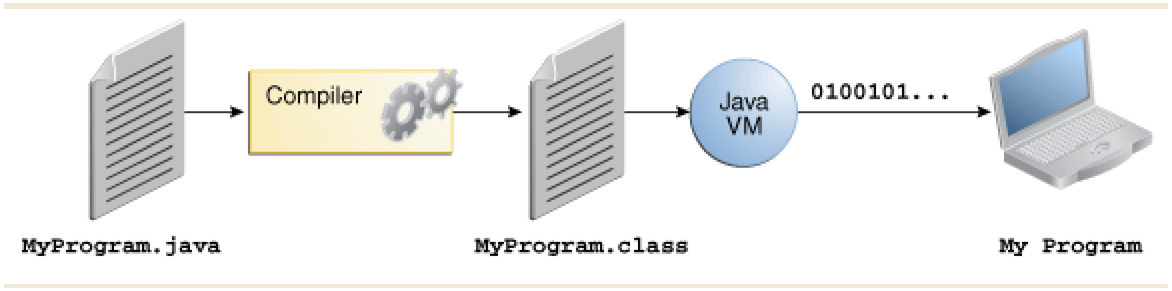
Labcycle 1: Introduction Lab

A java file is also known as source code file. A class file is also known as 'byte code'.

The **javac command** is used to compile Java programs, it takes .java file as input and produces bytecode.

The **java command** is used to execute the bytecode of java. It takes byte code as input and runs it and produces the output.



Step1: $java –version

Java Quickstart:
In Java, every application begins with a class name, and that class must match the filename.
Let's create our first Java file, called MyClass.java, in vi editor.
The file should contain a "Hello World" message, which is written with the following code:

Step 2: vi MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Step 2.1: Compile: $javac MyClass.java
Step 2.2: Run: $java MyClass
Output: Hello World

Example explained
Every line of code that runs in Java must be inside a class.
Every program must contain the main() method.

<u>public static void main(String[] args)</u>
Java JVM will always look for specific method signature to start running an application, and that would be **public static void main(String args[]).** Here args is an argument of type String array. Argument name could be anything, not necessarily args.
public: It is an Access Modifier, which defines who can access this Method. Public means that this Method will be accessible by any Class
static: Static is a keyword which identifies the class related thing. It means the given Method or variable is not instance related but Class related. It can be accessed without creating the instance of a Class.
void: Is used to define the Return Type of the Method. It defines what the method can return. Void means the Method will not return any value.
main(): Main is the name of the Method. This Method name is searched by JVM as a starting point for an application with a particular signature only.
String args[] / String… args: It is the parameter to the main Method. Argument name could be anything.

Inside the main() method : System.out.println() - we can use the println() method to print a line of text to the screen:
System – is a final class in java.lang package. Among the facilities provided by the System class are standard input, standard output, and error output streams
out – is a static member field of System class and is of type PrintStream. Its access specifiers are public final. This gets instantiated during startup and gets mapped with standard output console of the host. This stream is open by itself immediately after its instantiation and ready to accept data.
println – is a method of PrintStream class. println prints the argument passed to the standard console and a newline.

Step 3: Java User Input (Scanner): The Scanner class is used to get user input, and it is found in the java.util package.
To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

```
import java.util.Scanner;  // Import the Scanner class

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

Input Types: In the example above, we used the nextLine() method, which is used to read Strings. To read other types, look at the table below:

Method	**Description**
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Example:

```
import java.util.Scanner;

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);

    System.out.println("Enter name, age and salary");
    // String input
    String name = myObj.nextLine();
    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();

    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```

Step 4: Create a Class

To create a class, use the keyword class:

```
MyClass.java
Create a class called "MyClass" with a variable x:
public class MyClass {
  int x = 5;
}
```

Note:

A class should always start with an uppercase first letter, and that the name of the java file should match the class name.
A class can have only public and default access.
The public class needs to be in the same name java file.
A single java file can contain more than one non-public class but can have only one public class.
Classes are written in a Java source file.
A source file can contain more than one Java Class.
There are some rules associated to the Java source file as listed below.
Rules Applicable to Source file

- There can be only one public class per source code file but it can have multiple non-public classes.
- In case there is any public class present in the source code file, the name of the file should be the name of the class.
- The sequence of statements in a source code file should be package >> import >> Class declaration.
- No Sequence rule is applied for Comments.
- Comments can be there in any part of the source code file at any location.
- Files with no public class can have any name for the class.
- Import and package statements should be applied to all the classes in the same source code file.

