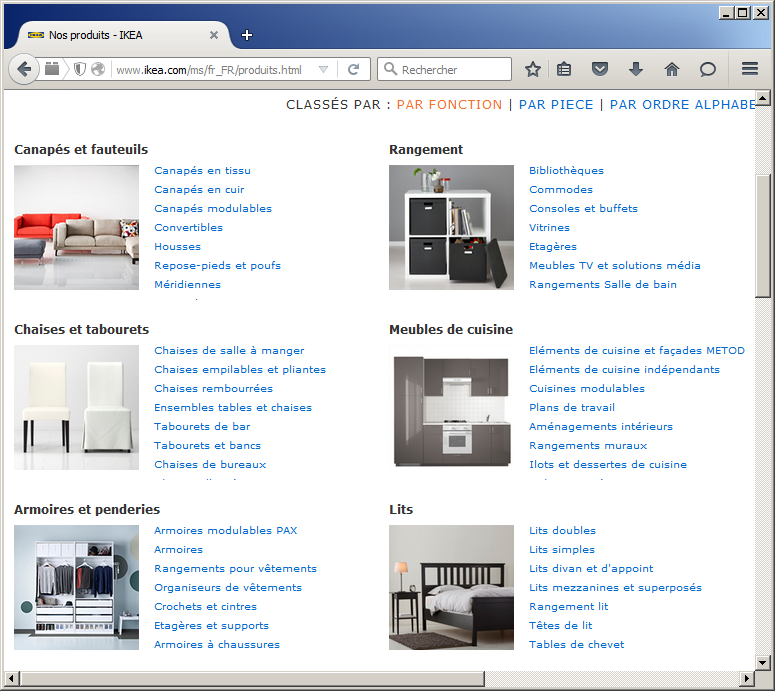
IUT d’Orsay - DUT Informatique - Jean-Claude MARTIN – JEAN-CLAUDE.MARTIN@U-PSUD.FR

**Programmation Orientée Objet (POO)  
COURS 2 : Classes et Objets**

# Principes de la programmation orientée objet

La Programmation Orientée Objet peut être utilisée dans de nombreux domaines : gérer les produits d’un catalogue en ligne, gérer les contrats d'assurance voiture d'une société d'assurance, … :

Exemple : catalogue de meubles sur un site web :



En programmation structurée, on se place dans l’espace “solution”

(contraint par les capacités de la machine : test, boucles, …).

En programmation orientée objet, on programme en se plaçant

dans l’espace “problème” (représentation explicite des concepts du problème

à résoudre, de leurs relations).

SCHEMA ESPACE PROBLEME / ESPACE SOLUTION

La programmation orientée objet peut se définir comme l'art de décomposer une application en plusieurs classes. Le but est de réduire la difficulté de la tâche à accomplir en la divisant en un grand nombre de petits problèmes qui sont plus simples à comprendre et à résoudre (modularité).

Une classe regroupe LOGIQUEMENT les différentes variables concernant une même entité ainsi que les sous-programmes qui manipulent ces variables.

Une fois les classes définies, l’exécution d’un programme orienté objet consiste à

1) créér des objets..

