

# Rappels sur *Java* et *PHP*

Olivier Flauzac & Cyril Rabat

Licence 3 Info - Info0503 - Introduction à la programmation client/serveur

2020-2021



## Cours n°1

*Rappels sur les langages utilisés en Info0503 : Java et PHP*

*Version 1<sup>er</sup> septembre 2020*

# Table des matières

## 1 Introduction

- *Java*
- PHP et *Javascript*

## 2 Rappels sur *Java*

- Les classes en *Java*
- L'héritage
- La classe `Object`
- Les interfaces et les classes abstraites
- Les paquetages
- Les flux
- Les fichiers

## 3 PHP : *Personal Home Pages*

- Rappels sur PHP
- Gestion des formulaires
- Un peu plus loin

# Table des matières

## 1 Introduction

- *Java*
- PHP et *Javascript*

## 2 Rappels sur *Java*

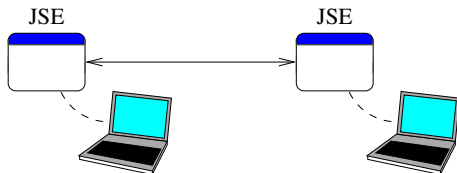
- Les classes en *Java*
- L'héritage
- La classe `Object`
- Les interfaces et les classes abstraites
- Les paquetages
- Les flux
- Les fichiers

## 3 PHP : *Personal Home Pages*

- Rappels sur PHP
- Gestion des formulaires
- Un peu plus loin

# JSE : *Java Standard Edition*

## Architecture

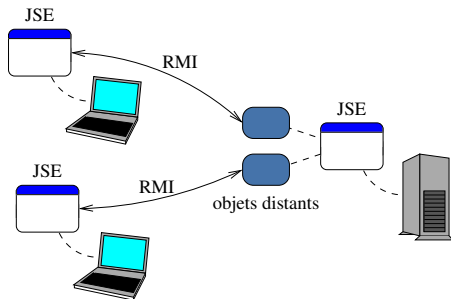


## Explications

- Clients lourds en *Java*
- Programmation : JDK JSE
- Communications : sockets, HTTP, *etc.*

# Objets distants : *Java* RMI (plus au programme)

## Architecture (simplifiée)

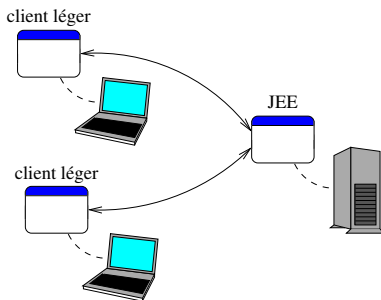


## Explications

- Clients lourds en *Java*
- Programmation : JDK JSE
- Communications : *Java* RMI

# JEE : *Java Enterprise Edition*

## Architecture

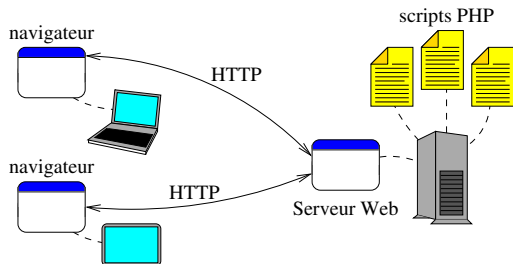


## Explications

- Clients légers (navigateur Internet)
- Programmation : JDK JEE
- Communications : HTTP

# PHP : *Personal Home Pages*

## Architecture classique

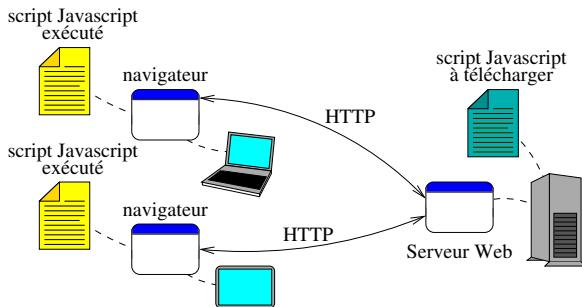


## Explications

- Clients légers (navigateur Internet)
- Programmation : PHP
- Communications : HTTP

# Javascript

## Architecture classique



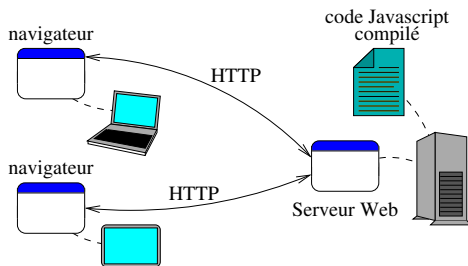
## Explications

- Clients capables d'exécuter le code mobile *Javascript*
- Script téléchargé puis exécuté sur le client



# Javascript côté serveur

## Architecture



## Explications

- Scripts côté serveur (généralement compilés)
- Moteur d'exécution côté serveur : par exemple *node.js*

# Table des matières

## 1 Introduction

- *Java*
- *PHP et Javascript*

## 2 Rappels sur Java

- Les classes en *Java*
- L'héritage
- La classe `Object`
- Les interfaces et les classes abstraites
- Les paquetages
- Les flux
- Les fichiers

## 3 PHP : *Personal Home Pages*

- Rappels sur PHP
- Gestion des formulaires
- Un peu plus loin

# Les classes (1/2)

- **Classe** : modèle décrivant...
  - ↪ ...des caractéristiques communes
  - ↪ ...des comportements communs d'un ensemble d'éléments
- **Objet** : instance d'une classe
  - ↪ Généré à partir de la classe
- **Membres** :
  - Attributs (données)
  - Méthodes
- En *Java*, tout est objet :
  - ↪ La fonction principale est située elle-même dans une classe

## Les classes (2/2)

### Structure générale d'une classe Java

```
public class Personne {  
    /* Attributs */  
    ...  
    /* Constructeurs */  
    ...  
    /* Getters/Setters */  
    ...  
    /* Autres méthodes */  
    ...  
}
```

- Classe “Personne”
- Contenue dans le fichier “Personne.java” :
  - ↪ **AVEC UNE MAJUSCULE AU DÉBUT**
  - ↪ **SANS ACCENT !!!**
- Une classe par fichier (sauf classées privées, etc.)

# Les attributs

- Par convention :
  - Déclarés au début de la classe
  - Généralement privés (non modifiables de l'extérieur)
  - Commencent par des minuscules avec des majuscules pour séparer les mots (exemple : unExempleDeNom)
- Déclaration :

Portée	Constante	Classe/Instance	Type	Nom	Initialisation
public	-	-	int		;
private	final	static	String	nom	= valeur;
protected			...		