How to Create an Object of a Class: To create an Object of a Class <new> Keyword can be used.
Syntax: <Class_Name> ClassObjectReference = new <Class_Name>();

Here the constructor of the Class(Class_Name) will be executed and an Object will be created(ClassObjectReference will hold the reference of the created Object in memory).

Step 5: Create an Object

In Java, an object is created from a class.
We have already created the class named MyClass, so now we can use this to create objects.
3 steps:

- Declaration : a variable declaration with a variable name and object type
- Instantiation : new keyword is used to create object
- Initialization : new keyword followed by call to constructor – this call initializes the object

To create an object of MyClass, specify the class name, followed by the object name, and use the keyword new:
Example**:** Create an object called "myObj" and print the value of x:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Step 6: Multiple Objects
You can create multiple objects of one class:
Example: Create two objects of MyClass:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

Step 7: Using Multiple Classes

You can also create an object of a class and access it in another class.
This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).
Remember that the name of the java file should match the class name.
In this example, we have created two files in the same directory/folder:

- MyClass.java
- OtherClass.java

MyClass.java	OtherClass.java

public class MyClass
{
  int x = 5;
}
	class OtherClass
	{
	  public static void main(String[] args) {
	        MyClass myObj = new MyClass();
	        System.out.println(myObj.x);
	            }
	}

When both files have been compiled:

```
$ javac MyClass.java
$ javac OtherClass.java
```

Run the OtherClass.java file:

```
$ java OtherClass
```

Output: 5

Step 8: Java Class Attributes
We used the term "variable" for x in the example (as shown below).
It is actually an attribute of the class.
Or you could say that class attributes are variables within a class:

Example: Create a class called "MyClass" with two attributes: x and y:

```
public class MyClass {
  int x = 5;   int y = 3;
}
```

You can access attributes by creating an object of the class, and by using the dot syntax (.):

The following example will create an object of the MyClass class, with the name myObj. We use the x attribute on the object to print its value:

Example
Create an object called "myObj" and print the value of x:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Step 9: Modify Attributes

Example: Set the value of x to 40:

```
public class MyClass {
  int x;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

Step 10: Override existing values:
Example: Change the value of x to 25:

```
public class MyClass {
  int x = 10;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

Step 11: If you don't want the ability to override existing values, declare the attribute as final:

```
public class MyClass {
  final int x = 10;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 25; // will generate an error: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

Step 12: Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

Example: Change the value of x to 25 in myObj2, and leave x in myObj1 unchanged:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Step 13: Multiple Attributes

Example: You can specify as many attributes as you want:

```
public class Person {
  String fname = "John";
  String lname = "Doe";
  int age = 24;

  public static void main(String[] args) {
    Person myObj = new Person();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

Step 14: Java Methods

A method is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as functions.

Step 14.1 : Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parantheses (). Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Example: Create a method inside MyClass:

```
public class MyClass {
  static void myMethod() {
    // code to be executed
  }
}
```

Example Explained

- myMethod() is the name of the method
- static means that the method belongs to the MyClass class and not an object of the MyClass class.
- void means that this method does not have a return value.

Step 14.2 :Call a Method

To call a method in Java, write the method's name followed by two parantheses () and a semicolon;
In the following example, myMethod() is used to print a text (the action), when it is called:
Example: Inside main, call the myMethod() method:

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

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

// Outputs "I just got executed!"
A method can also be called multiple times, if you want:
Example

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

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

```
// I just got executed!
// I just got executed!
// I just got executed!
```

Step 14.3: Method Parameters

Information can be passed to functions as parameter.
Parameters are specified after the method name, inside the parentheses.
You can add as many parameters as you want, just separate them with a comma.
The following example has a method that takes a String called fname as parameter.
When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
public class MyClass {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
```

// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes

Step 14.4: Return Values

The void keyword, used in the examples above, indicates that the method should not return a value.
If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

Example

```
public class MyClass {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
```

// Outputs 8 (5 + 3)
This example returns the sum of a method's two parameters:
Example

