# Programme

- Introduction à la programmation informatique  et aux différents langages de programmation.
- Présentation des concepts de base de la programmation de haut niveau. Aspect algorithmique. Introduction au langage Java.
- Syntaxe de base du langage Java, la sémantique et le concept de la compilation.
- Les variables, types et opérations mathématiques
- Opérateurs, affectation, ....
- conditionnelles et boucles (1 et 2)
- Tableaux ( accès aux composants, …).
- Procédures et Fonctions.
- Présentations de quelques classes clés.
- Utilisation d'objet : String
- Ma première Classe/Un exemple applicatif de différentes notions de Java.

# Aujourd'hui

## Utilisation d'objet : String

- Introduction
- ma première classe
- ma première classe : la porté
- ma première classe: les constructeurs
- Accesseurs et mutateurs
- mot-clé `this` !
- Exercice d'application

# Introduction

Rappel: Une classe peut être comparée à un moule qui, lorsque nous le remplissons, nous donne un objet ayant la forme du moule ainsi que toutes ses caractéristiques.

Notre classe contenant la méthode `main` ressemble à ceci :

```
public class ClassePrincipale {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }}
```

# Ma première Classe (1/8)

- Nous allons créer une seconde classe dans un projet!
    - Créons une classe Ville.
- Cette fois, vous ne devez pas y créer la méthode `main`.
    - Il ne peut y avoir qu'une seule méthode main active par projet !
    - celle-ci est le point de départ de votre programme.
    - plusieurs méthodes main peuvent cohabiter dans votre projet, mais une seule sera considérée comme le point de départ de votre programme !

# Ma première Classe (2/8)

```
public class Ville {
}
```

- La classe `Ville` est précédée du mot clé `public`: lorsque nous créons une classe, Eclipse nous facilite la tâche en ajoutant automatiquement ce mot clé, qui correspond à la portée de la classe.
- Retenons pour l'instant que `public class UneClasse{}` et `class UneClasse{}` sont presque équivalents !

# Ma première Classe (3/8)

## La porté

- En programmation, la portée détermine *qui* peut faire appel à une classe, une méthode ou une variable.

- La portée public : cela signifie que tout le monde peut faire appel à l'élément.

- Une méthode marquée comme public peut donc être appelée depuis n'importe quel endroit du programme.

- Une méthode marquée comme private, cela signifie que notre méthode ne pourra être appelée que depuis l'intérieur de la classe dans laquelle elle se trouve !

- Les méthodes déclarées private correspondent souvent à des mécanismes internes à une classe que les développeurs souhaitent « cacher » ou simplement ne pas rendre accessibles de l'extérieur de la classe…

# Ma première Classe (4/8)

## Les constructeurs

Notre objectif est de construire un objet Ville, cet objet possède :

- un nom, sous la forme d'une chaîne de caractères ;
- un nombre d'habitants, sous la forme d'un entier ;
- un pays apparenté, sous la forme d'une chaîne de caractères.

```
public class Ville{
   String nomVille;
   String nomPays;
   int nbreHabitants;
}
```

- Pour créer notre premier objet, nous allons devoir utiliser ce qu'on appelle des constructeurs.

# Ma première Classe (5/8)

- Un constructeur est une méthode d'instance qui va se charger de créer un objet et, le cas échéant, d'initialiser ses variables de classe !
- Cette méthode a pour rôle de signaler à la JVM (Java Virtual Machine) qu'il faut réserver de la mémoire pour notre futur objet et donc, par extension, d'en réserver pour toutes ses variables.
- Notre premier constructeur sera ce qu'on appelle communément *un constructeur par défaut*, c'est-à-dire qu'il ne prendra aucun paramètre,

```
public class Ville{
  //Stocke le nom de notre ville
  String nomVille;
  //Stocke le nom du pays de notre ville
  String nomPays;
  //Stocke le nombre d'habitants de notre ville
  int nbreHabitants;

  //Constructeur par défaut
  public Ville(){
    System.out.println("Création d'une ville !");
    nomVille = "Inconnu";
    nomPays = "Inconnu";
    nbreHabitants = 0;}}
```

# Ma première Classe (6/8)

- Le constructeur est une méthode qui n'a aucun type de retour (void, double…) et qui porte le même nom que la classe ! Ceci est une règle immuable : le (les) constructeur(s) d'une classe doit (doivent) porter le même nom que la classe !