# Les constructeurs (1/2)

- Objets instanciés à l'aide de constructeurs
- Méthodes particulières de la classe :
  - ↪ Pas de type de retour
  - ↪ Nom de la méthode correspondant au nom de la classe
- Fixe les valeurs de l'ensemble des attributs de l'objet
- Plusieurs sortes de constructeurs :
  - **Par défaut** : pas de paramètre
  - **Par initialisation** : un ou plusieurs paramètres
  - **Par copie** : un paramètre, référence vers un objet de la classe

## Les constructeurs (2/2)

```
public class Personne {  
    ...  
    /* Par défaut */  
    public Personne() {  
        prenom = "John";  
        nom = "Doe";  
        // ou this("John", "Doe");  
    }  
  
    /* Par initialisation */  
    public Personne(String prenom, String nom) {  
        this.prenom = prenom;  
        this.nom = nom;  
    }  
  
    /* Par copie */  
    public Personne(Personne p) {  
        prenom = p.prenom;  
        nom = p.nom;  
    }  
    ...  
}
```

# Les destructeurs

- Y'en a pas !



# Instanciación

- Instanciación = création d'un objet :
  - ↪ Utilisation de l'opérateur `new`
- L'allocation est dynamique (à l'exécution) :
  - ↪ Utilisation du constructeur approprié
  - ↪ Mécanisme de *lookup*

## Exemples

```
Personne p1 = new Personne();           /* Par défaut */  
Personne p2 = new Personne("Cyril", "Rabat"); /* Par init. */  
Personne p3 = new Personne(p2);         /* Par copie */
```

# Références et vie des objets (1/3)

## Les variables

- **Type de base** : contient la valeur
- **Tableau** : contient une référence vers les cases
- **Objet** : contient une référence vers l'objet

## Les références

- En *Java*, pas de pointeur !
  - ↪ Utilisation uniquement de références
- Suppression de l'objet/du tableau :
  - ↪ Lorsque plus aucune référence n'existe vers celui-ci
  - ↪ C'est le ramasse-miette qui s'en charge

## Références et vie des objets (2/3)

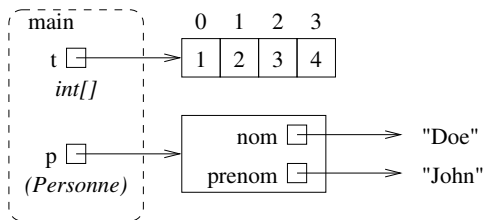
### Exemple

```
public static void test(int[] t, Personne p) {  
    t = new int[10];  
    p.setPrenom("Georges");  
}  
  
public static void main(String[] args) {  
    int[] t = {1, 2, 3, 4};  
    Personne p = new Personne("John", "Doe");  
    test(t, p);  
    // Affichage de t et p  
}
```

Quel est l'affichage obtenu ?

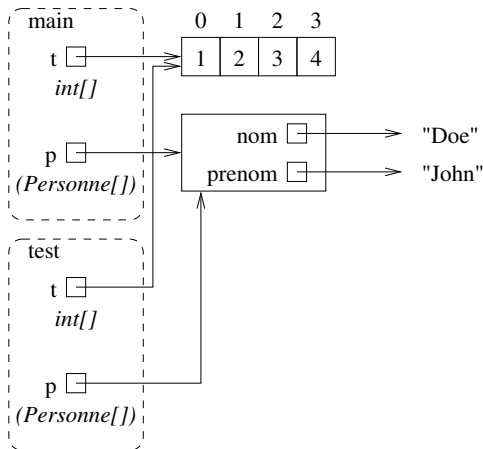
# Références et vie des objets (3/3)

## Illustration



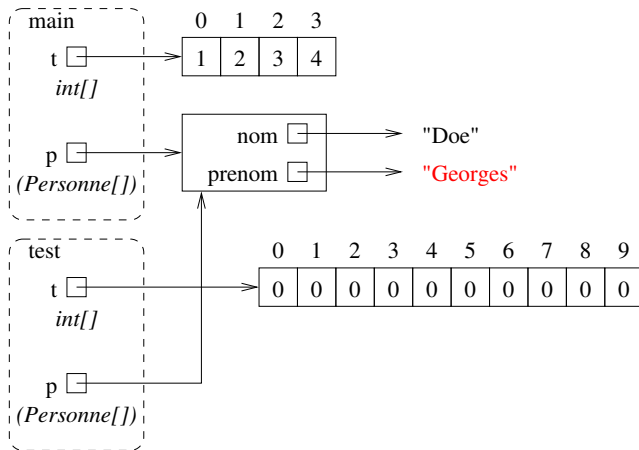
# Références et vie des objets (3/3)

## Illustration



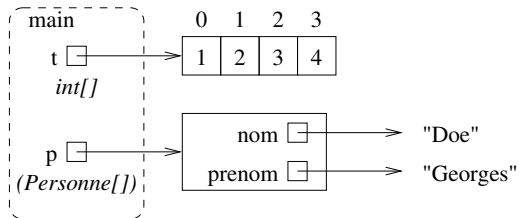
# Références et vie des objets (3/3)

## Illustration



# Références et vie des objets (3/3)

## Illustration



# Les méthodes (1/2)

## Description

- **Méthodes d'instance :**
  - ↪ Accès aux attributs d'instance ou de classe ...
  - ↪ ... peu importe le modificateur de portée
- **Méthodes de classe :**
  - ↪ Mot-clé `static`
  - ↪ Accès uniquement aux attributs/méthodes de classe ...
  - ↪ ... peu importe le modificateur de portée
- **Format général :**

Portée	Classe/Instance	Surcharge	Type	Nom	Code
public	-	-	void	nom	{ ... }
private	-	-	int		
protected	static	final	String		
			...		



## Les méthodes (2/2)

### Exemple de méthodes d'instance et de classe

```
public class Personne {  
    ...  
    public void afficher() {  
        System.out.println(nom + "_" + prenom);  
    }  
    public static void affichage(Personne p) {  
        p.afficher();  
    }  
}
```

### Exemple : la méthode principale

```
public class TestPersonne {  
    public static void main(String[] args) {  
        Personne p = new Personne();  
        p.afficher();  
        Personne.affichage(p);  
        p.affichage(p); // Ça marche ?  
    }  
}
```

## Les *getters/setters* (1/2)

- Méthodes “particulières” de par leur fonction uniquement
- **Getters** :
  - ↪ Fonction qui retourne la valeur de l'attribut
  - ↪ Un pour chaque attribut
  - ↪ Pas de modification
  - ↪ Nom : `getNomAttribut`
- **Setters** :
  - ↪ Procédure qui modifie la valeur de l'attribut
  - ↪ Nom : `setNomAttribut`

Quand doit-on écrire des *getters/setters* ?

