private int puissance; …
    public Voiture (…) {
            …}
    public void changeGear(int newValue) {
            …}
    public void speedUp(int increment) {
            …}
}
```

```
public class Velo implements Moteur{
    private int nbRoues = 2;
    private String Proprietaire;
    public Velo (String P) {
            Proprietaire = p; }
    public void changeGear (int newValue) {
            … }
    public void speedUp (int increment) {
            … }
}
```

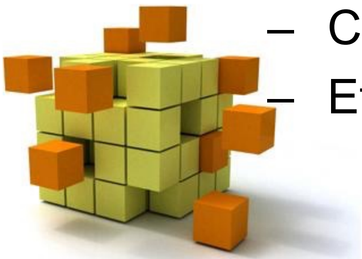
7- Interface

- Corps de l'Interface:
  - Contient des signatures de méthode
  - Peut contenir des constantes (*final*)
  - Peut contenir des implémentation de méthodes (mot-clé : *default*)
- Exemple : Comparer la taille d'objets quel qu'ils soient

```
public interface Relatable {
    public int isLargerThan(Relatable other);
}
```

Tous les objets provenant de classes qui implémentent *Relatable* pourront être comparés en taille (quel que soit la façon dont est calculée la taille). Exemples :

- Taille d'un livre: Taille = Nbre de pages
- Chaîne de caractères : Taille = Nombre de caractères
- Etudiant : Taille = taille en centimètres



7- Interface

- Exemple d'usage de *default*

```java
public interface Vehicle {
        String getBrand();
        String speedUp();
        String slowDown();
        default String turnAlarmOn() {
                return "Turning the vehicle alarm on.";
        }
}
```

Il existe aussi la possibilité d'utiliser des méthodes statiques das les interfaces…

7- Interface

Interface standard

- Le langage Java définit un certain nombre d'interfaces : *Cloneable*, *Serializable*, …
- Interface *Cloneable* pour dupliquer un objet (attention au rôle de l'opérateur = qui recopie les références) et garder la compatibilité avec les autres classes de Java.

Il faut alors implémenter la méthode public Object clone(), qui donne accès au service de clonage

```
public class Velo implements Moteur, Cloneable {
  …
  public Object clone() {
    Velo copie = new Velo( …);
    return copie;
  }
}
```

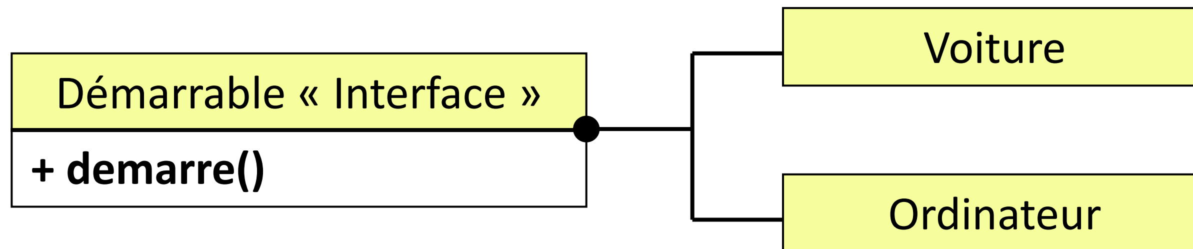
Object est la super-classe (héritage) de toute classe (on en parle plus tard).

7- Interface

Interface comme un type

- Une interface est un nouveau type. On peut ainsi définir une référence vers une interface et assigner cette référence vers un objet quelconque implémentant l'interface en question :

- Exemple : une Voiture et un Ordinateur sont des objets « Demarrable »




