utc
Université de Technologie
Compiègne

Rappel Java

Objectifs du cours

utc
Université de Technologie
Compiègne

- Apprendre la syntaxe du langage
- Pouvoir réaliser des applications et des applets en Java
- Savoir choisir les technologies adaptées et mettre en place des interfaces efficaces

Plan

- Présentation
- Syntaxe du langage
- Spécification de la plateforme Java
- Programmation Orientée Objet
- Déclaration d'attributs et de méthodes
- Agrégation et Encapsulation
- Héritage et Polymorphisme
- Exceptions
- Classes essentielles
- Collections
- Entrées/Sorties

Présentation

Historique

- 90th : James Grosling
- Première version : **OAK**
- Nouvelle version de Java avec Bill Joy
- Naissance de Java avec l'évolution du Web
- 1994 : Intégration des applets
- Intégration à Netscape
- 2006 : Java open source

Caractéristiques

utc
Université de Technologie
Compiègne

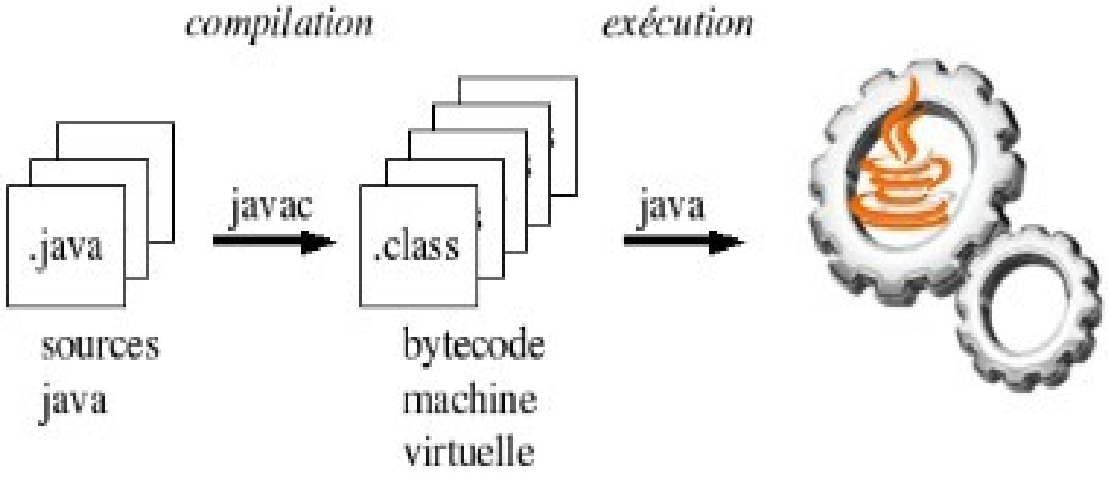
- Simple, familier et orienté objet
- Distribué
- Interprété
- Robuste et sûr
- Portable et indépendant des plates-formes
- Dynamique et multithread

Développement Java

Ecrire un programme Java ne nécessite pas d'outils sophistiqués :

- Un éditeur de texte tel notepad suffit.

- Et un JDK (Java Development Kit).
  (http://java.sun.com/javase/downloads/index.jsp)



Kit de développement Java

Ce kit de développement comprend de nombreux outils :

- le compilateur : ***javac***
- l'interpréteur d'application : ***java***
- le débogueur : ***jdb***
- le générateur de documentation : ***javadoc***

etc.

Packages Java (API)

java.lang : classes fondamentales

java.io : Entrées/sorties, gestion des flux, sérialization...

java.util : Collections framework

java.awt : Abstract Windowing Toolkit (IHM).