## Les *getters/setters* (2/2)

### Exemple

```
public class Personne {  
    ...  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
    public void setPrenom(String prenom) {  
        this.prenom = prenom;  
    }  
    public String getNom() {  
        return nom;  
    }  
    public String getPrenom() {  
        return prenom;  
    }  
    ...  
}
```

# L'héritage (1/2)

## Description

- Objectifs multiples :
  - ↪ Partage du code
  - ↪ Réutilisabilité
  - ↪ Factorisation
- Relation de généralisation / spécialisation
- En *Java* : pas d'héritage multiple
- Utilisation du mot-clé `extends` :
  - ↪ Classe mère non `final` (héritage interdit le cas échéant)

## Exemple

```
public class Etudiant extends Personne {  
    ...  
}
```

## L'héritage (2/2)

### Transmission des membres

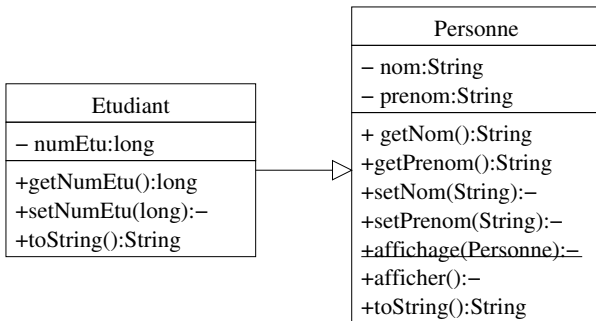
- Héritage de tous les membres
- `public` : accès total par la classe fille
- `private` : pas d'accès par la classe fille
- `protected` :
  - ↪ Accès comme s'ils étaient "`public`" pour la classe fille
  - ↪ Pas d'accès pour les autres classes

### Constructeurs dans la classe fille

- Appel à un constructeur de la classe mère via `super (...)` :
  - ↪ Première instruction du constructeur
  - ↪ Par défaut, appel au constructeur par défaut (s'il n'existe pas : erreur!)

## Exemple (1/2)

## Exemple



## Exemple (2/2)

### Extrait du code de la classe Etudiant

```
public class Etudiant extends Personne {  
  
    private int numEtu;  
  
    public Etudiant() {  
        // super(); => pas nécessaire  
        numEtu = -1;  
    }  
  
    public Etudiant(String prenom, String nom, int numEtu) {  
        super(prenom, nom);  
        this.numEtu = numEtu;  
    }  
  
    public Etudiant(Etudiant e) {  
        super(e);  
        numEtu = e.numEtu;  
    }  
    ...  
}
```

## Redéfinition de méthodes

- Possible de redéfinir une méthode existante dans la classe mère :  
↪ Sauf si la méthode est `final`
- Même signature que dans la classe mère
- Depuis la classe fille, possible d'appeler celle de la classe mère :  
↪ Instruction : `super.nomDeLaMethode()`
- De l'extérieur : seule la méthode redéfinie est accessible

### Extrait du code de la classe `Etudiant`

```
public class Etudiant extends Personne {  
    ...  
    @Override  
    public String toString() {  
        return super.toString() + "└(" + numEtu + ")";  
    }  
    ...  
}
```



# Typage statique et dynamique

## Typage statique vs typage dynamique

- **Typage statique** : type de la référence = type de l'objet

```
Personne p = new Personne();  
Etudiant e = new Etudiant();
```

- **Typage dynamique** : type de la référence = type plus spécifique

```
Personne p = new Etudiant();
```

## Exemples

```
Personne p = new Etudiant();  
p.setNom("Bob");
```

*// Ça marche ?*



# Typage statique et dynamique

## Typage statique vs typage dynamique

- **Typage statique** : type de la référence = type de l'objet

```
Personne p = new Personne();  
Etudiant e = new Etudiant();
```

- **Typage dynamique** : type de la référence = type plus spécifique

```
Personne p = new Etudiant();
```

## Exemples

```
Personne p = new Etudiant();  
p.setNom("Bob");  
p.setNumeroEtudiant(125464565);  
// Possible  
// Ça marche ?
```



# Typage statique et dynamique

## Typage statique vs typage dynamique

- **Typage statique** : type de la référence = type de l'objet

```
Personne p = new Personne();
Etudiant e = new Etudiant();
```

- **Typage dynamique** : type de la référence = type plus spécifique

```
Personne p = new Etudiant();
```

## Exemples

```
Personne p = new Etudiant();
p.setNom("Bob"); // Possible
p.setNumeroEtudiant(125464565); // Interdit
((Etudiant)p).setNumeroEtudiant(125464565); // Ça marche ?
```



# Typage statique et dynamique

## Typage statique vs typage dynamique

- **Typage statique** : type de la référence = type de l'objet

```
Personne p = new Personne();
Etudiant e = new Etudiant();
```

- **Typage dynamique** : type de la référence = type plus spécifique

```
Personne p = new Etudiant();
```

## Exemples

```
Personne p = new Etudiant();
p.setNom("Bob"); // Possible
p.setNumeroEtudiant(125464565); // Interdit
((Etudiant)p).setNumeroEtudiant(125464565); // Possible
```



# Polymorphisme

- Lors d'un appel de méthode sur une référence :  
 ↪ La méthode doit exister dans le type de la référence
- Si méthode redéfinie, appel à la méthode la plus spécifique

## Extrait de la classe Personne

```
public class Personne {
    ...
    public String toString() {
        return nom + " " + prenom;
    }
    ...
}
```