```
Demarrable unObj = new Voiture();
~~DemarrableunObj = new Demarrable();~~
```

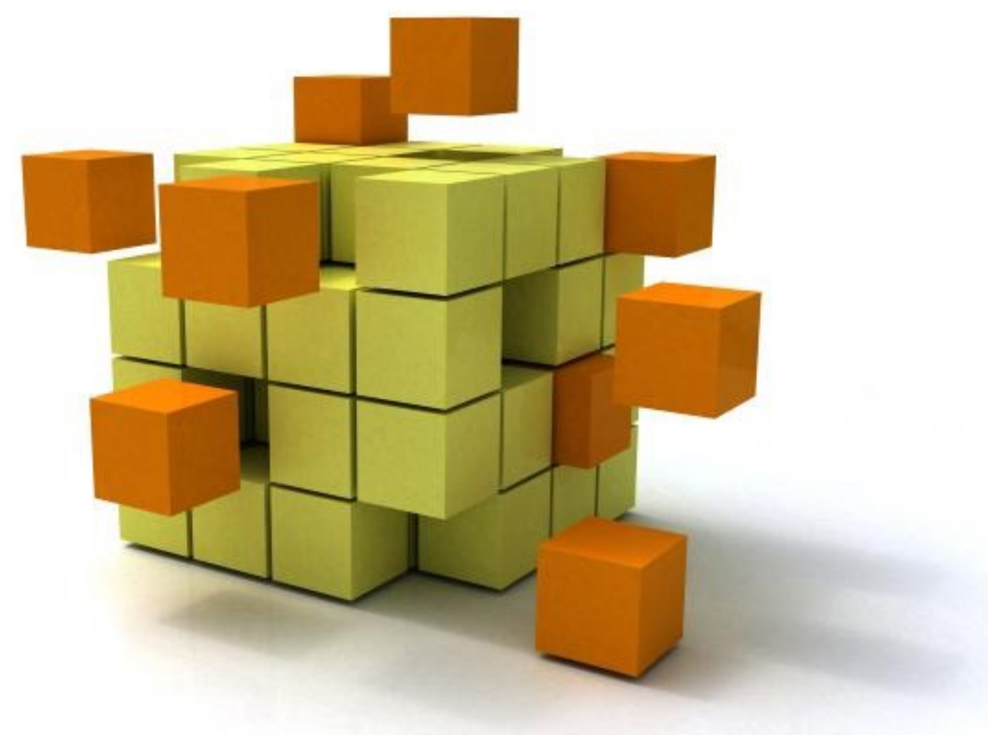
7- Interface

```
public class Person {
  private Demarrable objetDemarrable;
  public Person(Demarrable dem) {
    objetDemarrable = dem;
  }
  public void mettreEnRoute() {
    objetDemarrable.demarre();
}}
```

Une personne peut démarrer Voiture et Ordinateur sans connaître leur nature exacte

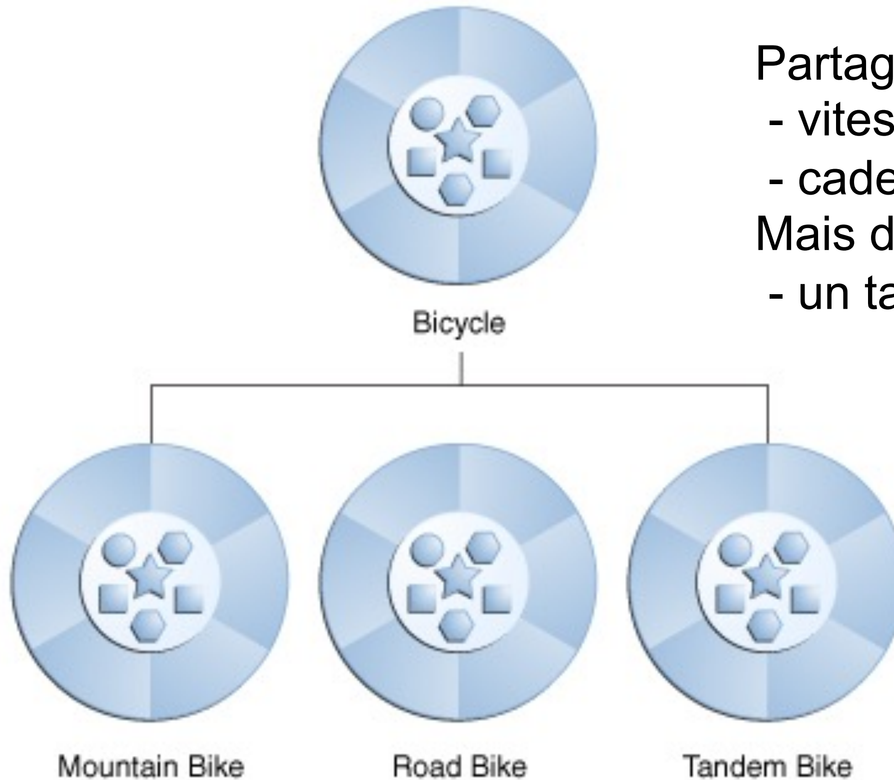
```
public class Test {
  public static void main (String[] argv) {
    Demarrable dem1 = new Voiture();
    Personne pers1 = new Personne(dem1);
    pers1.mettreEnRoute();
    Demarrable dem2 = new Ordinateur();
    Personne pers2 = new Personne(dem2);
    pers2.mettreEnRoute();
}}
```

Une personne peut démarrer tous les objets Demarrable



8- Héritage

8- Héritage



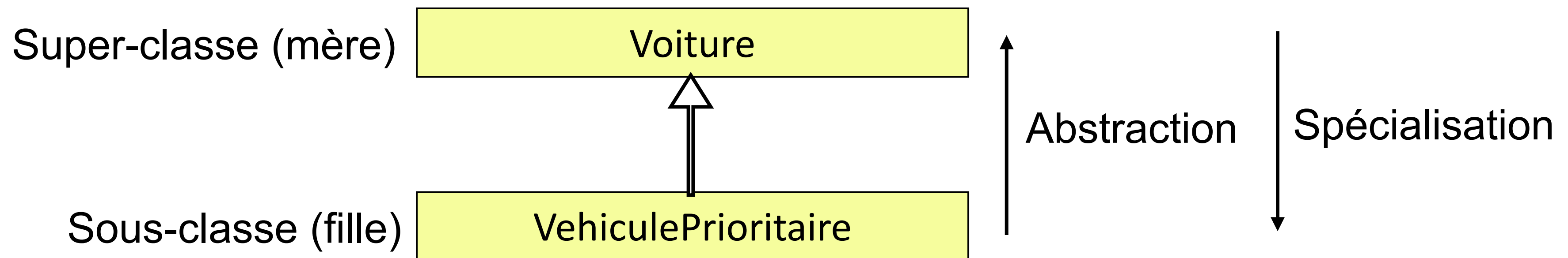
Partage de caractéristiques communes
- vitesse courant
- cadence des pédales…

Mais des différences aussi:
- un tandem à 2 selles…