java.security : framework de Securité

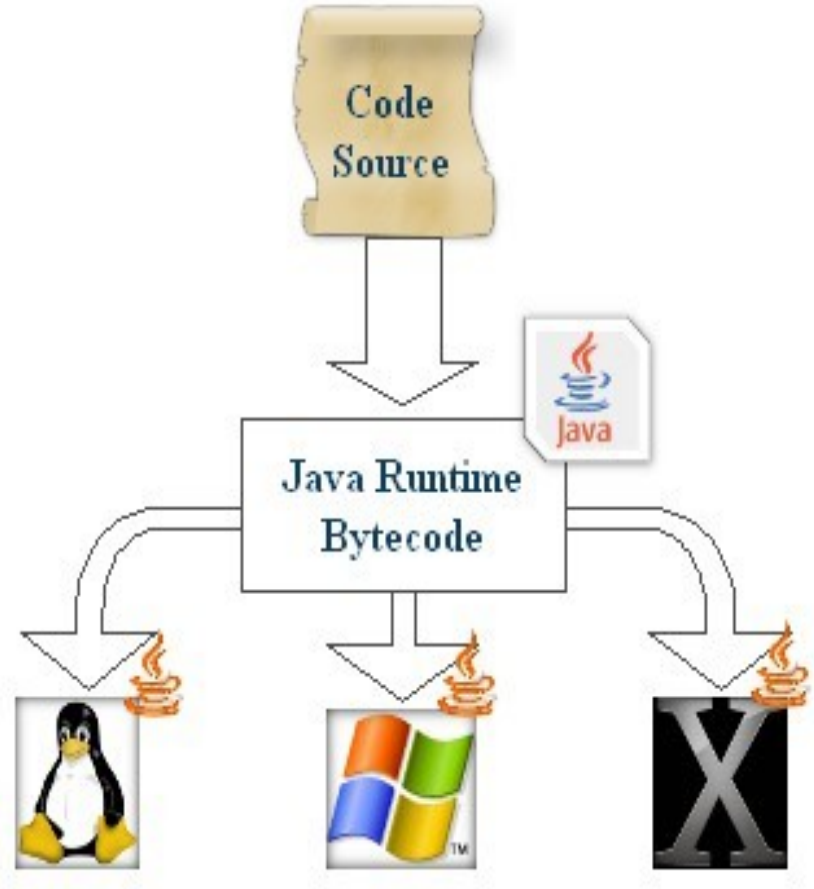
java.net : Applications Réseau

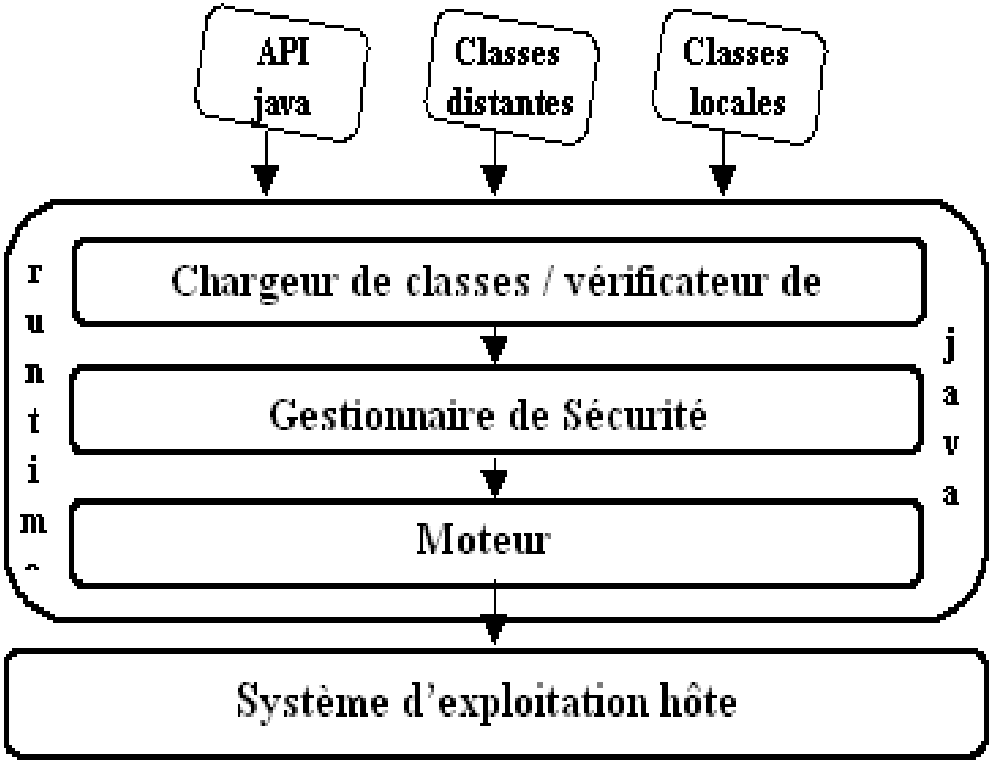
javax.swing : Composants Graphiques

java.text : Textes, dates, nombres et messages

java.sql : Accès aux données stockées dans des SD

Java Virtual Machine





Environnements de développement

Hormis l'outil de base de développement (***JDK***), il existe des environnements de développement intégrés (IDE) :

- ECLIPSE (Eclipse.org)
- NetBeans (Apache)
- Visual Studio (Microsoft)
- Jdeveloper (Oracle)
- IntelliJ IDEA

Première Application Java

- Installation d'un JDK
- Paramétrages des variables d'environnement
- Installation d'Eclipse (Eclipse.org)
- Ecriture et exécution d'un premier programme en Java :
  - Minuscules / majuscules différentiées
  - Espaces / CR / LF / Tabulations sans conséquences

```
public class BonjourMonde {

   // Définition de la méthode principale
   public static void main(String[]args){
      System.out.println("Bonjour tout le monde !");
   }
}
```

Syntaxe du langage

Types de données

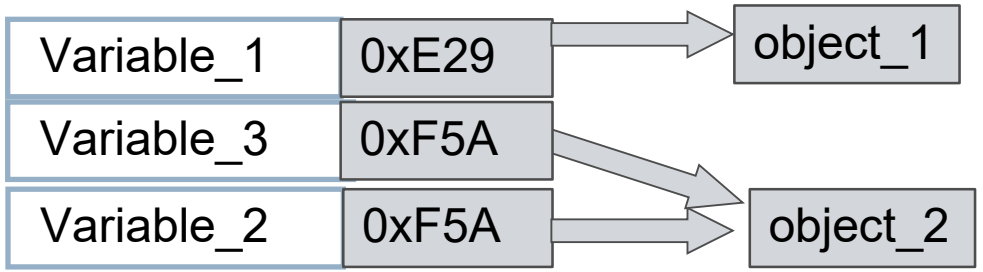
- Types Primitifs / Types Références
- Déclaration et initialisation
- Nombres
- Caractères
- Conversion
- Objets
- Wrappers

Types de données

- **Types Primitifs**

Variable_1	45
Variable_2	true
Variable_3	c
Variable_4	56.8

- **Types Références**



Types de données

Déclaration de variables :

```
int toto;
double t1, t2;
boolean test;
```

Initialisation pendant la déclaration :

```
int toto = 10;
double t1 = 1.25, t2 = 1.26;
boolean test = true;
```

Types de données

- Changement de type (**typecasting**) :

  - Transtypage implicite (automatique)
    type inférieur => type supérieur

  - Transtypage explicite (cast) :
    type supérieur => type inférieur.

```
double x = 3.5;
int y = (int) x;
```

Types de données - objets

Il existe des types complexes :

- Tableaux
- String (chaîne de caractère)
- Integer (encapsulation d'un entier)
- …

L'identifiant dans la pile contient l'adresse mémoire (virtuelle) de l'objet.

Types de données

- Wrappers (types enveloppes)

Wrappers sont des objets identifiants des variables primitives

Primitive type	*Wrapper*
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

```
int i = 3;
Integer integ = new
   Integer(i);

int j =
   Integer.parseInt("10");
```

```
char c = 'a';
Character ch = new
   Character(c);
```

Opérateurs

- **Opérateurs logiques :**
  ! | & ^ || &&

- **Opérateurs de comparaison :**
  < > == >= <= !=

- **Opérateurs mathématiques :**
  +/- ++/-- * / %

Méthodes

- Déclaration d'une méthode :

```
<type de retour> <nom de la méthode>(<arguments>...)
{
      <instructions>...
}
```

```
int somme(int a, int b){
    return a+b;
}
```

- Appel d'une méthode :

```
somme(2,3);
```

Instructions

- se terminent par un « ; »
- peut contenir une affectation, une déclaration ou un appel de méthode ...

```
int x = 10;       // Décl. Et Affectation
x = (x+30)/2;     // Opération
setName("John"); // Appel de méthodes
```

- **Conditions :**

```
if (condition) {
        // bloc d'instructions1
} else {
        // bloc d'instructions2
}
```

Instructions

- **Conditions : « switch »**

```
int jours = 7;
switch (jours) {
      case 1: System.out.println("Lundi");
              break;
      case 2: System.out.println("Mardi");
              break;

      default: System.out.println("week end !");
}
```

- **Conditions : « Opérateur Ternaire » (if/else)**

```
( condition ? condition_true : condition_false)
```

```
int j = (i > 2 ? i : 0);
//j=i if i est supérieur à 2, else j=0
```

Instructions

- **Boucles : « while »**

Tant que la condition est vérifiée, j'exécute le bloc d'instructions

```
while (condition) {instructions}
```

```
int i = 10, somme = 0, j = 0;
while (j <= i) {
        somme = somme + j;
        j = j + 1;
}
System.out.println("Somme= "  + somme);
```

Instructions

- **Boucles : « do/while »**

```
do {instructions} while (condition);
```

```
int i = 10; int j = 0; int somme = 0;
do {
        somme = somme + j;
        j = j + 1;
} while (j <= i);
System.out.println("Somme = " + somme) ;
```

Instructions

- **Boucles : « for »**

```
for ([initialisation]; [condition]; [operation]){}
```

```
for (int i = 0;  i < 10; i = i + 1)
{
        maMethode(i);
        System.out.println(i) ;
}
```

- **Toutes les boucles : *break* (sortie), *continue* (suite)**

Tableaux

- Déclaration d'un tableau :

```
type[] nom = new type [taille] ;
type[] nom = {valeur1, valeur2,..} ;
```

- Valeur d'un élément du tableau :

```
nom [indice]
```

- Taille d'un tableau : *length*

```
int n = nom.length;
```

Tableaux (suite)

- Tableau à 2 dimensions :

```
type[][] nom = new type [taille][] ;
nom[0] = new type [taille2];
...
```

- Itération complète :

```
for(type n : nom) {
...
}
```

Méthode main

- Point d'entrée du programme
- Doit être statique
- Peut recevoir des paramètres depuis une ligne de commande

```
java MonJavaProgram Hello!
```

Exercices

- Déclarer un tableau de 10 nombres.
- Ecrire des méthodes qui calculent le *minimum*, le *maximum*, la *moyenne* et l'*écart type* des valeurs de ce tableau.
- Tester ces méthodes.

Programmation Orientée Objet

Définition

L'orienté-objet = approche de résolution algorithmique de problèmes permettant de produire des programmes modulaires de qualité..