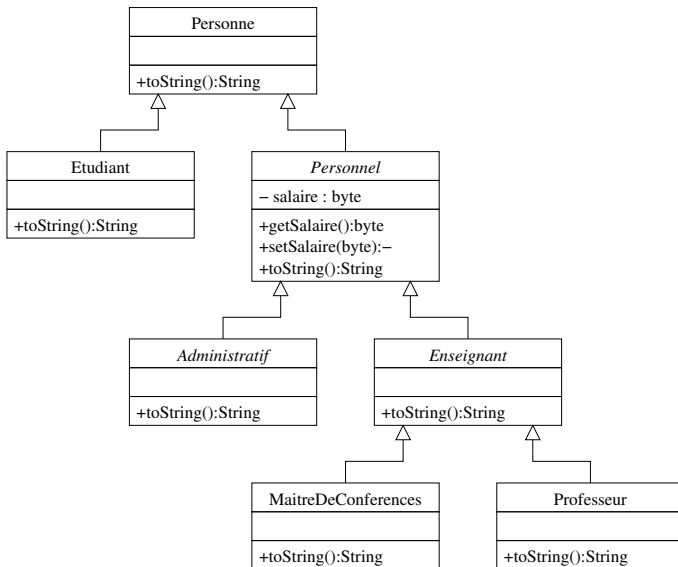
## Extrait de la classe Etudiant

```
public class Etudiant extends
    Personne {
    ...
    public String toString() {
        return super.toString() +
            "(" + numEtu + ")";
    }
    ...
}
```

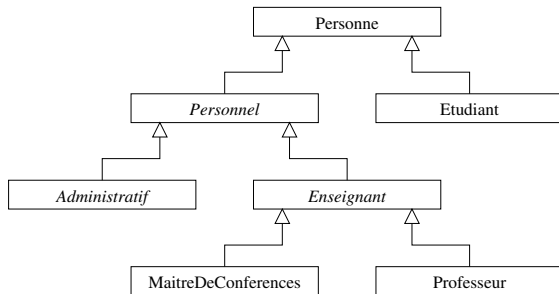
## Exemple de polymorphisme

```
Personne p = new Etudiant();
System.out.println(p.toString());
```

## Exemple (1/2)



## Exemple (2/2)



## Quelles méthodes sont appelées ?

```
Personne p1 = new Etudiant();
Enseignant e = new Professeur();
Personnel p2 = new MaitreDeConferences();

System.out.println(p1.toString());
System.out.println(e.toString());
System.out.println(p2.toString());
```

# La classe Object

- Classe mère de toutes les classes
- Plusieurs méthodes définies :
  - ↪ `public String toString()`
  - ↪ `protected Object clone()`
  - ↪ `public boolean equals(Object o)`
  - ↪ `public final Class<T> getClass()`
  - ↪ ...
- Nécessité de redéfinir la plupart d'entre elles



## Exemple : la méthode toString

- Conversion de l'objet en chaîne de caractères  
↪ Utile pour l'affichage
- Appel implicite lorsque la conversion est nécessaire

```
Personne p = new Personne();  
System.out.println(p.toString());  
System.out.println(p);
```

- Par défaut, pour les objets, tableaux :  
↪ Affichage du type et de l'adresse mémoire

## Exemple : la méthode equals

- Rappel : l'opérateur `==` compare les références et non le contenu
- Attention aux effets de bord liés au compilateur  
↳ Exemple des chaînes de caractères
- Pour comparer, utilisation de la méthode `equals`

```
@Override  
public boolean equals(Object obj)
```

- Relation réflexive, symétrique, transitive, consistante
- `x.equals(null)` retourne `false`

# Les interfaces

- Définie par le mot-clé `interface`
- Ne contient pas de données
  - ↪ Sauf des constantes (`public` et `final`)
- Contient uniquement des signatures de méthodes :
  - ↪ Pas de code (juste un `' ; '`)
- Pas d'instanciation :
  - ↪ Nécessite que des classes l'implémentent
  - ↪ Mot-clé `implements` après le nom de la classe

# Les classes abstraites

- Définie par le mot-clé `abstract`
- Classe “normale” mais non instanciable
- Peut contenir des méthodes abstraites
  - ↪ Utilisation du mot-clé `abstract`
- Pas d'instanciation :
  - ↪ Nécessite que des classes en héritent
  - ↪ Contient des constructeurs utilisables par les classes filles

## Remarque

Si une méthode d'une interface n'est pas implémentée dans la classe qui l'implémente, la classe doit être abstraite.

# Les paquetages (packages)

- Permettent de structurer le code
- Correspond à un groupement de classes :
  - ↪ Hiérarchisation à l'aide de répertoires/sous-répertoires
- Possible de rendre visible des classes uniquement pour le paquetage
  - ↪ Les classes doivent être déclarées `public` pour être accessibles
- Toutes les classes d'un paquetage doivent y être rattachées :
  - ↪ Première instruction : `package nomDuPackage;`
- Pour utiliser une classe d'un package :
  - ↪ `import nomDuPackage.NomDeLaClasse;`
- Pour utiliser toutes les classes :
  - ↪ `import nomDuPackage.*;`

## Organisation et nommage

- Les classes d'un paquetage sont dans un même répertoire
- Nom du répertoire = nom du paquetage
- Nécessité d'utiliser un nom unique :
  - ↪ Se baser sur un système de nommage universel
  - ↪ Exemple : `fr.univ-reims.info0503.personne`
- Généralement, les éditeurs créent des programmes avec un paquetage par défaut :
  - ↪ Utilisation du nom de l'application
  - ↪ Attention lors de la compilation en console !
- Le paquetage doit être accessible lors de la compilation :
  - ↪ Variable d'environnement `CLASSPATH`
  - ↪ Option de compilation `-cp` ou `-classpath`

## Paquetages et *Eclipse/NetBeans*

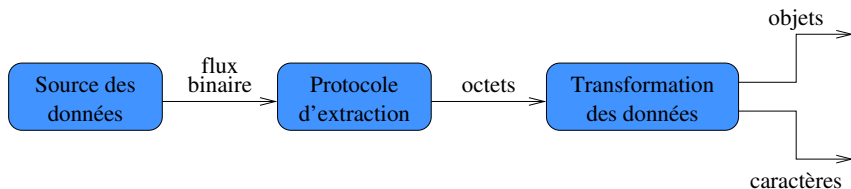
- Gestion automatique dans *Eclipse/NetBeans* :
  - ↪ Attention en cas de noms de classe identiques
- Importation de bibliothèques :
  - ↪ Attention au chemin lors de l'exportation du projet
  - ↪ Inclure dans le projet

### Attention

Testez votre projet sur une autre machine avant de l'envoyer !