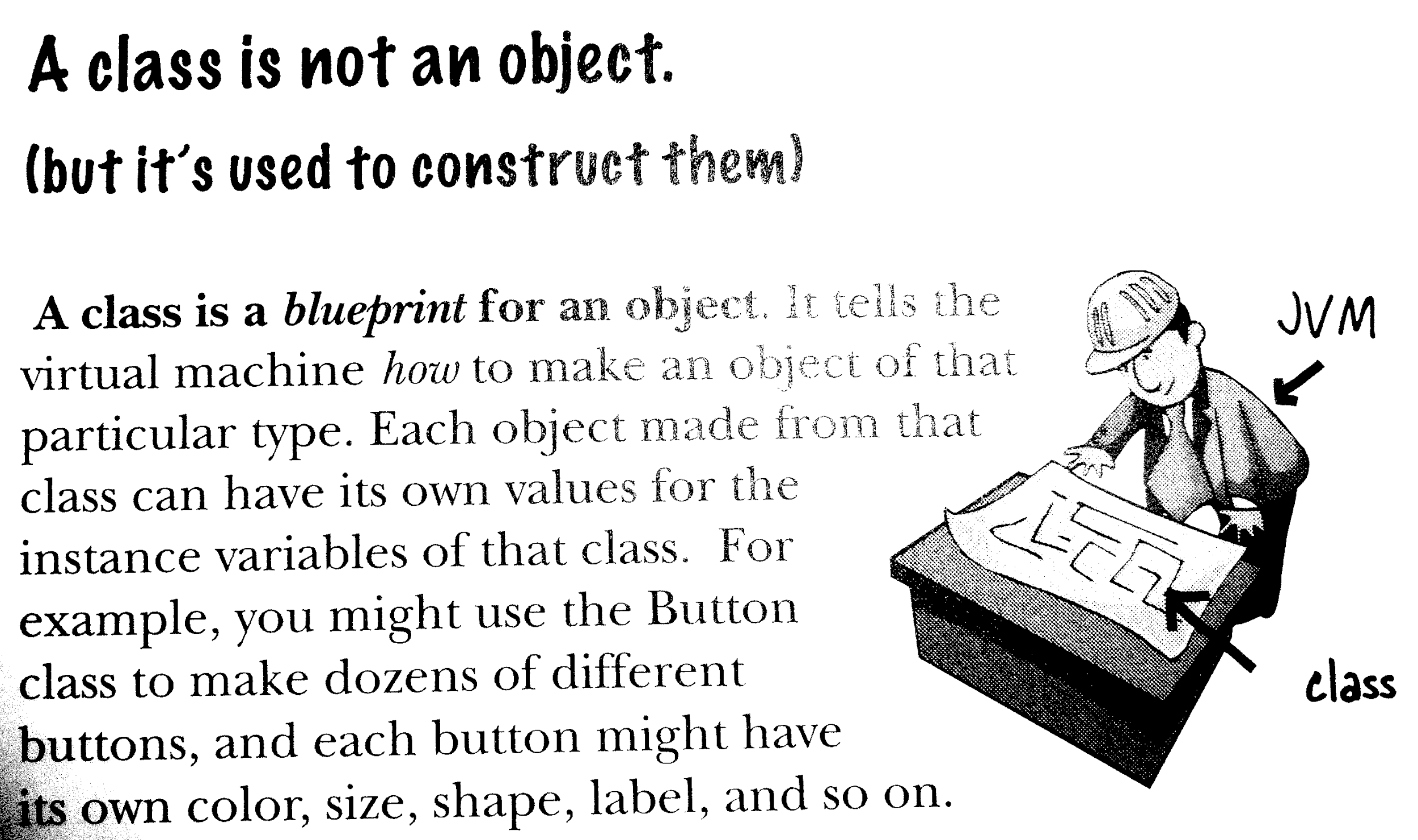
2) appeler des méthodes sur ces objets

<https://youtu.be/Rrm8usaH0sM?t=2m43s>

# Définition d’une classe

Une *classe* définit le “moule ou plan” selon lequel seront conçus plusieurs objets.

Un o*bjet* est définit par son état (valeur de ses variables / attributs) et un comportement (on sait quelles méthodes on peut appeler sur cet objet).



# Attributs d’objets

Exemple : Une voiture est caractérisée par une année, une puissance, une valeur.

**public** **class** Voiture {

int annee ;

int puissance ;

double valeur ;

}

ATTENTION : Les attributs des objets sont initialisés à des valeurs par défaut

(0 pour les nombres, null pour les références).

Les variables locales aux méthodes ne sont pas initialisées !

**public** **class** Assurance {

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

// Declarer une reference a une voiture

// Utiliser l'operateur new

// - allouer de la place memoire dans le tas pour tous les attributs de cet objet

// - initialiser les attributs à des valeurs par défauut

// - retourner l'adresse de la zone allouee

// Affecter des valeurs aux attributs de cet objet

// Afficher les valeurs des attributs de cet objet

// Declarer une reference vers une 2eme voiture

// Creer un objet, modifier les attributs et les afficher

………………………………………………………

………………………………………………………

}

}

SCHEMA DE LA MEMOIRE : RUBAN

SCHEMA DE LA MEMOIRE : BOITE ET FLECHE

Schémas de la mémoire

le but des schémas est de faire comprendre comment la machine virtuelle gère la mémoire.   
La zone "pile" est utilisée pour les variables locales aux sous programmes, paramètres passés lors des appels.   
La zone "tas" est utilisée pour l'allocation dynamique (attributs des objets, cases des tableaux).   
  
Le mode ruban représente un ruban avec des cases mémoires avec leurs adresses en bas et en haut le nom des variables.   
  
Le mode boite et flèche s'abstrait des adresses et représente les références par des flèches.   
C'est donc un mode plus lisible quand il y a beaucoup d'objets.

Chaque objet a ses propres attributs (ex : année) : on parle de **variable d’objet.**

Les variables / attributs d’un même objet (ex : la voiture de mr. Durand) sont regroupées dans une même zone de la mémoire.