Objectifs :

- développer une partie d'un programme sans qu'il soit nécessaire de connaître les détails internes aux autres parties;

- Apporter des modifications locales à un module, sans que cela affecte le reste du programme;

- Réutiliser des fragments de code développés dans un cadre différent.

Qu'est ce qu'un objet ?

utc
Université de Technologie
Compiègne

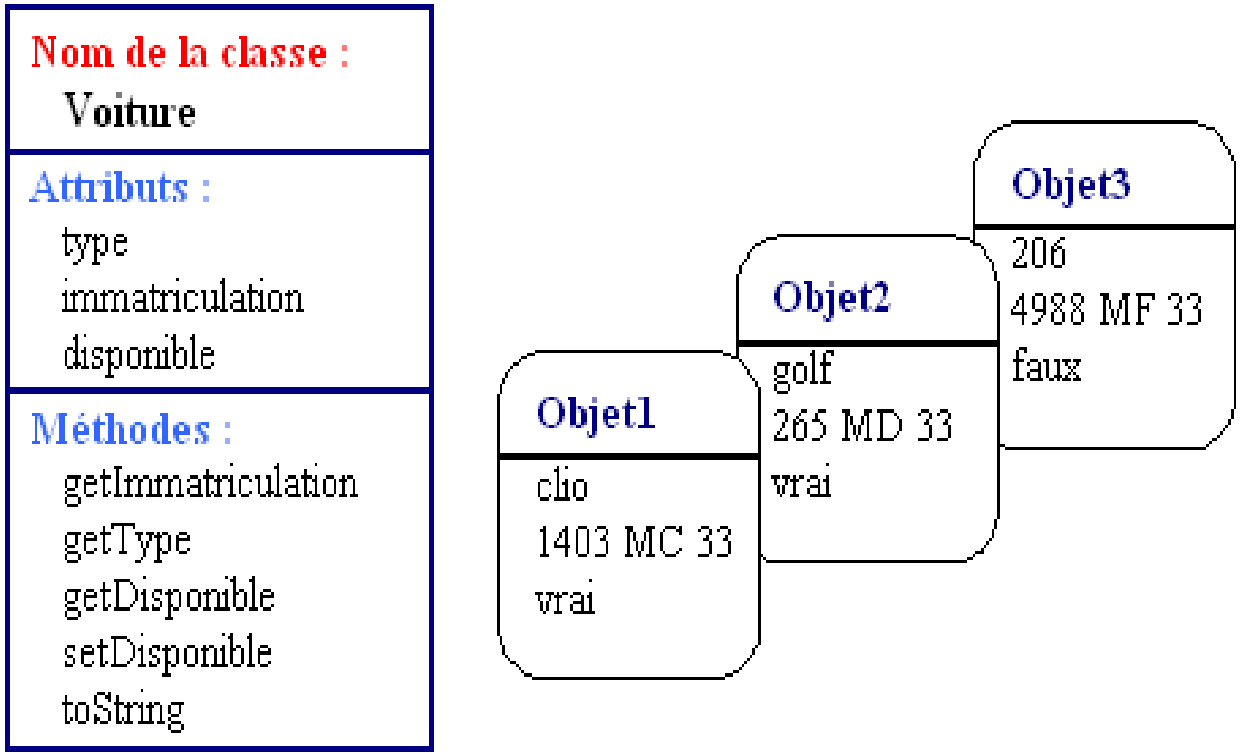
Objet = élément identifiable du monde réel, soit concret (voiture, stylo,...), soit abstrait (entreprise, temps,...)

Un objet est caractérisé par :

- son état (les données de l'objet)
- son comportement (opérations : ce qu'il sait faire)

Qu'est-ce qu'une Classe ?

- Une classe est un type de structure ayant des attributs et des méthodes.
- On peut construire plusieures **instances** d'une classe.



Déclaration d'une Classe

```
public class Voiture {

    // Attributs
      …

   // Méthodes
      …
}
```

Méthode principale (main)

- La méthode **`main`** représente le point d'entrée d'une application en exécution.

- Elle peut être intégrée dans une classe existante ou écrite dans une classe séparée.

```
public class Launch {

	// Méthode principale
	public static void main (String [] args) {
	// Création d'une instance de la classe Voiture
		Voiture Clio = new Voiture();
	}
}
```

Packages ou Espaces de noms

Package = groupement de classes qui traitent un même problème pour former des « bibliothèques de classes ».

Une classe appartient à un package s'il existe une ligne au début renseignant cette option :

```
package nompackage;
```

Pour utiliser une classe (au choix) :
- Etre dans le même package
- Préfixer par le package (à chaque utilisation)
- Au début du fichier, importer la classe, ou le package entier

```
import nompackage.*;
```

Déclaration d'attributs et de méthodes

Variables d'instance

Définissent l'état de l'objet :
- On les appelle également « attributs ».
- La valeur d'un attribut est propre à chaque instance.

```
public class Voiture {
    String marque;
    String plaque;
    String couleur;

    …
}
```

- L'appel de ses variables se fait comme suit :
  monInstance.monAttribut;

ex : clio.couleur;

Variables locales

- Variables temporaires qui existent seulement pendant l'exécution de la méthode.

```
public class MaClasse {
  // …
        public void maMethode() {
                int monNombre = 10;
        }
        public void maMethode2(){
                System.out.println(monNombre);
                // Erreur de compilation
        }
}
```

Méthodes d'instances

Bloc d'instructions définissant un comportement d'une instance.
- déclarées à l'intérieur d'une classe.
- peuvent être surchargées (même nom, différents paramètres,...)

```
public class MaClasse {
  // …
        public void maMethode() {
                // les actions
        }
}
```

L'appel de ses méthodes :
monInstance.maMethode();

Constructeur

- Le constructeur est une méthode spéciale dans la classe appelée à la création d'instances.
  - si on ne définit pas de constructeur, le compilateur en créera un par défaut).
- Un constructeur porte le nom de la classe.

```
public class Voiture {
    public Voiture() { //pas de void
        // Corps du constructeur
     }
    …
}
```

- Il peut aussi y avoir un destructeur, appelé aussi automatiquement (`public void finalize(){...}`)

Constructeurs multiples

Il est possible de déclarer plusieurs constructeurs différents pour une même classe afin de permettre plusieurs manières d'initialiser un objet. Les constructeurs diffèrent alors par leur signature.

```
public class Voiture {

    // Contructeur par défaut sans paramètres
    public Voiture() {
        // Corps du contructeur
     }

    // Contructeur avec un paramètre
    public Voiture(String couleur) {
        // Corps du contructeur
     }
    …
}
```

Exercices

- Créez une classe Voiture.

- Ajoutez un constructeur avec comme paramètres sa marque et sa plaque d'immatriculation.

- Ajoutez un constructeur avec comme paramètres sa marque, sa plaque d'immatriculation et sa couleur.

- Ecrivez une méthode principale permettant de créer deux objets avec les différents constructeurs.