## Présentation des flux (1/2)

- Canal d'échange d'information
- Possibilité d'“imbriquer” différents flux
- Contenus dans le package `java.io`





# Présentation des flux (2/2)

## Les différents types de flux

- Flux d'entrée (input) ou de sortie (output)
- Flux binaires "bruts" : terminent par "Stream"  
↪ `InputStream`, `OutputStream`...
- Flux de texte : terminent par "Reader" ou "Writer"  
↪ `InputStreamReader`, `OutputStreamWriter`...
- Flux de type buffered :  
↪ Utilisation d'un tampon pour réduire le nombre d'opérations
- D'une manière générale :  
↪ Le nom de la classe est suffisant pour comprendre sa fonction

# Entrées/sorties

- La classe “System” propose trois flux :
  - `System.err (PrintStream)` : sortie d'erreur “standard”
  - `System.in (InputStream)` : entrée “standard”
  - `System.out (PrintStream)` : sortie “standard”
- Écriture dans la console à l'aide des méthodes suivantes :
  - `System.out.print(...)`
  - `System.out.println(...)`
- Possibilité d'utiliser la méthode “console” :
  - Vérification si l'application est exécutée en console
  - Permet de formater la sortie (problèmes d'accents)
  - Peut être utilisée pour la lecture des mots de passe
- Pour la lecture :
  - Encapsulation de flux pour traiter les entrées
  - Par exemple : la classe “Scanner”

## La classe Scanner

- Permet de scanner un flux au format texte
- Possibilité de récupérer les types primitifs et les chaînes de caractères

### Exemple de lecture au clavier

```
Scanner sc = new Scanner(System.in);  
String s = sc.nextLine();  
System.out.println("Chaîne_lue:_ " + s);  
int i = sc.nextInt();  
System.out.println("Entier_lu:_ " + s);
```

### Remarque

Attention au “nextInt” qui ne lit pas le retour à la ligne !

# La sérialisation

- Transformation d'objets en flux d'octets :
  - ↪ Sauvegarde dans un fichier
  - ↪ Envoi via le réseau...
- La classe concernée doit implémenter l'interface `Serializable` :
  - ↪ Suffisant dans la plupart des cas
- Les classes pour manipuler des flux d'objets :
  - ↪ `ObjectInputStream`
  - ↪ `ObjectOutputStream`

## Exemple de déclaration de la classe

```
public class Personne implements Serializable {  
    ...  
}
```

# La classe `File`

- Ne représente pas un fichier mais un nom de fichier (ou de répertoire)
- Représentation abstraite indépendante du système de fichiers
- Peut être utilisé lors de l'ouverture des fichiers :
  - ↳ Évite les problèmes avec les “\” et “/”

# La lecture depuis un fichier

- Deux classes principales :
  - ↪ `FileReader` : flux de caractères
  - ↪ `FileInputStream` : flux d'octets

## Exemple

```
try {  
    FileInputStream fs = new FileInputStream("fichier.txt");  
    Scanner sc = new Scanner(fs);  
    while(sc.hasNext()) {  
        String s = sc.nextLine();  
        System.out.println(s);  
    }  
} catch(Exception e) {  
    System.err.println("Erreur_" + e);  
    System.exit(0);  
}
```

# L'écriture dans un fichier

- Deux classes principales :
  - ↪ `FileWriter` : flux de caractères
  - ↪ `FileOutputStream` : flux d'octets

## Exemple

```
try {  
    FileWriter f = new FileWriter("fichier.txt");  
    f.write("Bonjour\n");  
    f.write("tout_le\n");  
    f.write("monde_!!!\n");  
    f.close();  
} catch (Exception e) {  
    System.err.println("Erreur_" + e);  
    System.exit(0);  
}
```

# L'écriture d'objets

- Utilisation du flux `ObjectOutputStream`
- Méthode `writeObject`

## Exemple

```
try {
    Personne p = new Personne("Cyril", "Rabat");
    FileOutputStream fs = new FileOutputStream("fichier.bin");
    ObjectOutputStream oos = new ObjectOutputStream(fs);
    oos.writeObject(p);
    oos.flush();
    oos.close();
} catch (Exception e) {
    System.err.println("Erreur_" + e);
    System.exit(0);
}
```



## La lecture d'objets

- Utilisation du flux `ObjectInputStream`
- Méthode `readObject`  
↪ Lecture d'un `Object`

### Exemple

```
try {  
    FileInputStream fs = new FileInputStream("fichier.bin");  
    ObjectInputStream ois = new ObjectInputStream(fs);  
    Personne p = (Personne) ois.readObject();  
    ois.close();  
    System.out.println(p);  
} catch (Exception e) {  
    System.err.println("Erreur_" + e);  
    System.exit(0);  
}
```

## Autre exemple (1/5)

### Code

```
public static void main(String[] args) {  
    // Creation du tableau  
    Personne[] p = new Personne[6];  
    p[0] = new Personne();  
    p[1] = new Personne("Cyril", "Rabat");  
    p[2] = new Personne(p[0]);  
    p[3] = new Etudiant();  
    p[4] = new Etudiant("Cyril", "Rabat", 121010);  
    p[5] = new Etudiant((Etudiant)p[3]);  
  
    for(int i = 0; i < p.length; i++)  
        System.out.println(i + " : " + p[i]);  
}
```

## Autre exemple (2/5)

### Code (suite)

```
// Sauvegarde fichier
try {
    FileOutputStream fs = new FileOutputStream("fichier.bin");
    ObjectOutputStream oos = new ObjectOutputStream(fs);
    for(int i = 0; i < p.length; i++)
        oos.writeObject(p[i]);
    oos.flush();
    oos.close();
} catch(Exception e) {
    System.err.println("Erreur lors de l'écriture: " + e);
    System.exit(0);
}
```

## Autre exemple (3/5)

### Code (suite) : variante

```
// Sauvegarde fichier
try {
    FileOutputStream fs = new FileOutputStream("fichier.bin");
    ObjectOutputStream oos = new ObjectOutputStream(fs);
    oos.writeObject(p);
    oos.flush();
    oos.close();
} catch (Exception e) {
    System.err.println("Erreur lors de l'écriture: " + e);
    System.exit(0);
}
```

## Autre exemple (4/5)

### Code (fin)

```
// Lecture depuis le fichier
Personne[] p2 = new Personne[p.length];
try {
    FileInputStream fs = new FileInputStream("fichier.bin");
    ObjectInputStream ois = new ObjectInputStream(fs);
    for(int i = 0; i < p.length; i++)
        p2[i] = (Personne) ois.readObject();
    ois.close();
} catch(Exception e) {
    System.err.println("Erreur lors de la lecture: " + e);
    System.exit(0);
}
```

