TP3 JAVA – Programmation Orientée Objets

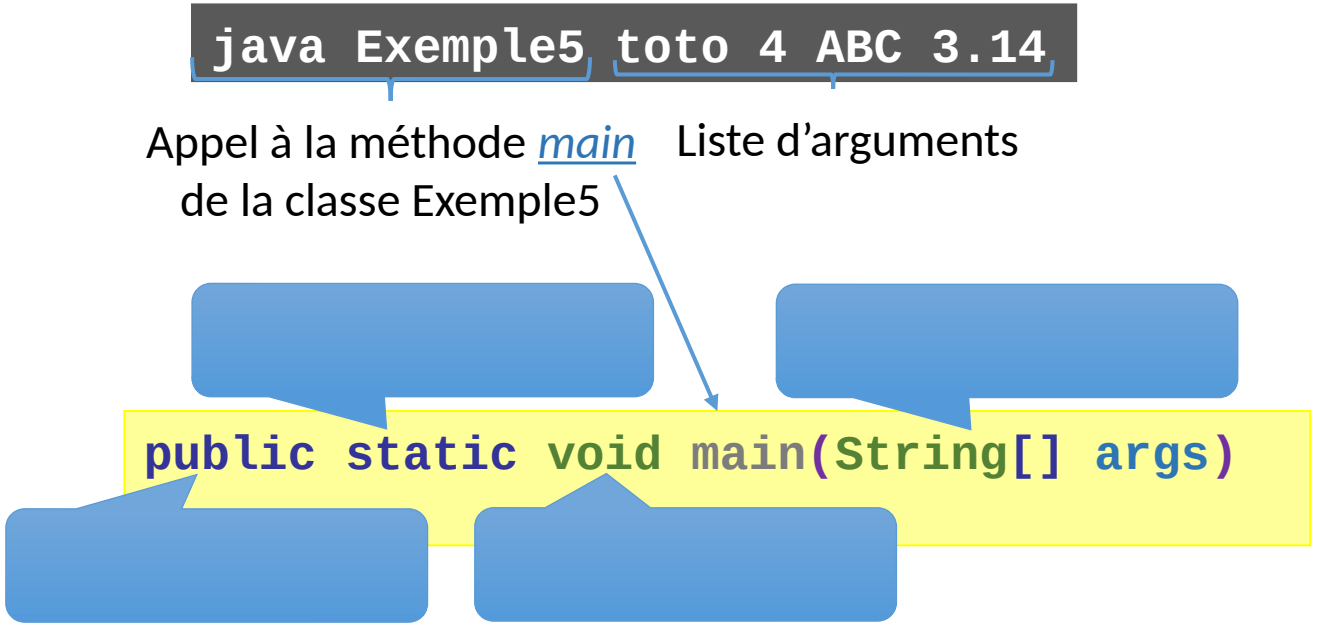
Rafael COLARES

Maitre de Conférences ISIMA

Bureau D104 – email:rafael.colares_borges@uca.fr

Leçons apprises du TP2

- Exécution avec arguments en ligne de commande



Leçons apprises du TP2

- Exécution avec arguments en ligne de commande

```
java Exemple5 toto 4 ABC 3.14
```

Appel à la méthode *main* de la classe Exemple5

Liste d'arguments

Pas besoin de créer un objet de la classe pour pouvoir l'appeler

Récupère les arguments passés en ligne de commande

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

La méthode doit être visible de l'extérieur de la classe

Pas de retour

`args[0]`	« toto »
`args[1]`	« 4 »
`args[2]`	« ABC »
`args[3]`	« 3.14»

Si l'on veut utiliser d'autre type que String en argument ?