Variables de classes

Variables partagées par toutes les instances de classe.
- Variables déclarées avec le mot clé **static**
- Pas besoin d'instancier la classe pour utiliser ses variables statiques (**static**).
- Chaque objet détient la même valeur de cette variable.

```
public class Voiture {
    String type;
    ...
    static int nbVoitures;

    …
}
```

L'appel de ses variables : **maClasse.maVarStatic;**
exple : Voiture.nbVoiture;

Méthodes de classes

Bloc d'instructions définissant un comportement global ou un service particulier.
- Déclarées avec le mot clé **static**
- Peuvent être surchargées (même nom, ≠ paramètres).

```
public class MaClasse {
  // …
        public static void maMethode() {
                // les actions
        }
}
```

- N'utilisent pas de variables d'instances parce qu'elles doivent être appelées depuis la classe.
L'appel de ses méthodes : **MaClasse.maMethode;**

Le mot clé « this »

- Fait référence à l'objet en cours
- On peut l'utiliser pour :
  - Manipuler l'objet en cours

```
maMethode(this);
```

  - Faire référence à une variable d'instance

```
this.maVariable;
```

  - Faire appel au constructeur propre de la classe

```
this("w44","BMW");
```

Exercices

- Ajout d'une variable statique « Nbre de voitures » à la classe Voiture.

- Cette variable devra être incrémentée à chaque instanciation de la classe

- Utilisation du mot clé *this* à l'intérieur du constructeur pour changer la valeur de ses attributs

Agrégation et encapsulation

Agrégation et Accessibilité

- **Agrégation** = associer un objet avec un autre
  ex : Objet Propriétaire à l'intérieur de la classe Voiture

- **Accessibilité :** utilisation de facteurs de visibilité

  public:
  - Accessible par toutes les classes

  protected:
  - Accessible par toutes les sous-classes et les classes du même package

  "nothing" ou **default**:
  - Accessible seulement par les classes du même package.

  private:
  - Accessible seulement dans la classe elle-même

Encapsulation

Encapsulation =

- Regroupement de code et de données.
- Masquage d'information par l'utilisation d'accesseurs (getters et les setters) afin d'ajouter du contrôle .

L'encapsulation permet de restreindre les accès aux membres d'un objet, obligeant ainsi l'utilisation des membres exposés.

Qu'est-ce qu'un JavaBean ?

Un type de classe ayant :

- un constructeur public sans arguments
  (et éventuellement d'autres constructeurs)
- des getters et setters pour chaque attribut
- Une moyen de sérialisation
  (généralement, implémente java.io.Serializable)

Exercices

- Implémentation d'une agrégation :
  Voiture – Propriétaire - Adresse

- Ajout de getters et setters.

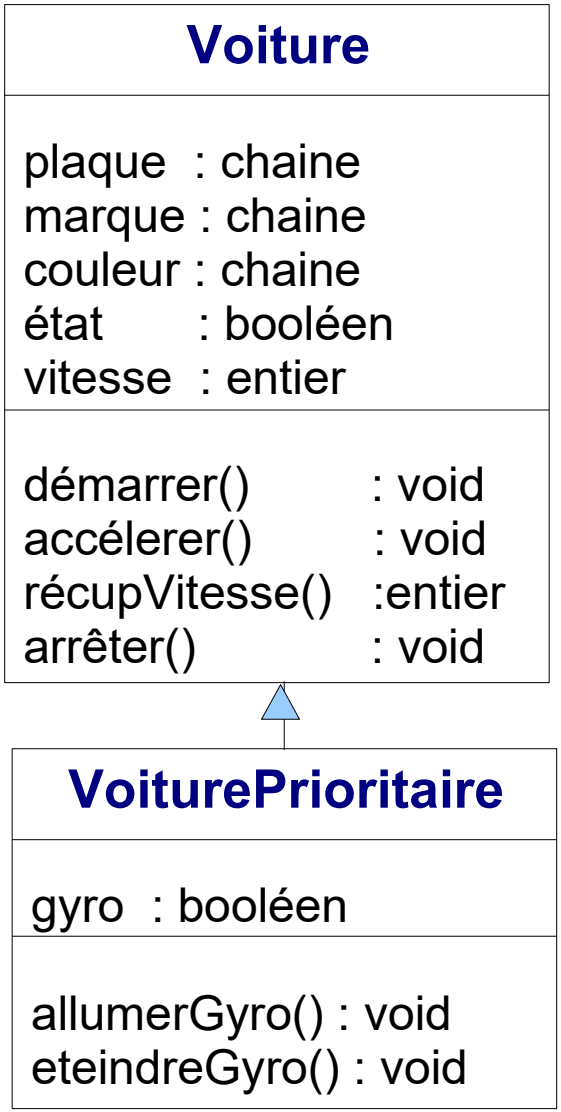
- Création d'instances de la classe Voiture

Héritage

Héritage

- L'héritage permet de créer la structure d'une classe à partir des membres d'une autre classe.

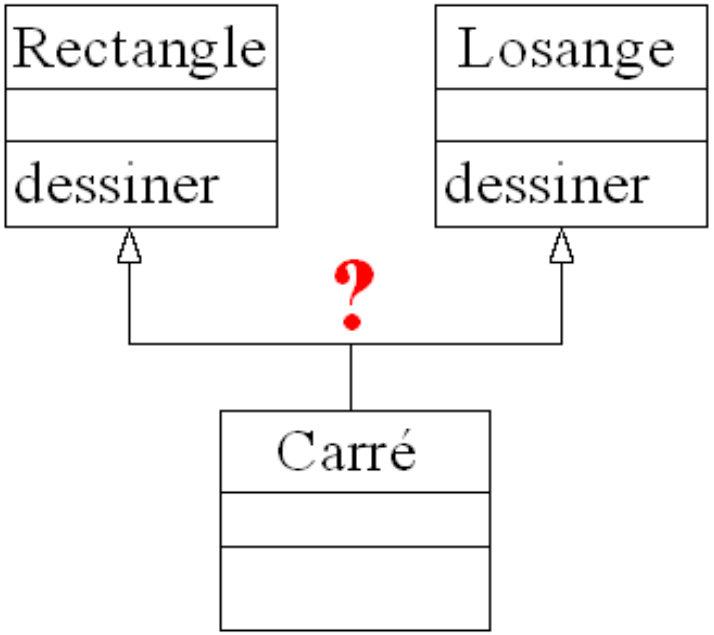
- La sous-classe hérite de tous les attributs et méthodes de sa classe mère.



Héritage multiple

L'héritage multiple permet à une classe d'hériter simultanément de plusieurs autres classes.

Problème :



Non disponible dans Java.

L'héritage multiple peut être partiellement comblé par une approche d'héritage en cascade des classes.

Héritage en Java

- Utilisation du mot clé **extends**.

```
public class VoiturePrioritaire extends Voiture{
 private boolean gyrode;
              // les actions
}
```

- Utilisation du mot clé **super** (référence à la classe mère)

```
private void demarrer() {
   gyrode = true;
   super.demarrer();
 }
```

Surcharge et redéfinition

La **redéfinition** (***overriding***) consiste à réimplémenter une version spécialisée d'une méthode héritée d'une classe mère (les signatures des méthodes dans la classe mère et la classe fille doivent être identiques).

La **surcharge** (***overloading***) consiste à proposer, au sein d'une même classe, **plusieurs « versions » d'une même méthode** qui diffèrent simplement par le nombre et le type des arguments (le nom et le type de retour doivent être identiques).

Utilisation de « final »

- Le mot clé **final** permet :

  - d'interdire l'héritage à partir de la classe.

  - d'interdire la redéfinition d'une méthode

  - de déclarer une constante

```
public final class C1 { …
}

public final void M1() { …
}

public static final int X = 3;
```

POO avancée

Polymorphisme

Le polymorphisme est la propriété d'une entité de pouvoir se présenter sous diverses formes. Ce mécanisme permet de faire collaborer des objets entre eux sans que ces derniers aient déclarés leur type exact.