## Autre exemple (5/5)

### Code (fin) : variante

```
// Lecture depuis le fichier
Personne[] p2;
try {
    FileInputStream fs = new FileInputStream("fichier.bin");
    ObjectInputStream ois = new ObjectInputStream(fs);
    p2 = (Personne[]) ois.readObject();
    ois.close();
} catch (Exception e) {
    System.err.println("Erreur_lors_de_la_lecture:_:" + e);
    System.exit(0);
}
```

# Table des matières

## 1 Introduction

- *Java*
- PHP et *Javascript*

## 2 Rappels sur *Java*

- Les classes en *Java*
- L'héritage
- La classe `Object`
- Les interfaces et les classes abstraites
- Les paquetages
- Les flux
- Les fichiers

## 3 PHP : *Personal Home Pages*

- Rappels sur PHP
- Gestion des formulaires
- Un peu plus loin

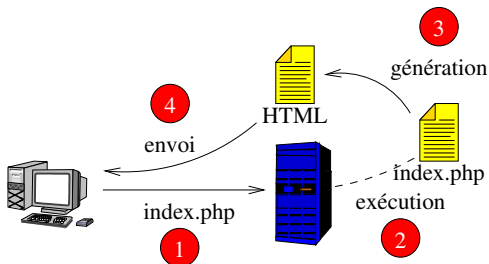
# Le PHP

- PHP pour *Personal Home Pages*
- Code inclus dans du HTML (vision réductrice !)
- Exécution du code sur le serveur
- Permet d'accéder aux ressources du serveur :
  - Les fichiers
  - Les bases de données
  - Les sessions. . .
- Exécution dans le contexte du serveur Web
- Extension des fichiers `.php` :
  - ↪ Permet d'identifier le code à exécuter par le moteur PHP



# Utilisation du PHP dans du HTML

- Code inclus en tout point du HTML
- Utilisation des balises `<?php` et `?>` :
  - ↪ Le code PHP est interprété par le serveur
  - ↪ Remplacement du code par le résultat (affiché via `echo`)
- Pas de PHP sur le client !



# Syntaxe

- Proche du C/C++/Java
- Utilisable en ligne de commandes ou dans le contexte du serveur Web
- Syntaxe très riche :
  - Nombreux opérateurs
  - Beaucoup de souplesse (scripting...)
- Utilisation de la POO :
  - Pas obligatoire
  - Permet cependant une structuration du code
- API très riche
- Nombreux frameworks, bibliothèques, etc.

## Les versions

- Actuellement, version 7.4
  - ↪ Plusieurs *releases* corrigeant différents bugs
- Apporte une rigueur sur la syntaxe :
  - ↪ Certaines pratiques deviennent interdites !
- Ajout d'opérateurs (pas nécessaires, mais simplification du code)
  - ↪ Exemple : ?? (*null coalescing*) et <=> (comparaison)
- Accélérations diverses, optimisation mémoire...
- Depuis la version 7.3, typage des paramètres ET du retour
  - ↪ Pour retourner un tableau : `array`
  - ↪ Pour retourner un tableau ou `null` : `?array`

### Attention

Les codes écrits dans les dernières versions ne sont pas compatibles avec les versions précédentes !

En INFO0503 : typage obligatoire !

## Un exemple

### Contenu du fichier `index.php`

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <title>Exemple</title>
  </head>
  <body>
    <h1>Bonjour</h1>
    <p>Aujourd'hui, nous sommes le <?php echo date("d/m/Y"); ?>.</p>
  </body>
</html>
```

### Exécution

**Bonjour**

Aujourd'hui, nous sommes le 30/08/2020.

# Exécuter du PHP

- Contrairement au HTML/CSS : nécessite un serveur Web  
    ↪ Wamp (pour Windows, Apache, MySQL, PHP), Lamp (Linux), Xamp, etc.
- Installation d'une distribution : *Wamp*, etc.
- Fichiers placés dans le répertoire `www` (répertoire d'installation)  
    ↪ Création d'un sous-répertoire `CM01` (par exemple)
- Pour visualiser :
  - Démarrer les serveurs (automatisé avec *Wamp*)
  - Ouvrez un navigateur
  - Saisissez l'adresse : `http://localhost/CM01/index.php`

# Variables

- Utilisation du \$ pour indiquer le nom de la variable  
↪ Nom de variable : lettres, chiffres (sauf au début), \_
- Noms sensibles à la casse (préférez les minuscules)
- Pas de déclaration au préalable :  
↪ Une variable peut changer de type à tout moment
- 10 types basiques :
  - Types scalaires : boolean, integer, float, string
  - Types composés : array, object, callable, iterable
  - Types spéciaux : resource, null

## Exemple