**Instanciation d'un objet ville**:

```
public static void main(String[] args) {
        Ville ville = new Ville();
}
```

# Ma première Classe (7/8)

Pour créer un Objet avec des paramètres, exemple:

```
Ville ville1 = new Ville("Marseille", 123456789, "France");
```

Il faut utiliser un constructeur avec paramètres:

```
public class Ville {
  String nomVille; //Stocke le nom de notre ville
  String nomPays; //Stocke le nom du pays de notre ville
  int nbreHabitants; //Stocke le nombre d'habitants de notre ville
  public Ville(){ //Constructeur par défaut
    System.out.println("Création d'une ville !");
    nomVille = "Inconnu";
    nomPays = "Inconnu";
    nbreHabitants = 0;}
  //Constructeur avec paramètres
  //J'ai ajouté un « p » en première lettre des paramètres.
  //Ce n'est pas une convention, mais ça peut être un bon moyen de les repérer.
  public Ville(String pNom, int pNbre, String pPays)
  {
    System.out.println("Création d'une ville avec des paramètres !");
    nomVille = pNom;
    nomPays = pPays;
    nbreHabitants = pNbre;}}
```

# Ma première Classe (8/8)

- Il faut respecter scrupuleusement l'ordre des paramètres passés lors de l'initialisation de votre objet : sinon, c'est l'erreur de compilation:

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

        //L'ordre est respecté -> aucun souci
        Ville ville1 = new Ville("Marseille", 123456789, "France");
        //Erreur dans l'ordre des paramètres -> erreur de compilation au
        final
        Ville ville2 = new Ville("France", 12456,  "Lille");
    }
```

**Pour afficher les paramètres**:

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

      { Ville ville =  new Ville();
        System.out.println(ville.nomVille);
        ville.nomVille = "Paris";
        System.out.println(ville.nomVille);

        Ville ville2 = new Ville("Marseille", 123456789, "France");
        ville2.nomPays = "Lyon";
        System.out.println(ville2.nomPays);}
```

# Accesseurs et mutateurs (1/7)

**Définition**

- Un accesseur est une méthode qui va nous permettre d'accéder aux variables de nos objets en lecture.

- Un mutateur nous permettra d'en faire de même en écriture !

Grâce aux accesseurs, vous pourrez afficher les variables de vos objets, et grâce aux mutateurs, vous pourrez les modifier.

# Accesseurs et mutateurs (2/7)

## Coté code:

```
//*************   ACCESSEURS
*************
  //Retourne le nom de la ville
  public String getNom()  {
    return nomVille;
  }
  //Retourne le nom du pays
  public String getNomPays()
  {
    return nomPays;
  }
  // Retourne le nombre d'habitants
  public int getNombreHabitants()
  {
    return nbreHabitants;
  }
```

```
 //*************   MUTATEURS
*************
  //Définit le nom de la ville
  public void setNom(String pNom)
  {
    nomVille = pNom;
  }
  //Définit le nom du pays
  public void setNomPays(String pPays)
  {
    nomPays = pPays;
  }
  //Définit le nombre d'habitants
  public void setNombreHabitants(int
nbre)
  {
    nbreHabitants = nbre;
  }
```

# Accesseurs et mutateurs (3/7)

- Les accesseurs sont bien des méthodes, et elles sont `public` pour pouvoir y accéder depuis une autre classe que celle-ci : depuis le `main`, par exemple.
- Les accesseurs sont du même type que la variable qu'ils doivent retourner. Les mutateurs sont, par contre, de type `void`. Ce mot clé signifie « rien » ; en effet, ces méthodes ne retournent aucune valeur, elles se contentent de les mettre à jour.

- Lorsqu'on parle d'accesseurs, ce terme inclut également les mutateurs.
- Question de convention de nommage: les accesseurs commencent par `get` et les mutateurs par `set`. On parle d'ailleurs parfois de *Getters* et de *Setters*.

# Accesseurs et mutateurs (3/7)

Exemple d'application:

Ecrire un programme qui va créer deux objets v1 et v2

- v1 = ville Marseille, nb d'habitants: 123456, pays: France.
- v2 = ville Rio, nb d'habitants: 321654 et pays: Brésil.

Les interchanger par l'intermédiaire d'un autre objet Ville.

- v1 = ville Rio, nb d'habitants: 321654 et pays: Brésil.
- v2 = ville Marseille, nb d'habitants: 123456, pays: France.

Changer le nom de ces deux villes tel que :

v1 = Hong Kong ville de 321654 habitants se situant en Brésil

v2 = Djibouti ville de 123456 habitants se situant en France

# Accesseurs et mutateurs (4/7)

```
public static void main(String[] args)
{
        Ville v = new Ville();
         Ville v1 = new Ville("Marseille", 123456, "France");
         Ville v2 = new Ville("Rio", 321654, "Brésil");

         System.out.println("\n v = "+v.getNom()+" ville de
        "+v.getNombreHabitants()+ " habitants se situant en
        "+v.getNomPays());
         System.out.println(" v1 = "+v1.getNom()+" ville de
        "+v1.getNombreHabitants()+ " habitants se situant en
        "+v1.getNomPays());
         System.out.println(" v2 = "+v2.getNom()+" ville de
        "+v2.getNombreHabitants()+ " habitants se situant en
        "+v2.getNomPays()+"\n\n");
```

# Accesseurs et mutateurs (5/7)

```
/*Nous allons interchanger les Villes v1 et v2 tout ça par l'intermédiaire
d'un autre objet Ville.*/
        Ville temp = new Ville();
        temp = v1;
        v1 = v2;
        v2 = temp;
        System.out.println(" v1 = "+v1.getNom()+" ville de
       "+v1.getNombreHabitants()+ " habitants se situant en
       "+v1.getNomPays());
        System.out.println(" v2 = "+v2.getNom()+" ville de
       "+v2.getNombreHabitants()+ " habitants se situant en
       "+v2.getNomPays()+"\n\n");
```