```
public class MyClass {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(5, 3));
  }
}
```

// Outputs 8 (5 + 3)
You can also store the result in a variable (recommended):
Example

```
public class MyClass {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = myMethod(5, 3);
    System.out.println(z);
  }
}
```

// Outputs 8 (5 + 3)

Step 15: Java Class Methods

Methods are declared within a class, and that they are used to perform certain actions:

Example

```
Create a method named myMethod() in MyClass:
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

myMethod() prints a text (the action), when it is called.
To call a method, write the method's name followed by two parantheses () and a semicolon;

Example
Inside main, call the myMethod() method:

```
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }

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

// Outputs "Hello World!"

Step 15. 1: Static or Public
You will often see Java programs that have either static or public attributes and methods.
In the example above, we created a static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:

Example: An example to demonstrate the differences between static and public methods:

```
public class MyClass {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    MyClass myObj = new MyClass(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

Step 15.2 : Access Methods With an Object

Example: Create a Car object named myCar. Call the fullThrottle() and speed() methods on the myCar object, and run the program:

```
// Create a Car class
public class Car {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
    Car myCar = new Car();     // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() method
    myCar.speed(200);          // Call the speed() method
  }
}
```

```
// The car is going as fast as it can!
// Max speed is: 200
```

Example explained

1) We created a custom Car class with the class keyword.
2) We created the fullThrottle() and speed() methods in the Car class.
3) The fullThrottle() method and the speed() method will print out some text, when they are called.
4) The speed() method accepts an int parameter called maxSpeed - we will use this in 8).
5) In order to use the Car class and its methods, we need to create an object of the Car Class.
6) Then, go to the main() method, which you know by now is a built-in Java method that runs your program (any code inside main is executed).
7) By using the new keyword we created a Car object with the name myCar.
8) Then, we call the fullThrottle() and speed() methods on the myCar object, and run the program using the name of the object (myCar), followed by a dot (.), followed by the name of the method (fullThrottle(); and speed(200);). Notice that we add an int parameter of 200 inside the speed() method.

Remember that..
The dot (.) is used to access the object's attributes and methods.
To call a method in Java, write the method name followed by a set of parantheses (), followed by a semicolon (;).
A class must have a matching filename (Car and Car.java).

Step 15. 3: Using Multiple Classes
It is a good practice to create an object of a class and access it in another class.
Remember that the name of the java file should match the class name.
In this example, we have created two files in the same directory:

- Car.java
- OtherClass.java

Car.java

```
public class Car {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

OtherClass.java

```
class OtherClass {
  public static void main(String[] args) {
    Car myCar = new Car();     // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() method
    myCar.speed(200);          // Call the speed() method
  }
}
```

When both files have been compiled:

```
$javac Car.java
$javac OtherClass.java
```

Run the OtherClass.java file:

```
$java OtherClass
```

And the output will be:

```
The car is going as fast as it can!
Max speed is: 200
```

Step 16: Java Constructors
A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:
Example
Create a constructor:

```
// Create a MyClass class
public class MyClass {
  int x;  // Create a class attribute

  // Create a class constructor for the MyClass class
  public MyClass() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    MyClass myObj = new MyClass(); // Create an object of class MyClass (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}
```

// Outputs 5

Note that the constructor name must match the class name, and it cannot have a return type (like void).
Also note that the constructor is called when the object is created.
All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

Step 16. 1: Constructor Parameters

Constructors can also take paramaters, which is used to initialize attributes.
The following example adds an int y parameter to the constructor.
Inside the constructor we set x to y (x=y). When we call the constructor, we pass a parameter to the constructor (5), which will set the value of x to 5:

Example

```
public class MyClass {
  int x;

  public MyClass(int y) {
    x = y;
  }

  public static void main(String[] args) {
    MyClass myObj = new MyClass(5);
    System.out.println(myObj.x);
  }
}
```

// Outputs 5
You can have as many parameters as you want:

Example

```
public class Car {
  int modelYear;
  String modelName;

  public Car(int year, String name) {
    modelYear = year;
    modelName = name;
  }

  public static void main(String[] args) {
    Car myCar = new Car(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}
```

// Outputs 1969 Mustang