Exemples :

- On peut avoir une voiture prioritaire avec le type Voiture

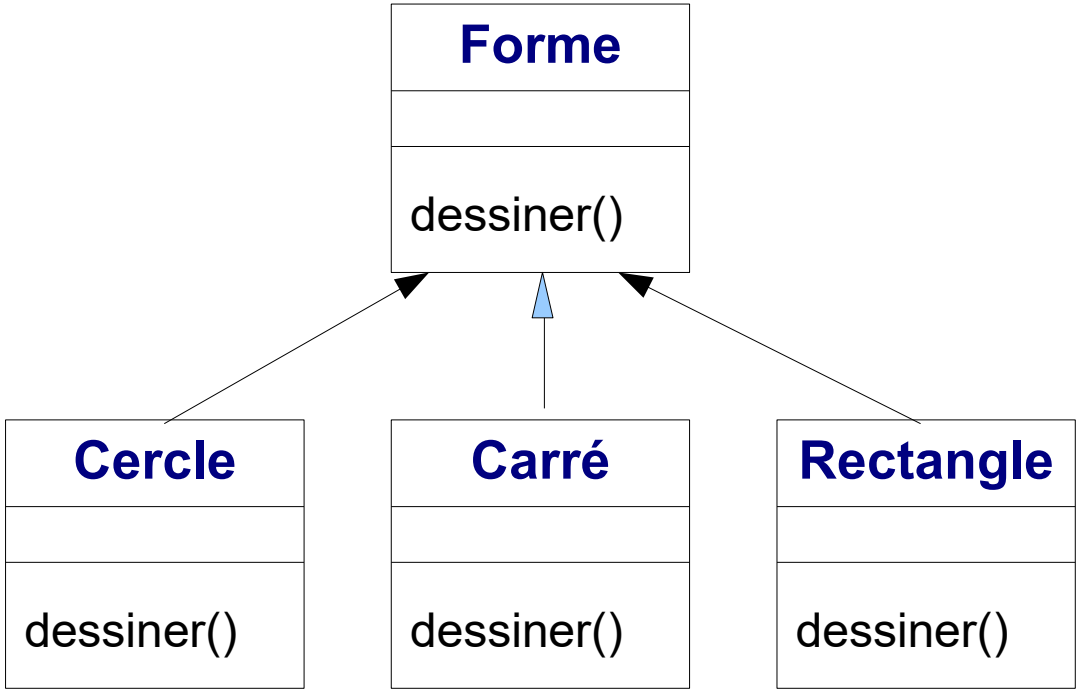
- On peut créer un tableau de Voitures et placer à l'intérieur des objets de type Voiture et d'autres de type VoiturePrioritaire

Classe Abstraite

Une classe qui ne peut être instanciée.

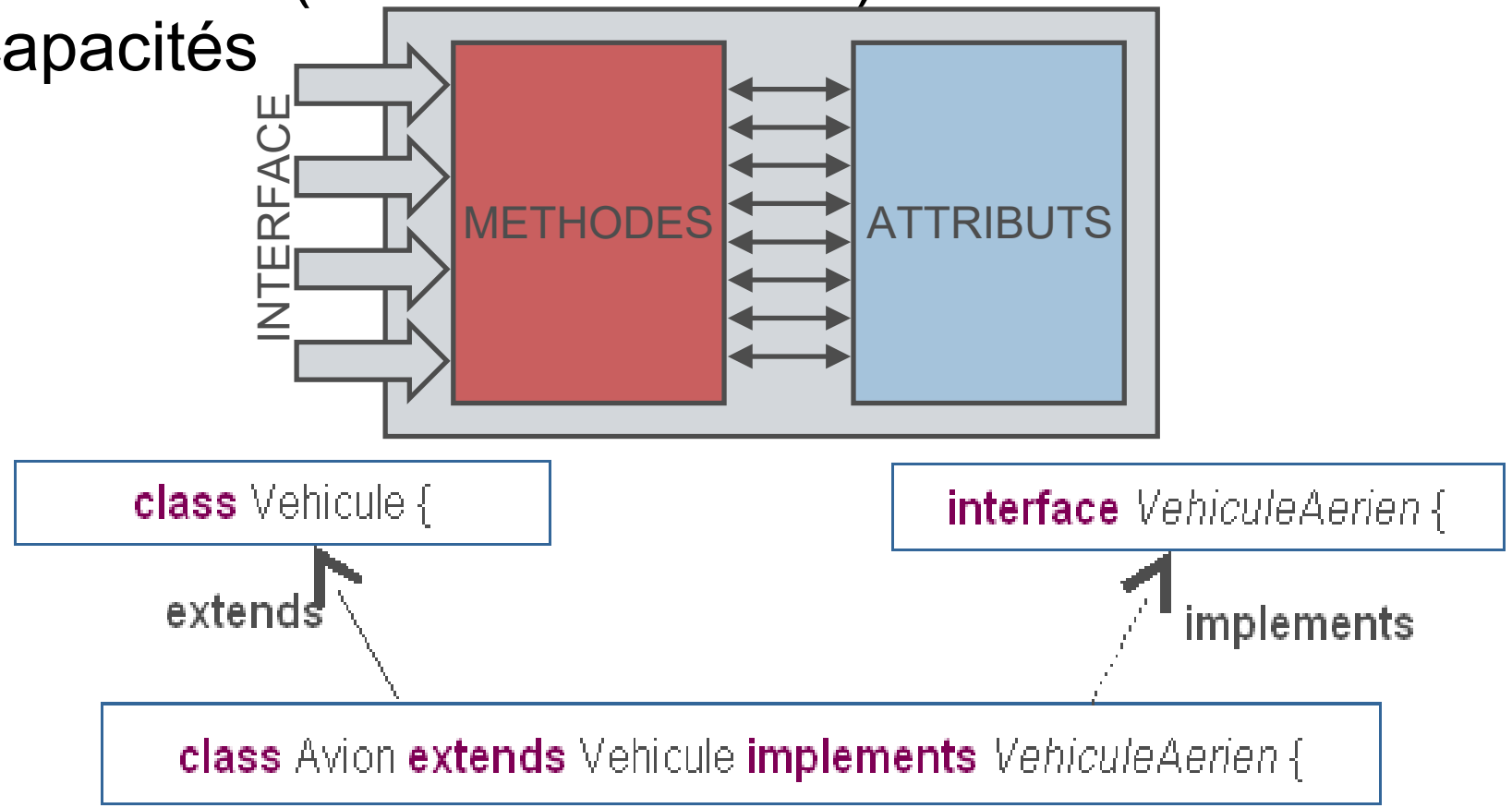
- Définit un type de squelette pour les sous-classes
  - Si elle contient des méthodes abstraites, les sous-classes doivent implémenter le corps des méthodes abstraites.

- Déclarée avec le mot clé **abstract**.