ATTENTION : Un programme = généralement plusieurs fichiers .java.

Une seule classe public par fichier .java

### Un attribut peut être un objet

Le propriétaire d’une voiture : instance de la classe Personne

**public** **class** Personne {

**int** anneeNaissance ;

}

**public** **class** VoitureProprietaire {

**int** annee ;

**int** puissance ;

**double** valeur ;

Personne proprietaire ;

}

SCHEMA MEMOIRE :

### Application

* Compléter la classe Personne en ajoutant un attribut numéroSecu
* Compléter le main de la classe VoitureProprietaire en créant une 2ème voiture.
* Le propriétaire de cette voiture sera le même pour les 2 voitures.
* Faire un schéma de la mémoire boite et fleches.

### NullPointerException

Attention : Erreur NullPointerException si la référence contient null (par exemple attribut initialisé à la valeur par défaut sans affectation ensuite à une adresse d’objet alloué)

**public** **class** VoitureProprietaire {

**int** annee ;

**int** puissance ;

**double** valeur ;

Personne proprietaire ;

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

VoitureProprietaire v = **new** VoitureProprietaire();

System.*out*.println (v.proprietaire.anneeNaissance);

}

}

Exécution :

………………………………………………………

………………………………………………………

………………………………………………………

………………………………………………………

### Références multiples sur un même objet

ATTENTION : Plusieurs références peuvent pointer sur un même objet.

Voiture v2 = scenic ;

v2.annee = 2007 ;

System.*out*.println(" Annee modifie : " + scenic.annee);

SCHEMA PLUSIEURS REFERENCES SUR UN MEME OBJET

# Constructeur

On alloue un objet en appelant l’opérateur new avec un *constructeur d’objet*.   
Un constructeur est un sous-programme particulier qui possède le même nom que sa classe et qui n’a aucun type de retour. Il peut avoir des paramètres.

Il est appelé lorsqu’une nouvelle instance de la classe est créée. Ceci permet à la classe de configurer un objet en vue de son utilisation (par exemple initialisation de ses attributs). L'intérêt est que, comme il est appelé automatiquement, le développeur qui utilise la classe n'oublie pas des initialisations importantes. Préférer grouper toutes les initialisations plutot que d’initialiser certains attributs explicitement (eg meme ligne que la declaration).

**this** : Plutôt que de donner des noms différents aux paramètres du constructeur et aux attributs de l’objet, on préfère souvent utiliser les mêmes noms. Dans ce cas, pour éviter une ambiguité, on utilise le mot clé **this** comme une référence vers l’objet qui vient d’être crée.

ATTENTION : Un constructeur sans paramètre est crée par défaut seulement si on ne définit pas de constructeur.

**public class VoitureConstructeur {**

**int annee ;**

**int puissance ;**

**double valeur ;**

**VoitureConstructeur (int an, double valeur) {**

**annee = an ;**

**this.valeur = valeur ; // l’attribut valeur de CET objet que je construit.**

**}**

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

**VoitureConstructeur v ;**

**v = new VoitureConstructeur (2007, 7);**

VoitureConstructeur v2 = **new** VoitureConstructeur ();

// ERREUR CAR …………………………………………………………….

}

}

Message d’erreur :

………………………………………………………

………………………………………………………

# 

# Tableaux d’objets

* Déclarer une référence qui contiendra l’adresse du tableau
* Allouer le tableau (n références d’objets)
* Créer les objets et stocker leur adresse dans une case du tableau
  + Parcourir le tableau en accédant a un attribut d’un objet par t[i].attribut

VERSION 1

Tableau d’objet déclaré, alloué et géré en variable locale au main

…

VERSION 2

Tableau d’objet déclaré en attribut, alloué dans le constructeur et avec une méthode d’objet pour ajouter un élément

**public class CabinetAssurance {**

**final int NB\_MAX = 10 ;**

**Voiture [] voitures ;**

**int nbVoitures ;**

**CabinetAssurance () {**

**voitures = new Voiture [NB\_MAX];**

**}**

**void ajouterVoiture (Voiture v1) {**

**voitures[nbVoitures] = v1 ;**

**nbVoitures++ ;**

**}**

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

**CabinetAssurance a = new CabinetAssurance();**

**Voiture durand = new Voiture ();**

**durand.annee = 1999 ;**

**a.ajouterVoiture(durand);**

}

}

SCHEMA MEMOIRE

# Attributs de classe (static)

Un attribut static est UN ATTRIBUT QUI EST PARTAGE PAR TOUTES LES INSTANCES DE LA CLASSE