```
class MountainBike extends Bicycle {

    // new fields and methods defining
    // a mountain bike would go here

}
```

8- Héritage

- La généralisation exprime une relation « **est-un** » entre une classe et sa super-classe



- L'héritage permet
  - de **généraliser** dans le sens abstraction
  - de **spécialiser** dans le sens raffinement

8- Héritage

Mammifère

Cétacé

Humain

Baleine

Dauphin

Etudiant

Enseignant

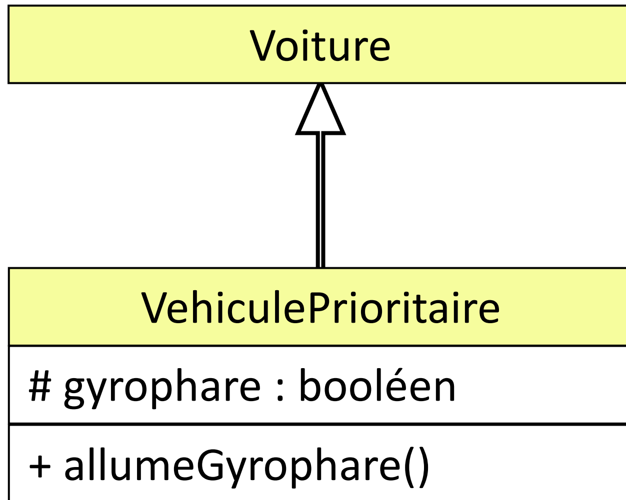
Mobi Dick

Ecco

Paul

George

8- Héritage




```
public class Voiture {
   protected boolean estDemarree;
   protected double vitesse;
   protected int puissance;
…
   public void demarre() {
      estDemarree = true; }
}
```

```
public class VehiculePrioritaire extends Voiture {
   protected boolean gyrophare;
   public void allumeGyrophare() {
      gyrophare = true; }
…}
```

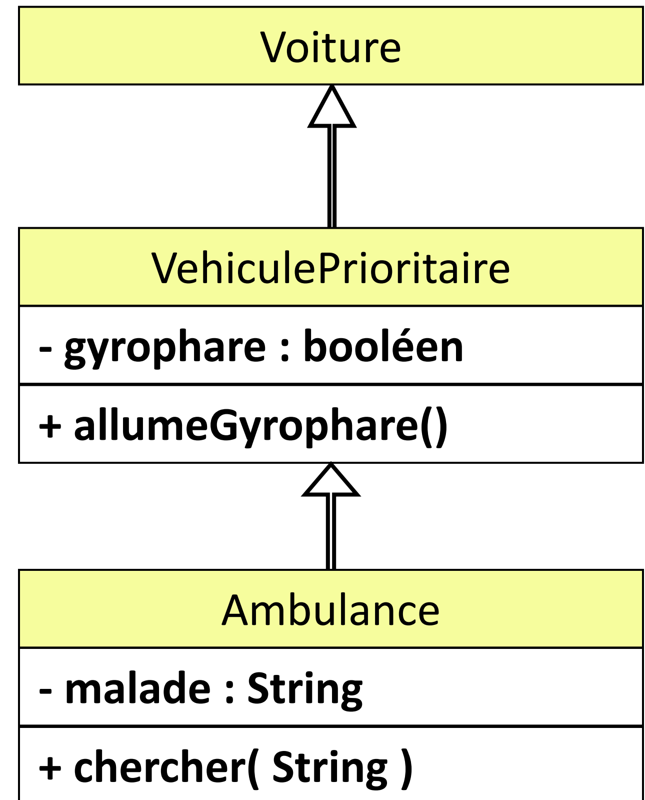
```
public class TestMaVoiture {
   public static void main (String[] argv) {
      VehiculePrioritaire VP= new VehiculePrioritaire (...);
      VP.demarre(); VP. allumeGyrophare();
   }
}
```

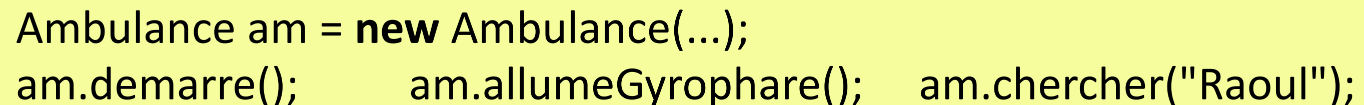
8- Héritage