Leçons apprises du TP2

- Exécution avec arguments en ligne de commande

```
java Exemple5 toto 4 ABC 3.14
```

Appel à la méthode *main* de la classe Exemple5

Liste d'arguments

Pas besoin de créer un objet de la classe pour pouvoir l'appeler

Récupère les arguments passés en ligne de commande sous la forme d'un tableau de Strings

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

La méthode doit être visible de l'extérieur de la classe

Pas de retour

`args[0]`	« toto »
`args[1]`	« 4 »
`args[2]`	« ABC »
`args[3]`	« 3.14»

Si l'on veut utiliser d'autre type que String en argument ? Parse !

```
double pi =Double.parseDouble(args[3]);
int  four = Integer.parseInt(args[1]);
```

Leçons apprises du TP2

- Exécution avec arguments en ligne de commande
- Initialisation d'un attribut de classe
  - À la déclaration

```
class Liste {
	static int[] liste = new int [10];
}
```

Les 10 éléments du tableau sont initialisés à leur valeur par défaut

  - Dar

```
class Liste {
	static int[] liste;
	static {
		liste = new int [10];
		...
	}
}
```

- Le bloc statique est appelé au chargement de la Classe
- Permet une initialisation plus customisé

Leçons apprises du TP2

- Exécution avec arguments en ligne de commande
- Initialisation d'un attribut de classe
  - À la déclaration

```
class Liste {
        static int[] liste = new
}
```

  - Dans un bloc statique

```
class Liste {
        static int[] liste;
        static {
                liste = new int [1
                ...
        }
}
```

  - Via une méthode de classe d'initialisation customisé

```
class Liste {
        static int[] liste = initListe(10);

        static int[] initListe(int n){
                int[] maListe = new int [n];
                ...
                return maListe;
        }
}
```

Comment choisir ?

Le

- Ex
- Ini
  -
  -

e

```
initListe(10);

e(int n){
 = new int [n];

e;
```

```
        ...
    }
}
```

Comment choisir ?

Leçons apprises du TP2

- Exécution avec arguments en ligne de commande
- Initialisation d'un attribut de classe
- Passage de paramètres
  - ***En Java, le passage se fait TOUJOURS par copie !!!***

```
public class PassByValue {
	public static void main(String[] args) {
		int i = 0;
		increase(i);
		System.out.println(i);
	}
	public static void increase(int n) {
		n = n + 1;
	}
}
```

```
$ javac PassByValue.java
$ java PassByValue

                ?
```

a) Erreur de compilation
b) Erreur d'exécution
c) 0
d) 1

Leçons apprises du TP2

- Exécution avec arguments en ligne de commande
- Initialisation d'un attribut de classe
- Passage de paramètres
  - ***En Java, le passage se fait TOUJOURS par copie !!!***

```
public class PassByValue {
	public static void main(String[] args) {
		int i = 0;
		increase(i);
		System.out.println(i);
	}
	public static void increase(int n) {
		n = n + 1;
	}
}
```

```
$ javac PassByValue.java
$ java PassByValue
0
```

- Quand la méthode `increase` est appelé, la valeur de `i` est copié et stocké dans la variable locale `n`
- La valeur de `n` est modifié mais pas celle de `i`

```
class ZZ2{
	private int javaKnowledge;

	public void setJavaKnowledge(int javaKnowledge) {
	    this.javaKnowledge = javaKnowledge;
	    }
	    public int getJavaKnowledge() {
	    return javaKnowledge;
	    }
}

public class PassByValue {
	public static void main(String[] args) {
		ZZ2 student = new ZZ2();
		study(student);
		System.out.println(student.getJavaKnowledge());
	}
	public static void study(ZZ2 seriousStudent) {
		seriousStudent.setJavaKnowledge(seriousStudent.getJavaKnowledge()+1);
	}
}
```

```
class ZZ2{
	private int javaKnowledge;

	public void setJavaKnowledge(int javaKnowledge) {
	    this.javaKnowledge = javaKnowledge;
	    }
	    public int getJavaKnowledge() {
	    return javaKnowledge;
	    }
}

public class PassByValue {
	public static void main(String[] args) {
		ZZ2 student = new ZZ2();
		study(student);
		System.out.println(student.getJavaKnowledge());
	}
	public static void study(ZZ2 seriousStudent) {
		seriousStudent.setJavaKnowledge(seriousStudent.getJavaKnowledge()+1);
	}
}
```

```
$ javac PassByValue.java
$ java PassByValue

                ?
```