Interface

- Une pseudo classe abstraite marquée par le mot clé **interface** contenant juste des signatures de méthodes (et des constantes) afin de montrer les capacités



Exercices

- https://www3.ntu.edu.sg/home/ehchua/programming/java/J3f_OOPExercises.html

Exceptions

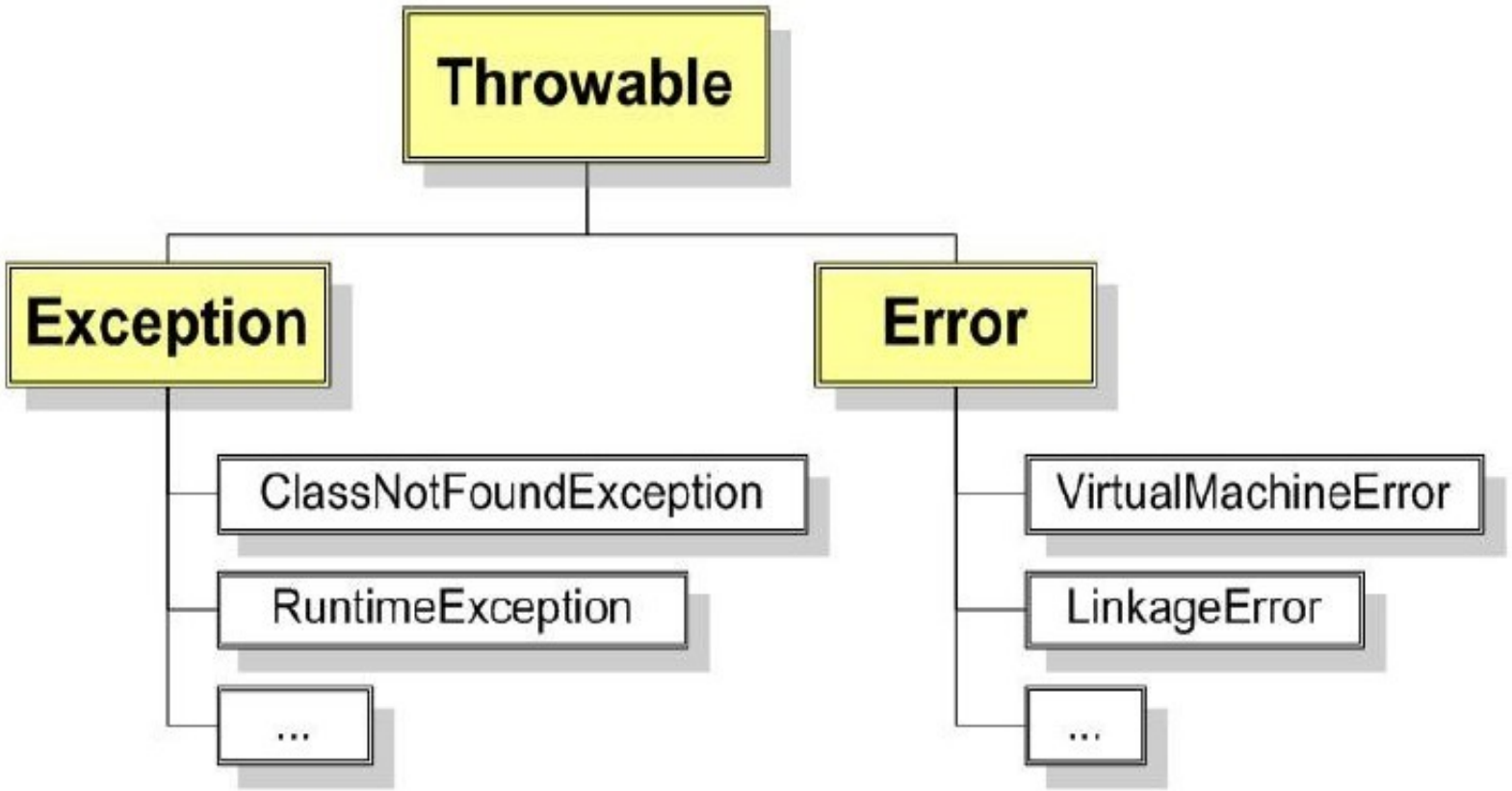
Définition

An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

— **Sun MicroSystems**
Docbook

La classe Throwable

- Diagramme des héritages :



Types d'Exceptions

- **Checked exceptions**
  - Le développeur doit les anticiper et coder des lignes pour les traiter.
- Exemple : Ouvrir un fichier qui n'existe pas.

- **Errors**
  - On ne doit pas les identifier et le programme s'arrête en les rencontrant.
- Exemple : La JVM charge une classe inexistante.

- **Runtime exceptions**
  - Ne peuvent être prévues (dans certains cas)
- Exemple : Essayer de lire une valeur en dehors d'un tableau.

Le bloc « try » et « catch »

Utilisé pour encadrer un bloc susceptible de déclencher une exception

```
try {
     // des lignes de code susceptibles
     // de lever une exception
}
catch   (IOException e) {
     // traitement de l'exception de type IO
}
finally {
      // toujours exécuter, même sans
      // exception ou une exception imprévue
}
```

Les mots clés « throw » et « throws »

- Le mot clé **throw** est utilisé pour déclencher une exception à n'importe quel moment
Ex : test d'une valeur positive

- Le mot clé **throws** est utilisé pour dire à la méthode de ne pas récupérer l'exception localement mais plutôt l'envoyer dans la méthode appelante.

```
public void methode3() throws IOException {
        throw new IOException("Fichier manquant");
}
```

Créer ses propres exceptions

Il faut seulement hériter de la classe Throwable ou une sous-classe (généralement la classe Exception)

```
public class MonException extends Exception {
      // Une exception valide !
}
```

```
public class NegativeNumberException
   extends NumberException {

      public NegativeNumberException(int num) {
            super("Le nombre "+num+"est négatif");
      }

      // C'est une exception valide également  !
}
```

Conventions

- Codage normalisé habituel :
  - Indentation significative (tabulation), 80 caractères par ligne, opérateur en début
  - Accolades : ouvrantes en fin de ligne, fermantes isolée
  - **statiques** puis d'instance**, Attributs** puis **constructeurs** puis **méthodes**, **public** puis **protégé** puis **privé**
  - Intitulés : un.package, UneClasse, UneInterface, uneMethode, uneVariable, unAttribut, UNE_CONSTANTE

Exercices

Créer une classe *Ville* (nomVille, nmbreHabitants) et mettre en œuvre une exception cru afin d'interdire l'instanciation d'un objet Ville présentant un nombre négatif d'habitants:

1. créer une classe héritant de la classe Exception : NombreHabitantException ;

2. renvoyer l'exception levée à notre classe NombreHabitantException ;

3. ensuite, gérer celle-ci dans notre classe NombreHabitantException.

Classes essentielles

Package *java.lang*

- Le package *java.*lang est importé automatiquement
- Contient les services que l'on retrouve éparpillés dans les langage proceduraux :
  - Wrappers (*Integer, Double*...)
  - *Exception/RuntimeException/Error*
  - Chaînes de caractères
  - Divers (*Class, Math, Thread, System...*)

String

- Chaîne *non mutable* de caractères unicodes
- Méthodes nombreuses de lecture et modifications des chaînes

```
String s = "abcd".substring(2); //cd
String s = "abcd".substring(2,3); //c
String s = "   ab cd ".trim(); //ab cd
String s = "aba".replace("a","c"); //cbc
int i = "abcd".indexOf("bc"); //1
int i = "abcd".length(); //4
```

Math

- Classe contenant des implémentations standard (*abs()*, *cos()*, *floor()*, *min()*, *round()*...)
- Les méthodes sont toutes statiques
- Constantes E et PI

```
int i = (int)Math.floor(3.99/2);
long l = Math.round(.51);
double d = Math.sin(Math.PI * .75);
double d2 = Math.pow(-1,.5);
```

Les Collections

Définition

- Les Collections sont un *framework* Java (depuis la version 1.2), le package est *java.util*

- *Collections framework* est une architecture pour la représentation et la manipulation de collections.