```
public class Voiture {
   public void demarre() {
      ...
}}
```

```
public class VehiculePrioritaire extends Voiture {
   ...
   public void allumeGyrophare() {
      ...
}}
```

```
public class Ambulance extends VehiculePrioritaire {
   private String malade;
   public void chercher(String ma) {
      ...
}}
```

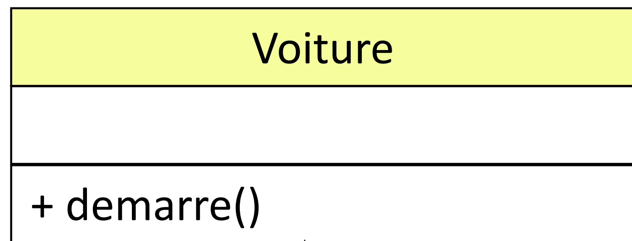


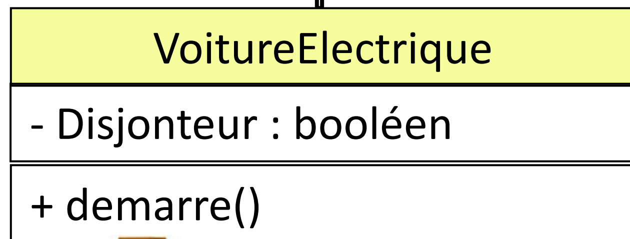


```
Ambulance am = new Ambulance(...);
am.demarre();        am.allumeGyrophare();    am.chercher("Raoul");
```

8- Héritage

- Redéfinition: une voiture électrique est une voiture dont l’opération de démarrage est différente
  - Une voiture électrique répond aux mêmes messages que la voiture
  - On démarre une voiture électrique en activant un disjoncteur

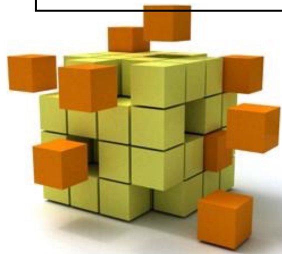




Redéfinition (masquage)

```
public class Voiture {
...
  public void demarre() {
    ...
}}
```

```
public class VoitureElectrique extends
Voiture {
  private boolean disjoncteur;
  public void demarre() {
    ...
  }
}
```



8- Héritage

- Super : Accès aux attributs et méthodes redéfinies de la classe mère à partir d'une classe fille

```
public class Voiture {
  private boolean estDemarree;
  public void demarre() {
    estDemarree = true;
  }
}
```

```
public class VoitureElectrique extends Voiture {
  private boolean disjoncteur;
  public void demarre() {
    disjoncteur = true;
    super.demarre();
}}
```

```
public class TestMaVoiture {
  public static void main (String[] argv) {
    VoitureElectrique laRochelle = new VoitureElectrique(...);
    laRochelle.demarre();
  }
}
```

- Dans un constructeur, on appelle le constructeur de la classe mère grâce à super(…). C'est nécessairement la 1ère ligne de code.

8- Héritage

- Toutes les classes de Java descendent de la classe Object
- Toutes vos classes descendent de Object (sans qu'il soit nécessaire de le préciser avec *extends*)

8- Héritage

Méthodes de la classe Object

- protected Object clone() throws CloneNotSupportedException
  Creates and returns a copy of this object.
- public boolean equals(Object obj)
  Indicates whether some other object is "equal to" this one.
- protected void finalize() throws Throwable
  Called by the garbage collector on an object when garbage collection determines that there are no more references to the object
- public final Class getClass()
  Returns the runtime class of an object.
- public int hashCode()
  Returns a hash code value for the object (ie. memory address).
- public String toString()
  Returns a string representation of the object.

+ Méthodes sur le multi-threading

8- Héritage

- Redéfinition de equals