Exemple : un compteur d’instances d’une classe pour numéroter les objets

1. Déclaration d’un compteur static
2. Déclaration d’un attribut numéro
3. Incrémentation du compteur dans le constructeur
4. Affectation de la valeur actuelle du compteur comme numéro de l’objet en construction

**public class Joueur {**

**static int *nbJoueur* ; // VARIABLE GLOBALE A TOUS LES OBJETS INSTANCES DE JOUEUR**

**int numero ;**

**Joueur () {**

***nbJoueur* ++ ;**

**numero = *nbJoueur* ;// AFFECTER UN NUMERO A CET OBJET QUE JE CREE**

**}**

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

**Joueur j1 = new Joueur ();**

**System.*out*.println (*nbJoueur*);**

**System.*out*.println ("no j1 : " + j1.numero);**

**Joueur j2 = new Joueur ();**

**System.*out*.println (*nbJoueur*);**

**System.*out*.println ("no j2 : " + j2.numero);**

**}**

**}**

# Définition d’une constante nommée et typée

Au même niveau que les attributs

final = l’attribut ne pourra se voir affecter une valeur qu’une seule fois

static final => la constante est partagée par tous les objets instances de la classe (par exemple nombre de points maximum d’un polygone)

**public** **class** Polygone {

**static final** **int** NB\_MAX = 10 ;

Point [] points ;

// Allocation dans le constructeur

final (non static) => la constante est différente pour chaque objet (par exemple un nombre maximum de contrats d’assurance qui sera différent pour chaque cabinet d’assurance)

# Modificateurs d’accès des attributs

### Visibilité / accessibilité des attributs d’une classe

Pour un attribut, accessible = on peut consulter ou modifier sa valeur.

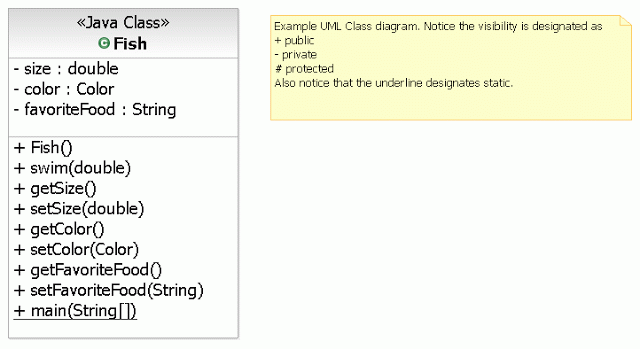
* public : accessible par toutes les classes
* private : accessible seulement dans les méthodes de cette classe

But :

* protéger les valeurs des attributs
* simplifier ce que les autres développeurs doivent connaître de votre classe.
* Eclipse permet de générer automatiquement les accesseurs !

UML :

* + = public
* - = private



# Méthodes d’objet

* Définit le comportement d’un objet.
* Définit les sous-programmes qu’on peut appeler SUR CET objet.
* Lors de l’exécution d’une méthode d’objet, l’interpréteur a accès à tous les attributs de cet objet.
* Une méthode peut utiliser “this”.

|  |  |  |
| --- | --- | --- |
| telecommande.jpg | unbalai.jpg | desbalais.jpg |
| bouton = appel une méthode  (une télécommande par objet) |  |  |

# Surcharge de méthodes / “Overloading”

La surcharge de méthodes correspond à la capacité de définir plusieurs méthodes qui possèdent le même nom dans la même classe mais pas la même liste de types de paramètres. Exemple :

* **constructeur de la classe Voiture (), Voiture (int an, double valeur)   
  => on dit que le constructeur est surchargé**

Lorsque la méthode est appelée, le compilateur choisit **LA** méthode qu’il convient d’exécuter en fonction des arguments qui doivent lui être passés.

Les différentes versions de la méthode doivent avoir un nombre ou des types d’arguments différents.

# Passage d’objets en paramètre

Dans un sous-programme auquel on passe un objet o en paramètre :

* On peut modifier ………les attributs de l’objet……………………………………………………….
* On ne peut pas modifier …l’adresse de l’objet (on travaille sur une copie du paramètres)r…………..

# Classes du JDK

* Utiliser la documentation des API Java (javadoc)
* Exemples de classes : System, Date, Wrapper pour les types de base : Integer (permet de gérer de manière homogène des objets et des types de base)

# Définition d’un objet

Un *objet est défini par :*

* …………………………………………………………………………………………………………………..
* …………………………………………………………………………………………………………………..
* …………………………………………………………………………………………………………………..