Il contient des :

  - Interfaces
  - Implémentations
  - Algorithmes

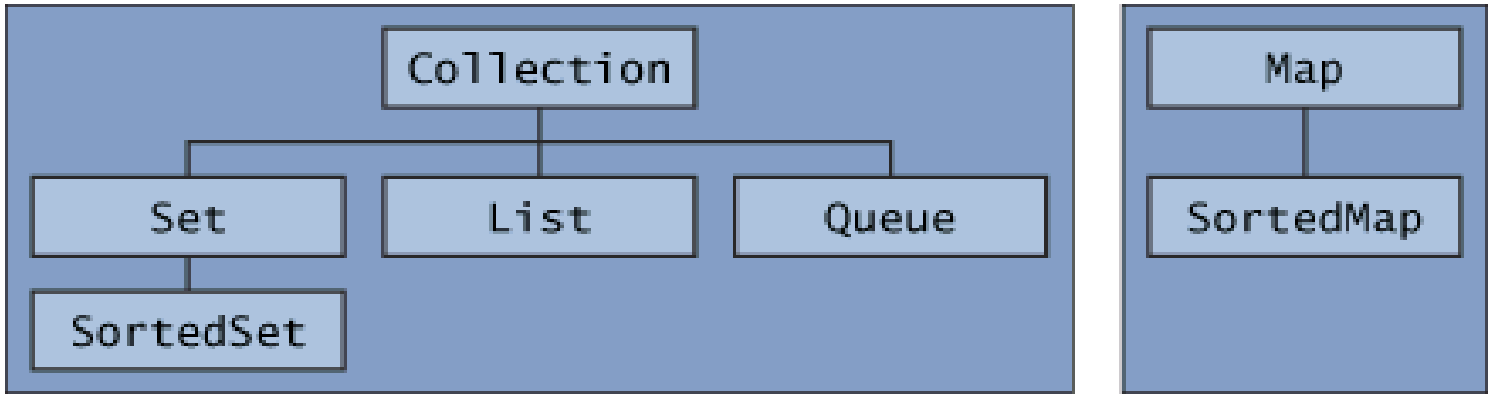
Les générics

Utilisés pour typer une variable

Depuis Java 5.0, ils sont utilisés dans les collections (i.e. ArrayList est une collection générique, que l'on spécialise par les symboles < >)

```
ArrayList<String>l = new ArrayList<String>();
l.add("Hell");
l.add("o world");
System.out.println(l.size()+" mots");
for(String s:l)
   System.out.print(s);
```

Interface Collection



Application :

- Parcours de collections (iterator ...)
- Manipulation des différentes implémentations de ces interfaces (Vector, ArrayList, LinkedList, HashSet...)

Entrées / Sorties

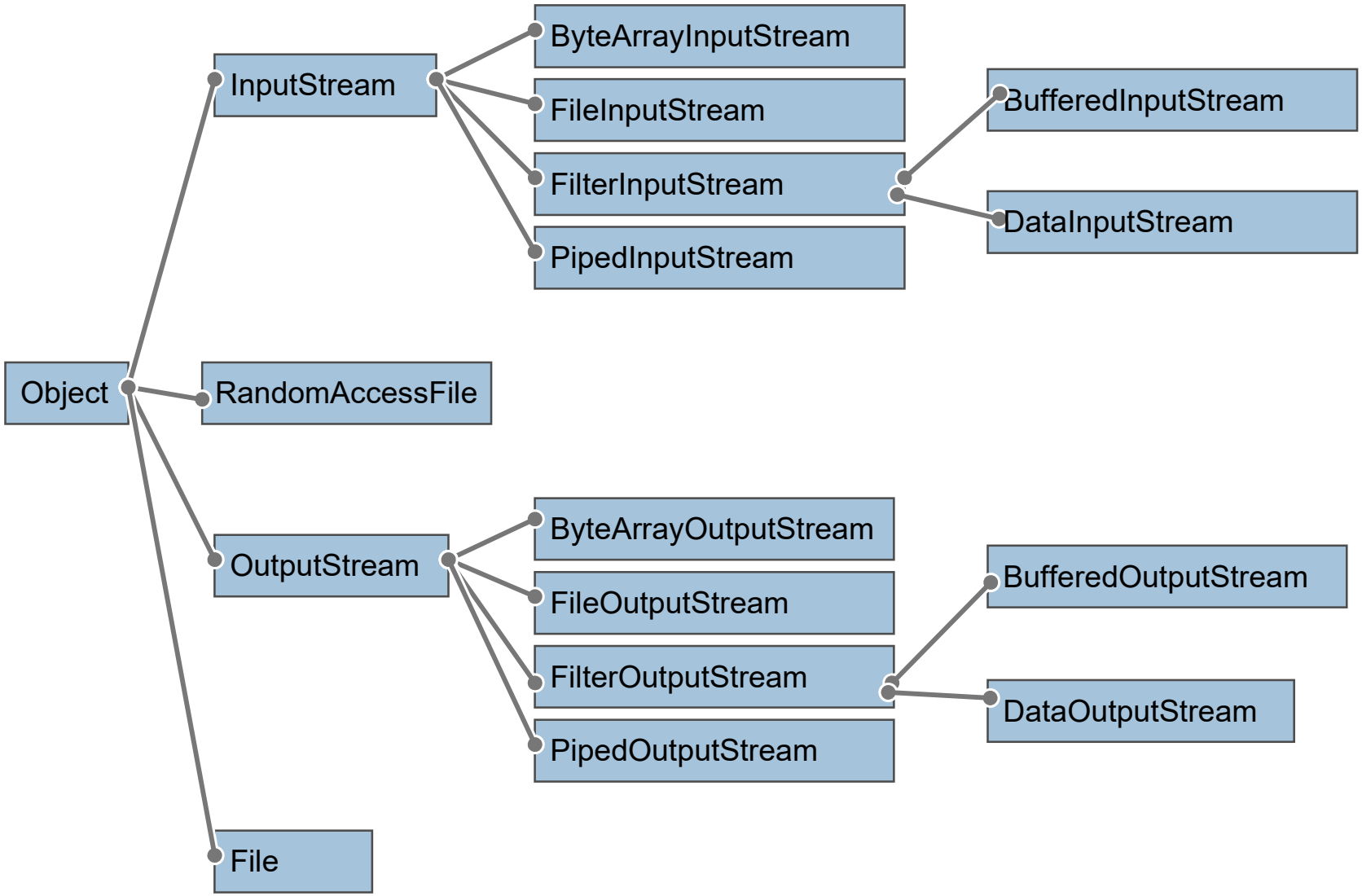
Définitions

- **Stream** = flux de données (sériel, temporaire, en entrée ou sortie)
- Le package **java.io** englobe les classes permettant la gestion des flux