```
boolean equals (Object o) {
  …
  return (this.hashCode() == o.hashCode());
}
```

```
public class HelloWorld {
    public static void main(String[] args) throws Exception {
        Voiture V1 = new Voiture (5);
        Voiture V2 = new Voiture (5);
        if (V1.equals(V2)) {
            System.out.println("Its true");
        }
        else {
            System.out.println("Its NOT true");
        }
    }
}
```

Résultat d'exécution : « false »

```
public class Voiture {
    int puissance;

    public Voiture (int p) {
        puissance = p;
    }
    boolean equals(Voiture o) {
        if (this == o)
            return true;
        return (puissance == o.puissance);
    }
}
```

Résultat d'exécution : « true »

8- Héritage

- Redéfinition de toString

```
public class Test {
  public static void main (String[] argv) {
    Voiture maVoiture = new Voiture(5);
    System.out.println(maVoiture);
 }
}
```

```
public String toString() {
  return (this.getClass().getName() + "@" + this.hashCode());
}
```

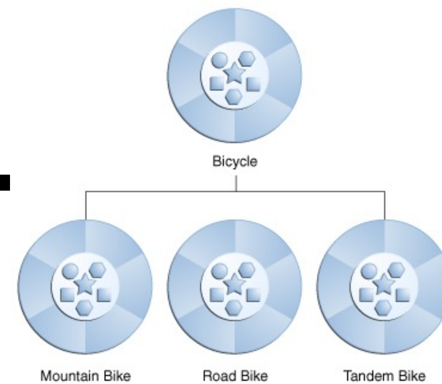
Résultats d'exécution : « Voiture@1087B6B »

Redéfinition

```
public class Voiture {
  ...
  public Voiture(int p) {
    this(p, new Galerie());
  }
  public String toString() {
    String H = "Puissance:" + p;
    return H;
}}
```

Résultats d'exécution :
« Puissance: 5 »

8- Héritage



Polymorphisme

```
MountainBike myBike = new MountainBike();
```

L'objet *myBike* est un *MountainBike*, un *Bicycle* et un *Object*.

L'inverse n'est pas nécessairement vrai : un *Bicycle* n'est pas nécessairement un *MountainBike*.

```
Object obj = new MountainBike();
```

L'objet *obj* est à la fois un *Object* et un *MountainBike*.

~~MountainBike myBike = obj;~~

Erreur au moment de la compilation car *obj* n'est pas connu comme un *MountainBike (implicit casting)*.

```
MountainBike myBike = (MoutainBike) obj;
```

On promet qu'à l'exécution l'*obj* sera un *MountainBike (explicit casting)* (si on ne tient pas sa promesse alors plantage prg.).

8- Héritage

- Mot-clé final : faire en sorte que l' on ne puisse pas hériter d'une classe en particulier

Héritage impossible

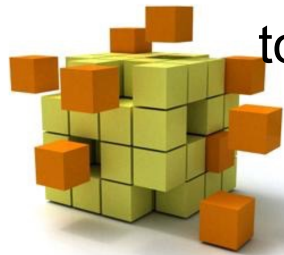
```
class Felin {
  protected boolean a_faim = true;
  void parler() { }
  void appeler() {
    System.out.println("minou minou,...");
    if (a_faim) parler();
  }}

final class Chat extends Felin {
  private String race;
  void parler() { System.out.println("miaou! "); }
}

final class Lion extends Felin {
  void parler() { System.out.println("roar! "); }
  void chasser() {......}
}
```

Remarque : « kif-kif » pour une méthode : on n'a pas le droit de la re-définir

8- Héritage

Mot-clé abstract :

- On ne connaît pas toujours le comportement par défaut d’une opération commune à plusieurs sous-classes
  - Exemple : On sait que toutes les décapotables peuvent ranger leur toit, mais le mécanisme est différent d’une décapotable à l’autre
  - Solution : on peut déclarer la méthode « abstraite » dans la classe mère et ne pas lui donner d’implémentation par défaut
- Méthode abstraite et conséquences : 3 règles à retenir
  - Si une seule des méthodes d’une classe est abstraite, alors la classe devient aussi abstraite
  - On ne peut pas instancier une classe abstraite car au moins une de ses méthodes n’a pas d’implémentation
  - Toute classe fille d’une classe mère abstraite doit implémenter toutes les méthodes abstraites, sinon elles sont aussi abstraites