a) Erreur de compilation
b) Erreur d'exécution
c) 0
d) 1
e) null

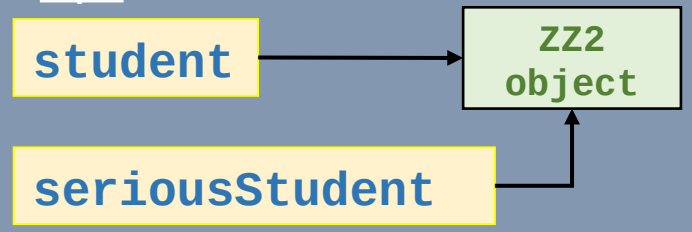
```
class ZZ2{
	private int javaKnowledge;

	public void setJavaKnowledge(int javaKnowledge) {
	    this.javaKnowledge = javaKnowledge;
	    }
	    public int getJavaKnowledge() {
	    return javaKnowledge;
	    }
}

public class PassByValue {
	public static void main(String[] args) {
		ZZ2 student = new ZZ2();
		study(student);
		System.out.println(student.getJavaKnowledge());
	}
	public static void study(ZZ2 seriousStudent) {
		seriousStudent.setJavaKnowledge(seriousStudent.getJavaKnowledge()+1);
	}
}
```

```
$ javac PassByValue.java
$ java PassByValue
1
```

- Quand la méthode `study` est appelé, la valeur de `student` est copié et stocké dans la variable locale `seriousStudent`
- La variable `student` est une ***référence***!
- La variable locale `seriousStudent` est une **copie** de la référence `student`



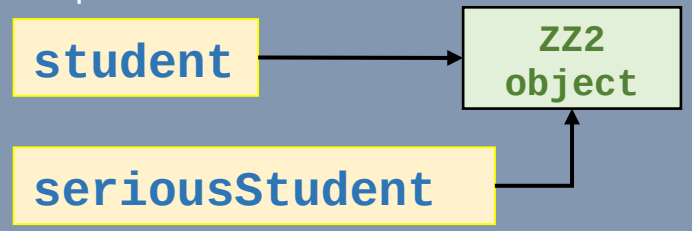
```
class ZZ2{
	private int javaKnowledge;

	public void setJavaKnowledge(int javaKnowledge) {
	    this.javaKnowledge = javaKnowledge;
	    }
	    public int getJavaKnowledge() {
	    return javaKnowledge;
	    }
}

public class PassByValue {
	public static void main(String[] args) {
		ZZ2 student = new ZZ2();
		study(student);
		System.out.println(student.getJavaKnowledge());
	}
	public static void study(ZZ2 seriousStudent) {
		seriousStudent.setJavaKnowledge(seriousStudent.getJavaKnowledge()+1);
	}
}
```

```
$ javac PassByValue.java
$ java PassByValue
1
```

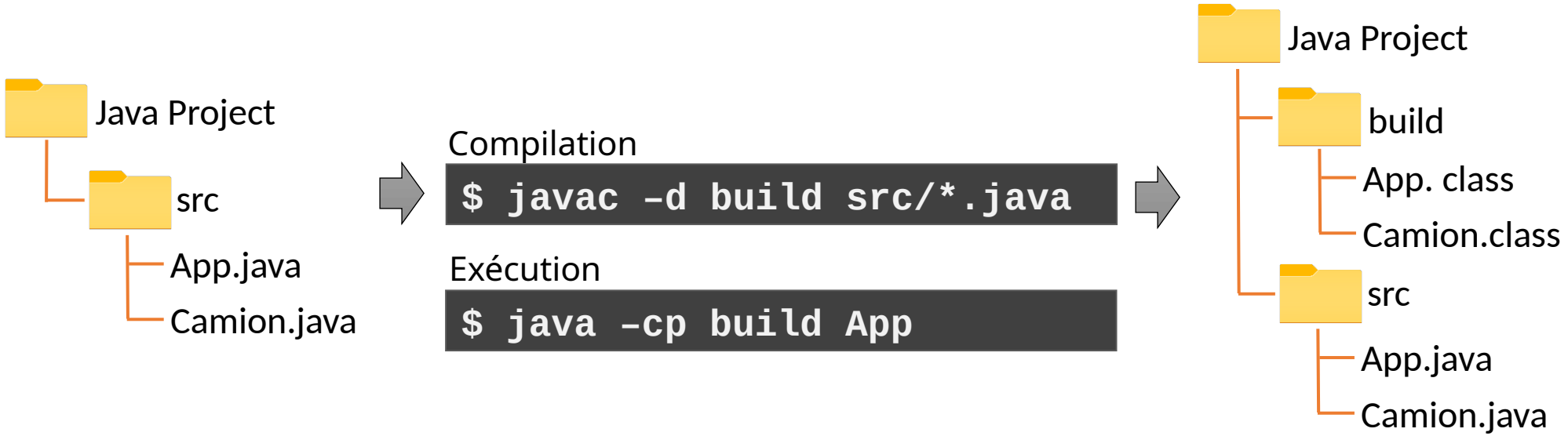
- Quand la méthode `study` est appelé, la valeur de `student` est copié et stocké dans la variable locale `seriousStudent`
- La variable `student` est une ***référence***!
- La variable locale `seriousStudent` est une copie de la référence `student`