```
<?php
$a = 12;
echo "La_valeur_de_a_=" . $a;
```

## Fonctions utiles pour les variables

- `isset` : teste si une variable existe  
↪ Retourne `true` ou `false` (booléen)
- `empty` : teste si une variable est vide  
↪ Retourne `true` si une chaîne n'existe pas ou est égale à ""  
↪ Pour les autres types : 0 pour un entier, `NULL`, "0", *etc.*
- Pour supprimer une variable : `unset`
- `gettype` : retourne le type de la variable  
↪ `settype` permet de spécifier le type d'une variable
- `is_int` : teste si la variable est un entier  
↪ Idem avec `is_string`, `is_float`, `is_double`
- Conversions : `floatval`, `intval`, *etc.*

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12) ?`
- `is_int("12")`
- `is_int(12.5)`
- `intval("12Baba");`
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`



## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`
- `is_int(12.5)`
- `intval("12Baba");`
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")` ?
- `is_int(12.5)`
- `intval("12Baba");`
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`
- `intval("12Baba");`
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a ='.$a;`
- `$a = 0; echo "$a =".$a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)` ?
- `intval("12Baba");`
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");` ?
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");` ?
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`



## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");` ?
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a ='.$a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`    **12**
- `floatval("12.5");`
- `floatval("12,5");`
- `$a = 0; echo '$a ='.$a;`
- `$a = 0; echo "$a =".$a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`    **12**
- `floatval("12.5");` ?
- `floatval("12,5");`
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`    **12**
- `floatval("12.5");`    **12.5**
- `floatval("12,5");`
- `$a = 0; echo '$a ='.$a;`
- `$a = 0; echo "$a =".$a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`    **12**
- `floatval("12.5");`    **12.5**
- `floatval("12,5");` ?
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`    **12**
- `floatval("12.5");`    **12.5**
- `floatval("12,5");`    **12**
- `$a = 0; echo '$a =' . $a;`
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)` **true**
- `is_int("12")` **false**
- `is_int(12.5)` **false**
- `intval("12Baba");` **12**
- `intval("--12--");` **0**
- `intval("12.5");` **12**
- `floatval("12.5");` **12.5**
- `floatval("12,5");` **12**
- `$a = 0; echo '$a =' . $a; ?`
- `$a = 0; echo "$a =" . $a;`



## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`    **12**
- `floatval("12.5");`    **12.5**
- `floatval("12,5");`    **12**
- `$a = 0; echo '$a =' . $a;`    **\$a = 0**
- `$a = 0; echo "$a =" . $a;`

## Quelques tests (1/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `is_int(12)`    **true**
- `is_int("12")`    **false**
- `is_int(12.5)`    **false**
- `intval("12Baba");`    **12**
- `intval("--12--");`    **0**
- `intval("12.5");`    **12**
- `floatval("12.5");`    **12.5**
- `floatval("12,5");`    **12**
- `$a = 0; echo '$a = ' . $a;`    **\$a = 0**
- `$a = 0; echo "$a =" . $a; ?`    **0 = 0**

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b); ?`
- `$b = ""; echo empty($b);`
- `echo empty($c);`
- `$c = "0"; echo empty($c);`
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);` 1
- `$b = ""; echo empty($b);`
- `echo empty($c);`
- `$c = "0"; echo empty($c);`
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);` 1
- `$b = ""; echo empty($b);` ?
- `echo empty($c);`
- `$c = "0"; echo empty($c);`
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);`    1
- `$b = ""; echo empty($b);`    1
- `echo empty($c);`
- `$c = "0"; echo empty($c);`
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);`    1
- `$b = ""; echo empty($b);`    1
- `echo empty($c);` ?
- `$c = "0"; echo empty($c);`
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);` 1
- `$b = ""; echo empty($b);` 1
- `echo empty($c);` 1
- `$c = "0"; echo empty($c);`
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`



## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);` 1
- `$b = ""; echo empty($b);` 1
- `echo empty($c);` 1
- `$c = "0"; echo empty($c);` ?
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);` 1
- `$b = ""; echo empty($b);` 1
- `echo empty($c);` 1
- `$c = "0"; echo empty($c);` 1
- `$val = "10";  
settype($val, "int");  
var_dump($val);`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);`    1
- `$b = ""; echo empty($b);`    1
- `echo empty($c);`    1
- `$c = "0"; echo empty($c);`    1
- `$val = "10";  
settype($val, "int");  
var_dump($val);  
?`
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);`    1
- `$b = ""; echo empty($b);`    1
- `echo empty($c);`    1
- `$c = "0"; echo empty($c);`    1
- `$val = "10";  
settype($val, "int");  
var_dump($val);`  
int(10)
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);`    1
- `$b = ""; echo empty($b);`    1
- `echo empty($c);`    1
- `$c = "0"; echo empty($c);`    1
- `$val = "10";  
settype($val, "int");  
var_dump($val);`  
int(10)
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`  
?

## Quelques tests (2/2)

Quel est l'affichage ou la valeur des instructions suivantes ?

- `$b = ""; echo isset($b);`    1
- `$b = ""; echo empty($b);`    1
- `echo empty($c);`    1
- `$c = "0"; echo empty($c);`    1
- `$val = "10";  
settype($val, "int");  
var_dump($val);`  
int(10)
- `$val = "popo";  
settype($val, "int");  
var_dump($val);`  
int(0)

# Généralités sur les tableaux

- Permettent de regrouper des données différentes sous un même nom
- En PHP : cartes ordonnées
  - À une clé, on associe une valeur
  - L'ordre d'ajout est conservé
- Utilisation du mot-clé : `array`  
↔ Syntaxe courte : crochets
- Pour accéder à un élément / le modifier :  
↔ Utilisation des crochets

## Exemple

```
<?php
$tab = array(1, 2, "Chaine");
var_dump($tab);
echo $tab[0];
```

## Exemple (sans array)

```
<?php
$tab = [1, 2, "Chaine"];
var_dump($tab);
echo $tab[0];
```

## Tableaux associatifs

- Pour créer un tableau associatif, utilisation de "=>"
- Une clé = une chaîne de caractères ou un entier
- Accéder à la valeur associée à une clé : `$tab["cle"]` (ou `$tab['cle']`)
- Pour vérifier l'existence d'une clé : `array_key_exists`
- Pour supprimer un élément : `unset`

### Exemple

```
<?php
$tab = array("prenom" => "Cyril", "nom" => "Rabat", "age" => 35);

echo $tab["prenom"]." ".$tab["nom"]." ".$tab["age"]." _an(s)";
```



# Exercices pour se détendre (1/2)

## Exercice 1

Soit le script PHP suivant :

```
<?php
$a = [ 0, 1, 2 ];
$b = [ 1, 0, 2 ];

if($a == $b) echo "Vrai_1<br/>";
if($a === $b) echo "Vrai_2<br/>";
```

Quel est l'affichage obtenu ?



# Exercices pour se détendre (1/2)

## Exercice 1

Soit le script PHP suivant :

```
<?php
$a = [ 0, 1, 2 ];
$b = [ 1, 0, 2 ];

if($a == $b) echo "Vrai_1<br/>";
if($a === $b) echo "Vrai_2<br/>";
```

Quel est l'affichage obtenu ?

## Solution

- Aucun affichage
- Les deux tableaux ont les mêmes valeurs mais avec des clés différentes !



## Exercices pour se détendre (2/2)

### Exercice 2

Soit le script PHP suivant :

```
<?php
$a = [ 'nom' => 'Rabat', 'prenom' => 'Cyril', 'age' => 40 ];
$b = [ 'prenom' => 'Cyril', 'nom' => 'Rabat', 'age' => 40 ];

if($a == $b) echo "Vrai_1<br/>";
if($a === $b) echo "Vrai_2<br/>";
```

Quel est l'affichage obtenu ?



## Exercices pour se détendre (2/2)

### Exercice 2

Soit le script PHP suivant :

```
<?php
$a = [ 'nom' => 'Rabat', 'prenom' => 'Cyril', 'age' => 40 ];
$b = [ 'prenom' => 'Cyril', 'nom' => 'Rabat', 'age' => 40 ];

if($a == $b) echo "Vrai_1<br/>";
if($a === $b) echo "Vrai_2<br/>";
```

Quel est l'affichage obtenu ?

### Solution

- Solution : Vrai 1
- L'opérateur '===' teste également l'ordre des éléments



# Les classes en PHP

- Définition proche du *Java*
- Respectez la conception orientée objet
  - ↪ Modificateurs de portée sur les attributs et les méthodes
- Possède toutes les fonctionnalités classiques :
  - ↪ Héritage, polymorphisme, surcharge, redéfinition, ...
- Propose les *traits* :
  - ↪ Permet de réutiliser du code dans des classes indépendantes

En INFO0503, usage de la POO obligatoire !

## Un autre exemple : une classe PHP (1/3)

### Contenu du fichier MaClasse.php - pas beau