```
public abstract class NomMaClasse { … }
```

```
public abstract void maMethode(...);
```

8- Héritage

- La classe VoitureDecapotable
  - Hérite de Voiture
  - Définit un attribut protégé toitReplié
  - Définit une méthode abstraite replieLeToit()
- Les classes DeuxChevaux et C3Pluriel héritent de VoitureDecapotable

8- Héritage

- Exemple de la VoitureDecapotable

```
public abstract class VoitureDecapotable
  extends Voiture {
    protected boolean toitReplié;
    public abstract void replieLeToit();
}
```

```
public class DeuxChevaux extends VoitureDecapotable {
  private boolean capoteAttachee;
  public void replieLeToit() {
      toitReplie = true;
      capoteAttachee = true;
  }
}
```

```
public class C3Pluriel extends VoitureDecapotable {
  private boolean arceauxRetirés;
  public void replieLeToit() {
      toitReplie = true;
      arceauxRetirés = true;
  }
}
```

Attention : ce n'est pas de la redéfinition. On parle d'implémentation de méthode abstraite

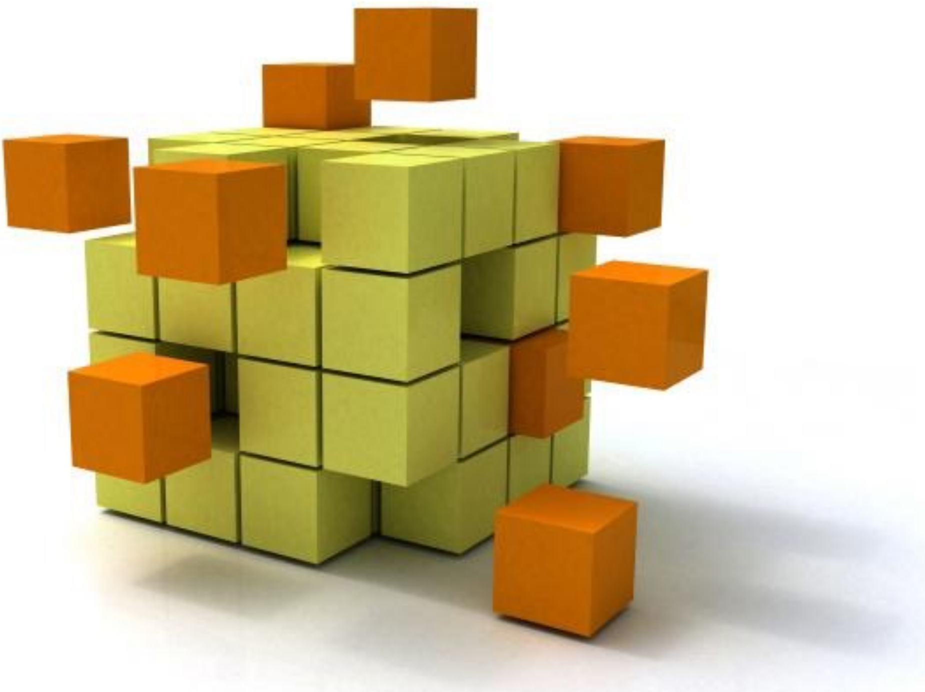
8- Héritage