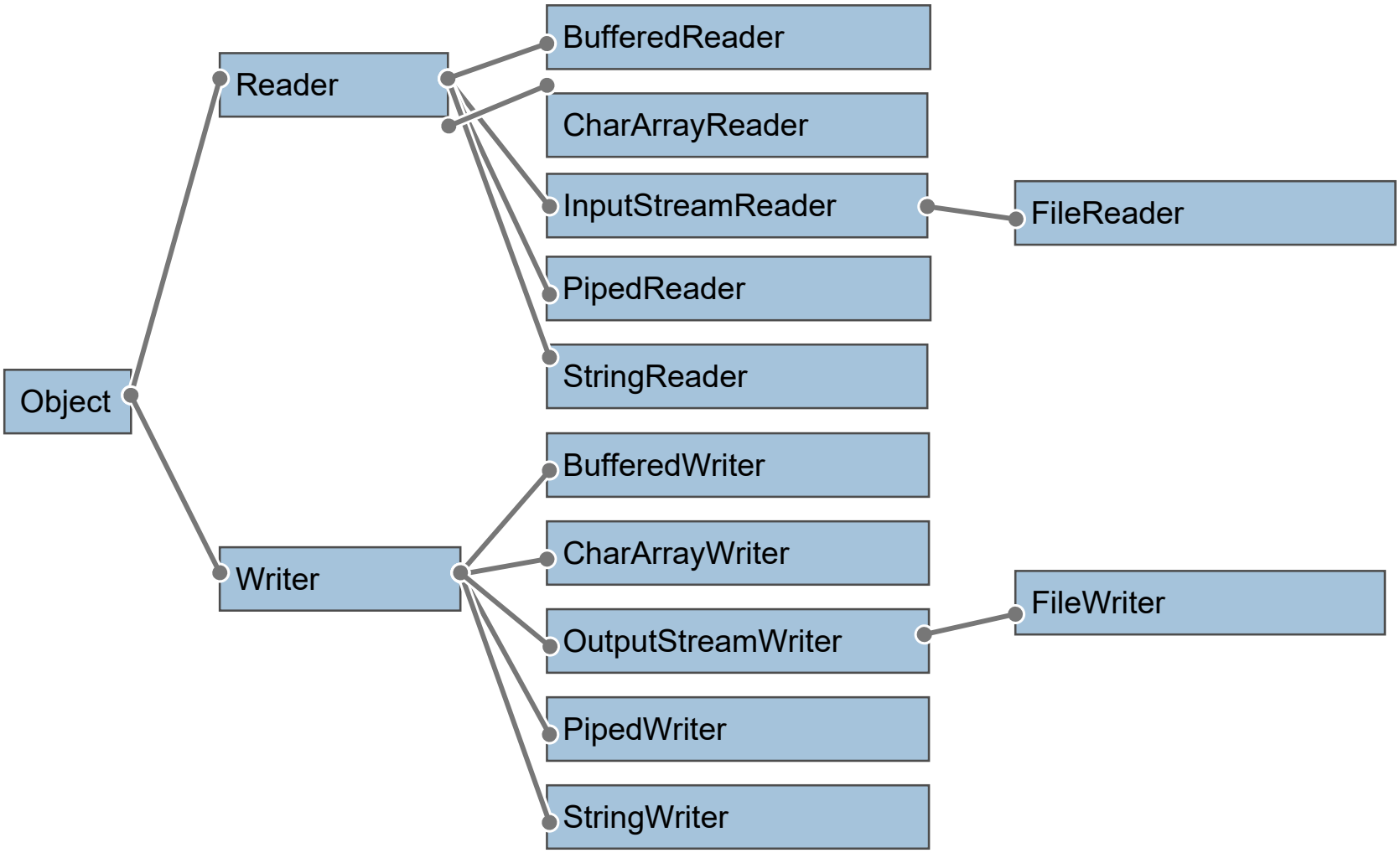
- **Principe d'utilisation d'un flux**:
  - Ouverture du flux
  - Identification de l'information (lecture/écriture)
    - Fermeture du flux

  InputStream et *OutputStream* pour la gestion d'un flux binaire, *Reader* et *Writer* pour un flux de caractères

Aperçu des classes



Aperçu des classes



Applications

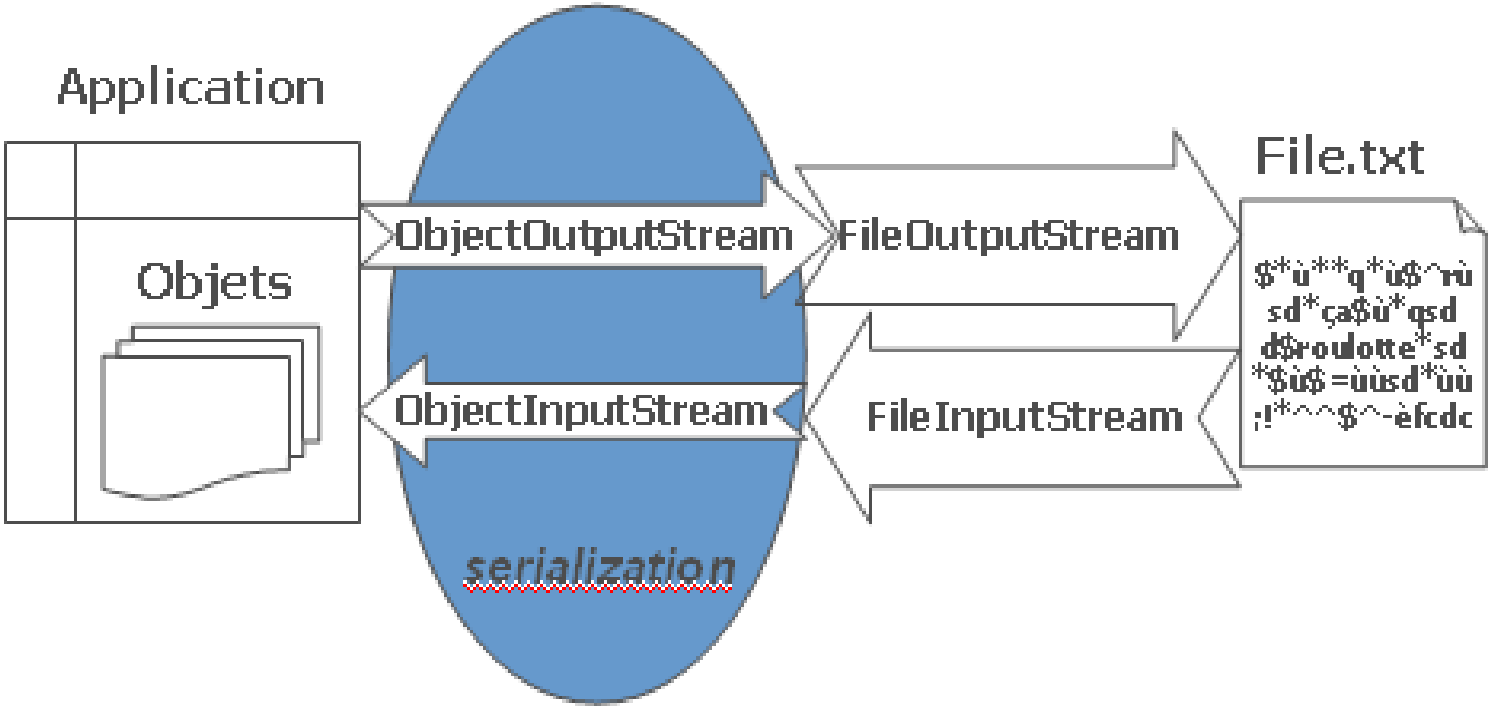
- Manipulation des différentes méthodes de la classe File : création d'un fichier, vérification si répertoire et lister les différents fichiers d'un répertoire.

- Manipulation des différents types de flux : Object, Data, byte, char etc...

- Ecriture de programmes permettant la copie de fichiers : utilisant Input/Output puis Reader/Writer

Sérialisation d'objets

La sérialisation permet de sauvegader l'état d'un objet dans un support persistant.




```
class Test implements java.io.Serializable{…}
```

Sérialisation d'objets

- transient pour définir un attribut non sérialisable :

```
public transient String password = "";
```

ObjectOutputStream pour la sérialisation :

```
void writeObject(Object o);
```

ObjectInputStream pour la désérialisation :

```
Object readObject();
```

Application

- Sérialisation d'une instance de Voiture dans un fichier.

- Désérialisation à partir d'un fichier