En Java, le passage se fait TOUJOURS par copie !!!

Leçons apprises du TP2

- Organisation du projet
  - Séparation code source / builds



Leçons apprises du TI

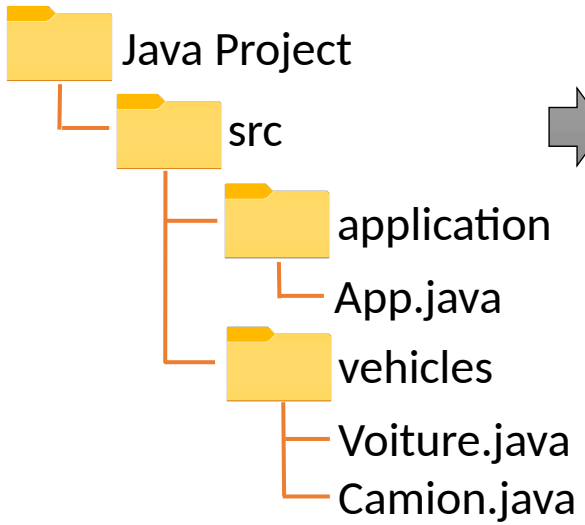
- Organisation du projet
  - Séparation code source / builds
  - Organisation en packages

App.java

```
package application;
import vehicles.Voiture;
public class App {
        public static void main(String[] args) {
                // code ...
        }
}
```

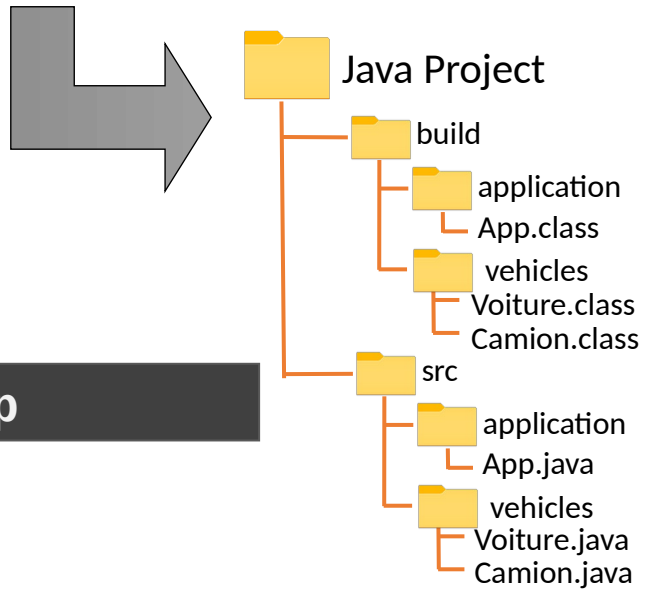
Voiture.java

```
package vehicles;
public class Voiture {
        // code ...
}
```



Compilation

```
$ javac -d build src/application/App.java src/vehicles/*.java
```



Exécution

```
$ java -cp build application.App
```

TP 3

- Contenu:
  - ✓ Héritage
  - ✓ Polymorphisme

- Aller voir *https://perso.isima.fr/loic/java/tp_03.php*
  - Attention à la Section 2 (contructeurs + héritage)