



LETS LEARN JAVA

BUILDING BLOCKS OF OOPS

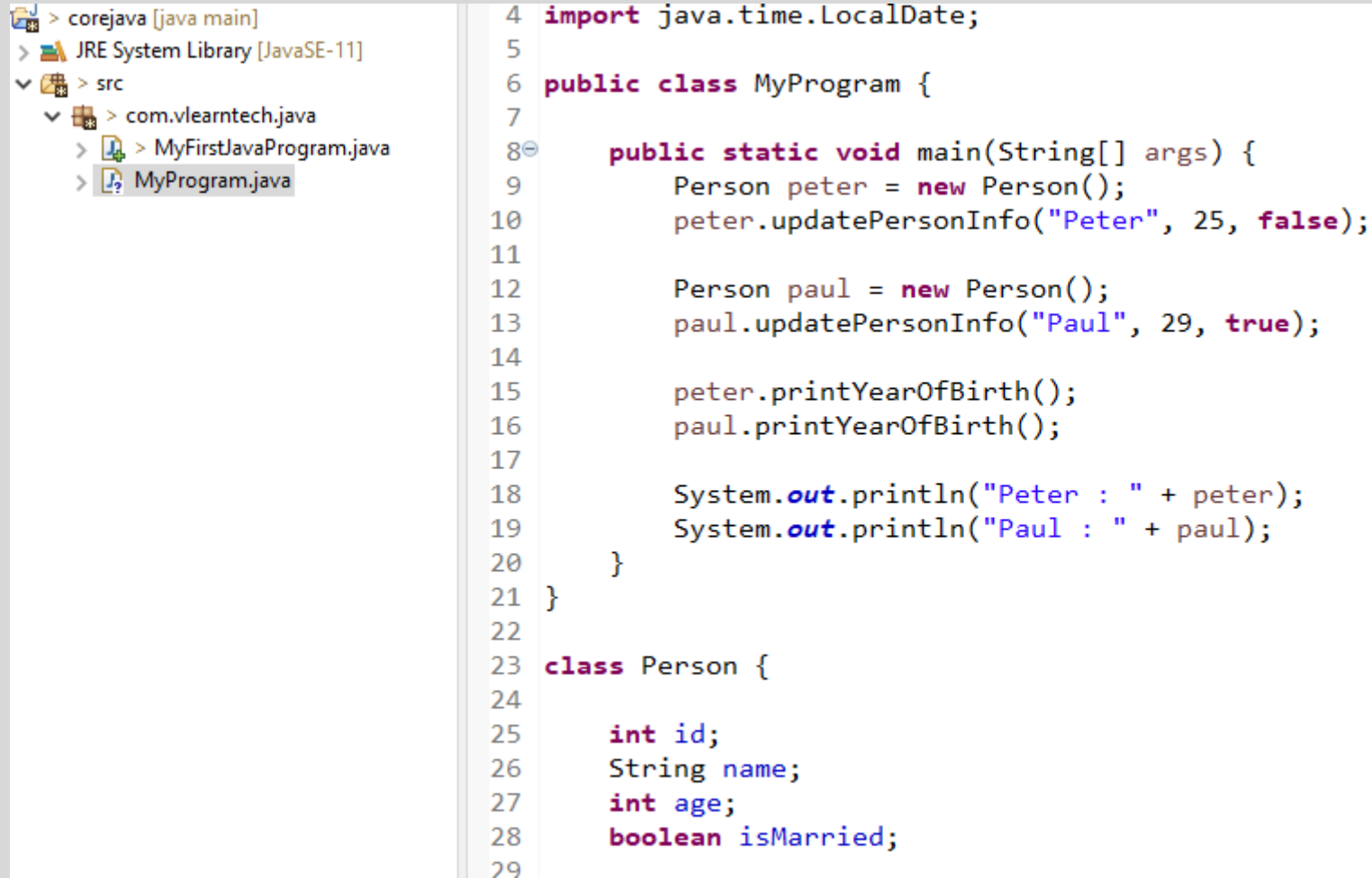
Training Plan - Basics

Topics to be covered

What is a Class. Public Classes & File Names – what is the correlation
Components of a class – Variables, Methods – Behaviors and Actions
Instances of class – how to create and assign class instances to variables.

How to execute methods of a class through a wrapper class/program
Variables and its categories based on usage and modifiers – Local, Instance, Static, Final
Constructors

What is a class.



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'src' directory with a package 'com.vlearntech.java' containing two files: 'MyFirstJavaProgram.java' and 'MyProgram.java'. The code editor displays the following Java code:

```
4 import java.time.LocalDate;
5
6 public class MyProgram {
7
8     public static void main(String[] args) {
9         Person peter = new Person();
10        peter.updatePersonInfo("Peter", 25, false);
11
12        Person paul = new Person();
13        paul.updatePersonInfo("Paul", 29, true);
14
15        peter.printYearOfBirth();
16        paul.printYearOfBirth();
17
18        System.out.println("Peter : " + peter);
19        System.out.println("Paul : " + paul);
20    }
21 }
22
23 class Person {
24
25     int id;
26     String name;
27     int age;
28     boolean isMarried;
29 }
```

Note that File name is **same as the main class**.