```
<?php
class MaClasse {
    private $a;
    private static $b = 0;

    public function __construct($a) {
        $this->a = $a;
        self::$b++;
    }
    public function getA() {
        return $this->a;
    }
    public static function getB() {
        return self::$b;
    }
    public function __toString() {
        return "MaClasse_:a=$this->a;b=".self::$b;
    }
}
```

## Un autre exemple : une classe PHP (2/3)

### Contenu du fichier MaClasse.php - vérification des types

```
<?php
class MaClasse {
    private $a;
    private static $b = 0;

    public function __construct(int $a) {
        $this->a = $a;
        self::$b++;
    }
    public function getA() : int {
        return $this->a;
    }
    public static function getB() : int {
        return self::$b;
    }
    public function __toString() : string {
        return "MaClasse_:a=$this->a;b=".self::$b;
    }
}
```

## Un autre exemple : une classe PHP (3/3)

### Contenu du fichier test.php

```
<?php
require_once("MaClasse.php");

$objet = new MaClasse(3);
echo $objet;
```



## Un exemple d'héritage (1/2)

### Contenu du fichier MaClasseFille.php

```
<?php
class MaClasseFille extends MaClasse {

    private $c;

    public function __construct(int $a, string $c) {
        parent::__construct($a);
        $this->c = $c;
    }

    public function getC() : string {
        return $this->c;
    }

    public function __toString() : string {
        return "MaClasseFille_␣a=".self::getA().";␣b=".self::getB().
            ";␣c=$this->c";
    }
}
```

## Un exemple d'héritage (2/2)

### Contenu du fichier `test.php`

```
<?php
require_once("MaClasse.php");
require_once("MaClasseFille.php");

$objet1 = new MaClasse(1);
$objet2 = new MaClasse(2);
$objet3 = new MaClasseFille(3, "toto");

echo $objet1."<br/>";
echo $objet2."<br/>";
echo $objet3."<br/>";
```

## Un exemple de trait (1/2)

```
trait Exemple {  
    public function __toString() : string {  
        return $this->nom;  
    }  
}  
  
class Chien {  
    use Exemple;  
    private $nom;  
    public function __construct(string $nom) {  
        $this->nom = $nom;  
    }  
}  
  
class Ville {  
    use Exemple;  
    private $nom;  
    public function __construct(string $nom) {  
        $this->nom = $nom;  
    }  
}
```

# Les formulaires HTML

- Permettent d'envoyer des données au serveur
- Deux méthodes principales (voir le cours sur HTTP) :
  - ↪ POST et GET
- Données récupérées depuis le script PHP :
  - ↪ Variables globales `$_POST[]` et/ou `$_GET[]`
  - ↪ Tableaux associatifs

## Remarques

- Les éléments des formulaires ont des comportements différents
  - ↪ Pour un `checkbox`, `$_POST['nom']` n'existe pas si non coché !
- Attention aux données récupérées depuis les formulaires
  - ↪ Même pour un `select` !

## Exemple (1/2)

### Le formulaire HTML

```
<form action="log.php" method="post">  
  <label for="login">Login :</label>  
  <input id="login" name="login" type="text"/>  
  <label for="motDePasse">Mot de passe :</label>  
  <input id="motDePasse" name="motDePasse" type="password"/>  
  <button type="submit">Connexion</button>  
</form>
```

## Exemple (2/2)

### Le traitement en PHP

```
...
<?php
if(!isset($_POST['login']) || ($_POST['login'] == "")) {
    echo "Vous devez spécifier un login!";
}
if(!isset($_POST['motDePasse']) || ($_POST['motDePasse'] == "")) {
    echo "Vous devez spécifier un mot de passe!";
}
?>
...
```

## Un peu de sécurité

- Toujours vérifier les valeurs saisies dans les formulaires  
↪ Vérification des types, conversions
- Attention à l'usage de `isset` et `empty` !
- Rappel : comparaison en PHP avec le type `===`

### Exemples

```
$a = "2";  
$b = 2;  
if($a == $b) echo "Cool";           // Ça affiche Cool ?
```



## Un peu de sécurité

- Toujours vérifier les valeurs saisies dans les formulaires  
↪ Vérification des types, conversions
- Attention à l'usage de `isset` et `empty` !
- Rappel : comparaison en PHP avec le type `===`

### Exemples

```
$a = "2";  
$b = 2;  
if($a == $b) echo "Cool";           // Oui  
if(+$a === $b) echo "Cool";        // Ça affiche Cool ?
```





## Un peu de sécurité

- Toujours vérifier les valeurs saisies dans les formulaires  
↪ Vérification des types, conversions
- Attention à l'usage de `isset` et `empty` !
- Rappel : comparaison en PHP avec le type `===`

### Exemples

```
$a = "2";  
$b = 2;  
if($a == $b) echo "Cool";           // Oui  
if(+$a === $b) echo "Cool";        // Oui
```



## Quelques précisions

- Le langage PHP possède de nombreuses subtilités :
  - ↪ Gestion des variables, inclusion de scripts, *etc.*
  - ↪ Pas de cours spécifique (voir TP 2)
- Permet de modifier l'en-tête HTTP
- Gestion des sessions, cookies

### Un programmeur Web doit au minimum...

- Maîtriser le HTTP (en-tête, modes, cookie)
- Savoir configurer un serveur Web
- Maîtriser les langages du Web (HTML, CSS, *Javascript*)
- Comprendre comment éviter les principales failles de sécurité

# Exercice

# Constructeur automobile

## Modélisation à réaliser

- Gestion de la production de véhicules
- Un véhicule est caractérisé par :
  - Numéro d'identification unique (lettres et chiffres)
  - Moteur (identifiant, carburation, puissance)
  - Date de fabrication
  - Couleur (unie)
  - Options : vitres électriques, radar de recul, GPS, armement divers, etc.
- Plusieurs modèles de véhicules :
  - Choix dans un catalogue
  - Production d'un véhicule suivant un modèle
- Véhicules produits dans différentes usines, composées de parkings  
↔ Nécessité de connaître l'emplacement exact de chaque véhicule