```
public class Test {
   public static void main (String[] argv) {
      // Déclaration et création d'une DeuxCheveaux
      VoitureDecapotable V1 = new DeuxCheveaux(...);
      V1.replieLeToit();
      // Déclaration et création d'une C3Pluriel
      VoitureDecapotable V2 = new C3Pluriel(...);
      V2.replieLeToit();
      // Déclaration et création d'une VoitureDecapotable
      VoitureDecapotable V3 = new VoitureDecapotable(...);
   }
}
```

Attention : La classe VoitureDecapotable ne peut être instanciée puisqu'elle est abstraite

8- Héritage

- Les classes
  - Elles sont complètement implémentées
  - Une autre classe peut en hériter
- Les classes abstraites
  - Sont partiellement implémentées
  - Une autre classe non abstraite peut en hériter mais doit donner une implémentation aux méthodes abstraites
  - Ne peuvent pas être instanciées mais peuvent fournir un constructeur
- Les interfaces
  - Elles ne sont pas implémentées
  - Toute classe qui implémente une ou plusieurs interfaces doit implémenter toutes leurs méthodes (abstraites)
  - Elles peuvent s'hériter entre elles
- Les énumérations
  - Elles étendent toutes Enum
  - Elles peuvent s'hériter entre elles



9- Les exceptions

9- Les exceptions

- En cas d'erreur
  - Abandon du programme avec affichage d'un message
  - Retour à la méthode appelante avec un code d'erreur
  - Signalisation (levée d'exception) et déroutement de l'exécution du programme vers un code séparé (utilisation d'un gestionnaire d'exception)

- Les exceptions:
  - Séparation entre code normal et code de traitement d'erreur
  - Regroupement du traitement des erreurs relatives à une séquence de code
  - Mécanisme souple

9- Les exceptions standards

Les classes **Exception** en Java permettent de représenter une erreur par un objet transmis à un gestionnaire d'exception pour traitement

java.lang → Standard package

Object

Throwable

Errors

Exceptions

Runtime Exceptions

Other Exceptions

9- Les exceptions standards

Description des exceptions standards (héritées de RuntimeException) : https://www.tutorialspoint.com/java/java_builtin_exceptions.htm

	Exception & Description
1	**ArithmeticException** Arithmetic error, such as divide-by-zero.
2	**ArrayIndexOutOfBoundsException** Array index is out-of-bounds.
3	**ArrayStoreException** Assignment to an array element of an incompatible type.
4	**ClassCastException** Invalid cast.
5	**IllegalArgumentException** Illegal argument used to invoke a method.
8	**IndexOutOfBoundsException** Some type of index is out-of-bounds.

9- Les exceptions standards

Description des exceptions standards (héritées de RuntimeException) :
https://www.tutorialspoint.com/java/java_builtin_exceptions.htm

	Exception & Description
10	**NegativeArraySizeException** Array created with a negative size.
11	**NullPointerException** Invalid use of a null reference.
12	**NumberFormatException** Invalid conversion of a string to a numeric format.
13	**SecurityException** Attempt to violate security.
14	**StringIndexOutOfBounds** Attempt to index outside the bounds of a string.
15	**UnsupportedOperationException** An unsupported operation was encountered.

9- Les exceptions standards

Sans gestion d'une exception standard

```
public class Main {
    public static void main(String[] args) {
        int num[] = {0, 1, 2, 3};
        System.out.println(num[5]);
    }
}
```

> Task :run FAILED
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 4
        at testException1.Main.main(Main.java:5)

9- Les exceptions standards

Avec gestion d'une exception standard

```
public class Main {
   public static void main(String[] args) {
      try {
         int num[] = {0, 1 ,2, 3};
         System.out.println(num[5]);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception:" + e);
         System.out.println("getMessage:" + e.getMessage());
      }
   }
}
```

```
> Task :run
Exception:java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 4
getMessage:Index 5 out of bounds for length 4
```

9- Les exceptions standards

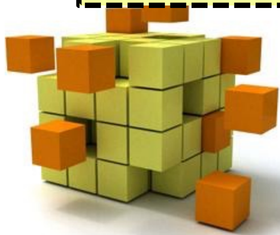
Avec gestion d'une exception standard

```java
public class Main {
    public static void main(String[] args) {
        try {
            int num[] = {0, 1 ,2, 3};
            //System.out.println(num[5]);
            System.out.println("Value:" + num[1]/num[0]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception:" + e);
            System.out.println("getMessage:" + e.getMessage());
        } catch (ArithmeticException ae) {
            System.out.println("Exception:" + ae);
            System.out.println("getMessage:" + ae.getMessage());
        }
    }
}
```

```
> Task :run
Exception:java.lang.ArithmeticException: / by zero
getMessage:/ by zero
```

9- Les exceptions standards

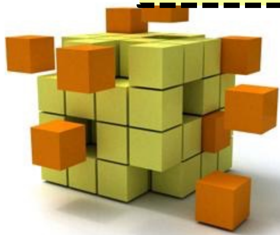
```
public class test {
    public static void main(String[] args) {
        byte[] b = new byte[80];
        int u;
        System.out.println("Entrez U(0) :");
        try {
            System.in.read(b);
            u = Integer.parseInt(new String(b).trim());
        }
        catch (IOException | NumberFormatException e) {
            System.out.println("Exception:" + e);
            System.out.println("--> Valeur par défaut : 10");
            u = 10;
        }
        System.out.println("Valeur de u=" + u);
    }
}
```



Multicatch !!!

9- Les exceptions : *user-defined*

```
public class CheckingAccount {
   private double balance;

   public void deposit(double amount) {
      balance += amount;
   }

   public void withdraw(double amount)
                   throws InsufficientFundsException {
      if(amount <= balance) {
         balance -= amount;
      } else {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }
}
```



Lancement de sa propre exception !

9- Les exceptions : *user-defined*