Public classes need to be defined in separate Java files with the name same as name of the class.

Class “**describes**” an object.

Examples -

- Animal
- Cycle
- Car
- Person
- Employee

Any common noun can be a class

Components of class

```
class Person {  
  
    String name;  
    int age;  
    boolean isMarried;  
  
    void printYearOfBirth() {  
        int birthYear = calculateYearOfBirth();  
        System.out.println("You were born in " + birthYear);  
    }  
  
    int calculateYearOfBirth() {  
        int yearOfBirth = 2021 - age;  
        return yearOfBirth;  
    }  
}
```

What is a class?

Fundamentals - Which are the variables here

Eclipse Tricks - How to refactor and rename variable names

How many methods in this class

How many variables

What will happen if I rename birthyear to yearOfBirth

Creating instances of Classes

```
public class MyProgram {  
  
    public static void main(String[] args) {  
        Person p = new Person();  
        p.name = "Peter";  
        p.age = 25;  
        p.isMarried = false;  
  
        p.printYearOfBirth();  
    }  
}
```

What is the meaning of the line

```
Person p = new Person();
```

What is the difference between the two occurrences of Person in above line.

Do it yourself - What should we change in the program so that output says

"Peter was born in 1996"

Multiple instances of a class

```
public class MyProgram {  
  
    public static void main(String[] args) {  
        Person peter = new Person();  
        peter.name = "Peter";  
        peter.age = 25;  
        peter.isMarried = false;  
  
        Person paul = new Person();  
        paul.name = "Paul";  
        paul.age = 29;  
        paul.isMarried = true;  
  
        peter.printYearOfBirth();  
        paul.printYearOfBirth();  
  
        System.out.println("Peter : " + peter);  
        System.out.println("Paul : " + paul);  
    }  
}
```

What is the meaning of this block

What is the meaning of the values that are printed in place of **peter** and **paul** variables

```
Peter was born in 1996  
Paul was born in 1992  
Peter : com.vlearntech.java.Person@5ccd43c2  
Paul : com.vlearntech.java.Person@4aa8f0b4
```

What happens if I say before the 1st print statement

```
paul = peter;
```

How to organize this code better

```
public class MyProgram {  
  
    public static void main(String[] args) {  
        Person peter = new Person();  
        peter.name = "Peter";  
        peter.age = 25;  
        peter.isMarried = false;  
  
        Person paul = new Person();  
        paul.name = "Paul";  
        paul.age = 29;  
        paul.isMarried = true;  
  
        peter.printYearOfBirth();  
        paul.printYearOfBirth();  
  
        System.out.println("Peter : " + peter);  
        System.out.println("Paul : " + paul);  
    }  
}
```

What are the **problems with the current code**

1. Every time I construct a new Person instance, I have to write 4 lines of code
2. This program will work only when run in 2021, since current year is hard coded in Person class.

```
class Person {  
  
    String name;  
    int age;  
    boolean isMarried;  
  
    void printYearOfBirth() {  
        int birthYear = calculateYearOfBirth();  
        System.out.println(name + " was born in " + birthYear);  
    }  
  
    private int calculateYearOfBirth() {  
        int yearOfBirth = 2021 - age;  
        return yearOfBirth;  
    }  
}
```

Refactored Person class

```
class Person {  
  
    String name;  
    int age;  
    boolean isMarried;  
  
    void updatePersonInfo (String personName, int personAge, boolean isPersonMarried) {  
        name = personName;  
        age = personAge;  
        isMarried = isPersonMarried;  
    }  
  
    void printYearOfBirth() {  
        int birthYear = calculateYearOfBirth();  
        System.out.println(name + " was born in " + birthYear);  
    }  
  
    private int calculateYearOfBirth() {  
        int yearOfBirth = LocalDate.now().getYear() - age;  
        return yearOfBirth;  
    }  
}
```

What is the new method doing

How would we use this method

This is called code "refactoring".

In a subsequent session you will learn about a better way of handling this (using constructors)

Updated wrapper class

```
public class MyProgram {  
  
    public static void main(String[] args) {  
        Person peter = new Person();  
        peter.updatePersonInfo("Peter", 25, false);  
  
        Person paul = new Person();  
        paul.updatePersonInfo("Paul", 29, true);  
  
        peter.printYearOfBirth();  
        paul.printYearOfBirth();  
  
        System.out.println("Peter : " + peter);  
        System.out.println("Paul : " + paul);  
    }  
}
```

Looks neater!

This can be further cleaned up as below, but we will understand it better when we learn Constructors

```
public class MyProgram {  
  
    public static void main(String[] args) {  
        Person peter = new Person("Peter", 25, false);  
        Person paul = new Person("Paul", 29, true);  
  
        peter.printYearOfBirth();  
        paul.printYearOfBirth();  
  
        System.out.println("Peter : " + peter);  
        System.out.println("Paul : " + paul);  
    }  
}
```

Special Requirement - personId

We want to add one more field called **id** in the **Person** class which will be an “auto incremented” number.