# Accesseurs et mutateurs (6/7)

```
/*Nous allons maintenant interchanger leurs noms cette fois par le biais de
leurs mutateurs.*/
         v1.setNom("Hong Kong");
         v2.setNom("Djibouti");
         System.out.println(" v1 = "+v1.getNom()+" ville de
        "+v1.getNombreHabitants()+ " habitants se situant en
        "+v1.getNomPays());
         System.out.println(" v2 = "+v2.getNom()+" ville de
        "+v2.getNombreHabitants()+ " habitants se situant en
        "+v2.getNomPays()+"\n\n");
      }
```

**Exécution**:

```
Création d'une ville !
Création d'une ville avec des paramètres !
Création d'une ville avec des paramètres !
v = Inconnu ville de  0 habitants se situant en Inconnu
v1 = Marseille ville de  123456 habitants se situant en France
v2 = Rio ville de  321654 habitants se situant en Brésil
Création d'une ville !
 v1 = Rio ville de  321654 habitants se situant en Brésil
 v2 = Marseille ville de  123456 habitants se situant en France
 v1 = Hong Kong ville de  321654 habitants se situant en Brésil
 v2 = Djibouti ville de  123456 habitants se situant en France
```

# Accesseurs et mutateurs (7/7)

- les constructeurs -> méthodes servant à créer des objets ;

- les accesseurs -> méthodes servant à accéder aux données des objets ;

- les méthodes d'instance → méthodes servant à la gestion des objets.

# mot-clé this ! (1/4)

Nous souhaitons faire évoluer notre programme de ville tel que:

- nous souhaitons faire un système de catégories de villes par rapport à leur nombre d'habitants ( <1000 -> A, <10 000 -> B…): ajoutons donc une variable d'instance de type char à notre classe et appelons-la categorie;

- faire une méthode de description de notre objet Ville ;

- une méthode pour comparer deux objets par rapport à leur nombre d'habitants.

# mot-clé this ! (2/4)

- Pour réaliser ces évolutions, nous allons voir le mot-clé this en action dans le code de la classe Ville en entier.
- Pour simplifier, this fait référence à l'objet courant ! Bien que la traduction anglaise exacte soit « ceci », il faut comprendre « moi ».

À l'intérieur d'un objet, ce mot clé permet de désigner une de ses variables ou une de ses méthodes.

```
  //Définit le nombre d'habitants
  public void setNombreHabitants(int nbre)
  { nbreHabitants = nbre;
    this.setCategorie();
  }
  //Définit la catégorie de la ville
  private void setCategorie() {
    int bornesSuperieures[] = {0, 1000, 10000, 100000, 500000, 1000000, 5000000,
10000000};
    char categories[] = {'?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};
    int i = 0;
    while (i < bornesSuperieures.length && this.nbreHabitants > bornesSuperieures[i])
      i++;
    this.categorie = categories[i];
  }
  //Retourne la description de la ville
  public String decrisToi(){
    return "\t"+this.nomVille+" est une ville de "+this.nomPays+ ", elle comporte :
"+this.nbreHabitants+" habitant(s) => elle est donc de catégorie : "+this.categorie;
  }
  //Retourne une chaîne de caractères selon le résultat de la comparaison
  public String comparer(Ville v1){
    String str = new String();
    if (v1.getNombreHabitants() > this.nbreHabitants)
      str = v1.getNom()+" est une ville plus peuplée que "+this.nomVille;
    else
      str = this.nomVille+" est une ville plus peuplée que "+v1.getNom();
    return str;}
```

# mot-clé this ! (3/4)

Reprenons l'exemple précédent et:

- afficher la description des deux villes v1 et v2;
- définir leur catégorie,
- faire une comparaison entre elles.

# mot-clé this ! (4/4)

Dans la classe Principale, on ajoute:

```
System.out.println("\n\n"+v1.decrisToi());

System.out.println(v.decrisToi());
System.out.println(v2.decrisToi()+"\n\n");
System.out.println(v1.comparer(v2));
```

Exécution du code:

```
Hong Kong est une ville de Brésil, elle comporte : 321654 habitant(s) =>

elle est donc de catégorie : D

Inconnu est une ville de Inconnu, elle comporte : 0 habitant(s) => elle est
donc de catégorie : ?
Djibouti est une ville de France, elle comporte : 123456 habitant(s) => elle
est donc de catégorie : D


Hong Kong est une ville plus peuplée que Djibouti
```

# Exercices d'application (1/2)

Exo-1:

a) Créer une classe Adresse…

La classe doit posséder un constructeur qui permet de valoriser les 3 propriétés

b) La ville peut être retrouvée à partir du code postal. Ajoutez un constructeur qui permette de ne saisir que les 2 attributs rue et codePostal