```
public class InsufficientFundsException extends Exception {
    private double amount;

    public InsufficientFundsException(double amount) {
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }
}
```

Classe qui hérite d'Exception

Constructeur qui stocke les "raisons de l'exceptions" (susceptible de l'expliquer)

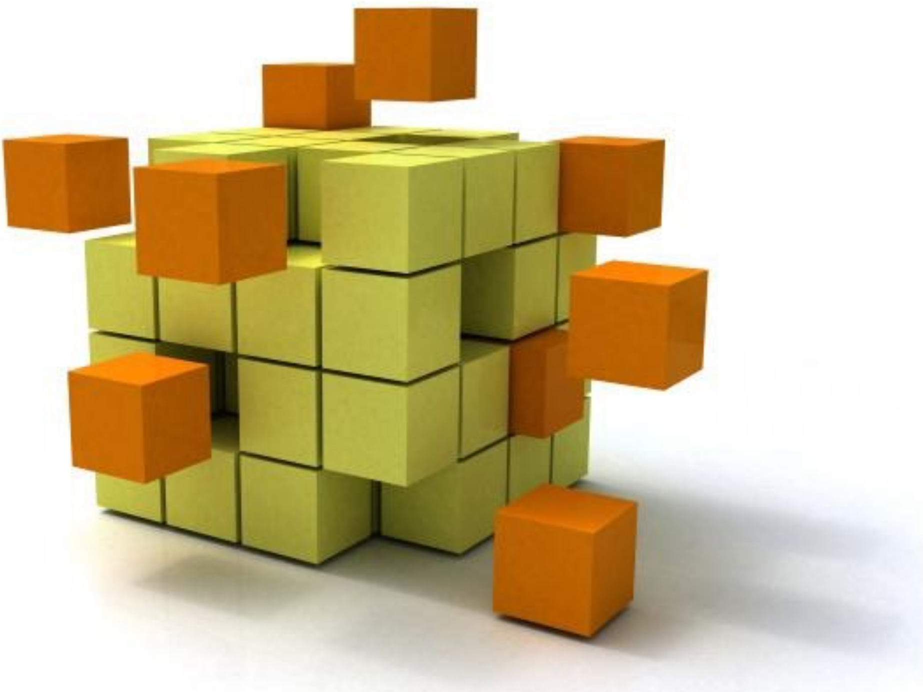
9- Les exceptions : *user-defined*

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

      CheckingAccount c = new CheckingAccount();
      System.out.println("Depositing $500...");
      c.deposit(500.00);

      try {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      } catch (InsufficientFundsException e) {
         Sys...ln("Sorry, but you are short $" + e.getAmount());
      }
   }
}
```

On attrape notre exception!

9- Les exceptions

```
public class TestExcemtions {
    public static void main(String[] args) {
        try { // séquence à surveiller
            ...
        } catch (Exc1 e) { // gestionnaire d'exception
            ... // traitement de l'exception de type Exc1
        } catch (Exception e) { // gestionnaire d'exception
            ... // traitement de l'exception de type Exception
        } finally {
            ... // traitement commun à tous les cas
                // (qu'il y ai ou non un exception d'attrapée)
                // Ex: fermer des fichiers, libérer des ressources…
        }
    }
}
```



10- Ce que l’on n’a pas vu…

10- Ce que l’on n’a pas vu…

- Les classes emboitées ou imbriquées

  nested and inner classes, anonymous class

- Les flux d’entrées / sorties (dont les fichiers). Encore que l’on a vu comment récupérer des valeurs fournies par l’utilisateur…

- Les classes (et les interfaces) génériques (vu comme exemple en TP2 et TP3 : ArrayList<Point>)

  Patrons de classe ou *template class* ou *parametrized class*

- Les décorateurs (meta-programmation) :

  Exemples : @Deprecated, @Override

  Objectif : Les décorateurs sont un moyen simple de modifier le comportement « par défaut » de fonctions. Les décorateurs sont des fonctions dont le rôle est de modifier le comportement par défaut d'autres fonctions ou classes.

10- Ce que l'on n'a pas vu…

- Les collections et les algorithmes standards

  ArrayList (vu!), Vector, LinkedList, HashSet, TreeSet, Queue and Deque.

- Les librairies spécifiques (API) :
  - AWT, Swing, SWT : interface graphique (prg évènementielle)
  - Thread : exécution simultanée de plusieurs processus
  - RMI : notamment pour le calcul distribué
  - JDBC : accès aux BDD relationnelles (SQL)
  - Comment créer ses propres packages (.jar)

- Les applets Java (*old-fashioned*) : programmes Java qui se téléchargent et s'exécutent sur une page Web.

  http://docs.oracle.com/javase/tutorial/deployment/applet/