How will we address this requirement

Updated Person class using the static variable

```
class Person {  
  
    int id;  
    String name;  
    int age;  
    boolean isMarried;  
  
    static int counter = 0;  
  
    void updatePersonInfo (String personName, int personAge, boolean isPersonMarried) {  
        id = ++counter;  
        name = personName;  
        age = personAge;  
        isMarried = isPersonMarried;  
    }  
  
    void printYearOfBirth() {  
        int birthYear = calculateYearOfBirth();  
        System.out.println(name + " was born in " + birthYear);  
    }  
  
    private int calculateYearOfBirth() {  
        int yearOfBirth = LocalDate.now().getYear() - age;  
        return yearOfBirth;  
    }  
}
```

What will happen if we take out the **static** keyword

So what are the **types of variables** you see

Which are the local variables here

Which are the instance variables

And which is the static variable

Special Requirement - maxCapacity

Suddenly your product manager comes and says – *“Thanks for creating the Person class. It is good. But while you have assigned id to each person, I only want to add instances till a point. This limit should be configurable, and system should not allow this limit to be changes programmatically”*

In summary - **Once initialized, its value can never change**

Now let us see how will we address this requirement.

Updated Person class – using the final keyword

```
class Person {  
  
    int id;  
    String name;  
    int age;  
    boolean isMarried;  
  
    final int maxCapacity = 99;  
    static int counter = 0;  
  
    void updatePersonInfo(String personName, int personAge, boolean isPersonMarried) {  
        id = ++counter;  
        if (id > maxCapacity) {  
            // Do not allow creation of the object  
        }  
        name = personName;  
        age = personAge;  
        isMarried = isPersonMarried;  
    }  
  
    void printYearOfBirth() {  
        int birthYear = calculateYearOfBirth();  
    }  
}
```

So what is the **final** keyword. What does that indicate

Notice how we are using the variable in the IF condition

What does this IF condition mean

How have we addressed configurability

Best Practice – Comments

```
class Person {  
  
    int id;  
    String name;  
    int age;  
    boolean isMarried;  
  
    final int maxCapacity = 99;  
    static int counter = 0;  
  
    void updatePersonInfo(String personName, int personAge, boolean isPersonMarried) {  
        id = ++counter;  
        if (id > maxCapacity) {  
            // Do not allow creation of the object  
        }  
        name = personName;  
        age = personAge;  
        isMarried = isPersonMarried;  
    }  
  
    /*  
    * Finds the year of birth and prints the same to the output console  
    * Does not accept any parameter and does not return anything.  
    */  
    void printYearOfBirth() {  
        int birthYear = calculateYearOfBirth();  
        System.out.println(name + " was born in " + birthYear);  
    }  
  
    private int calculateYearOfBirth() {  
        int yearOfBirth = LocalDate.now().getYear() - age;  
        return yearOfBirth;  
    }  
}
```

Comments in code help make the code readable, understandable.

Types of comments –

- Inline comments //
- Block comments /*....*/
- Javadoc

Eclipse Trick:

Press **Ctrl + /** on any line or multiple lines for Inline Comments

Press **Ctrl + Shift + /** on any selection for block commenting

Best Practice – Keep Classes separate

```
class Person {  
  
    int id;  
    String name;  
    int age;  
    boolean isMarried;  
  
    final int maxCapacity = 99;  
    static int counter = 0;  
  
    void updatePersonInfo(String personName, int personAge, boolean isPersonMarried) {  
        id = ++counter;  
        if (id > maxCapacity) {  
            // Do not allow creation of the object  
        }  
        name = personName;  
        age = personAge;  
        isMarried = isPersonMarried;  
    }  
  
    /*  
    * Finds the year of birth and prints the same to the output console  
    * Does not accept any parameter and does not return anything.  
    */  
    void printYearOfBirth() {  
        int birthYear = calculateYearOfBirth();  
        System.out.println(name + " was born in " + birthYear);  
    }  
  
    private int calculateYearOfBirth() {  
        int yearOfBirth = LocalDate.now().getYear() - age;  
        return yearOfBirth;  
    }  
}
```

Person as a class could be kept in another file called Person.java

Eclipse Trick: Cut the entire Person class from MyProgram.java and just paste directly on the package name in the package explorer.

Notice how there is no compilation issue and such a big change was handled seamlessly by ECLIPSE.

That's all for today. Please practice below

Read the below paragraph, and create a class called Animal to hold the information. Identify the variables and methods in the class. **Additionally, write a wrapper main method to create instances of multiple animals and invoke their methods.** Keep code well refactored and well commented and follow proper naming conventions of class, variables and methods. Use

*We are trying to create a system for a Zoo. In the Zoo, there are many animals. Children visit the zoo and watch the bright colors of animals, hear the sounds that the animals make and also see them move around. The zoo officials have designated each animal an unique identity number so that their health information can be tracked. When new animal is admitted into the zoo, this id number should be autogenerated and assigned.. **They also try to keep track of number of steps an animal has taken in a year through a micro device attached to each animal's leg/wing.***