

Introduction to Java and Netbeans

Lab No 2

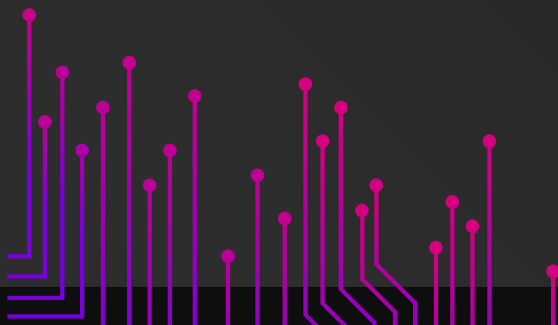
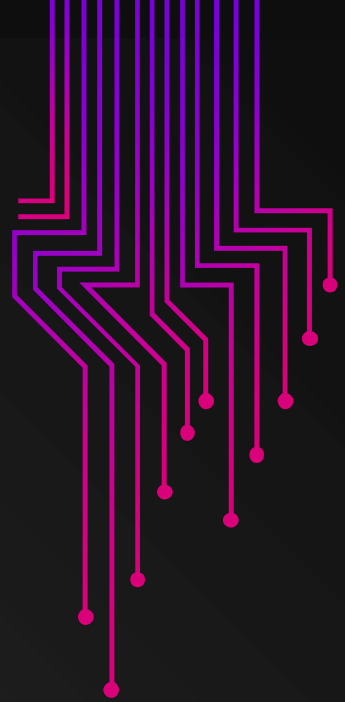


Table of contents

01

Classes and Objects

02

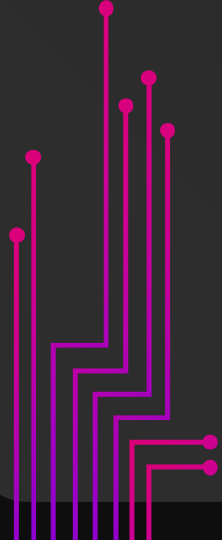
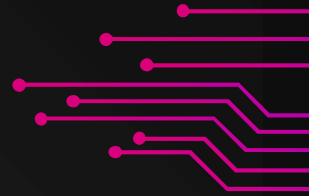
Input and Output in Java

03

Constructors

04

Access Modifiers



A decorative graphic of stylized circuit lines in purple and pink, located in the top right corner of the slide. The lines are vertical and horizontal, with small circles at the ends, resembling a circuit board.

01

Classes and Objects

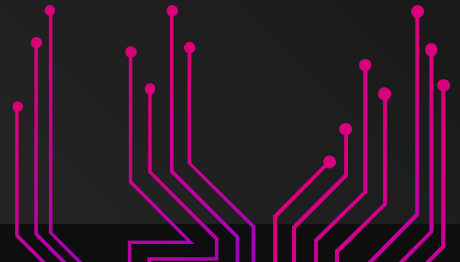
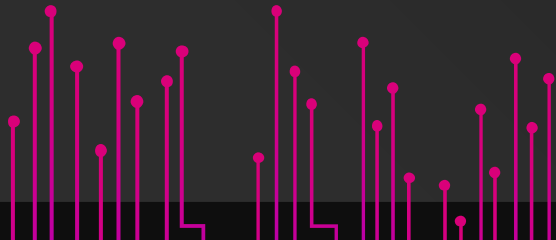
OOP Concepts

A decorative graphic of stylized circuit lines in purple and pink, located in the bottom left corner of the slide. The lines are vertical and horizontal, with small circles at the ends, resembling a circuit board.

Introduction of Classes


In Java, **classes and objects** are basic concepts of Object Oriented Programming (OOPs) that are used to represent real-world concepts and entities. The class represents a group of objects having similar properties and behavior.

For example, the animal type **Dog** is a class while a particular dog named **Tommy** is an object of the **Dog** class.

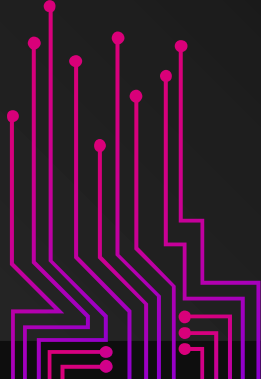



Understanding the difference between classes and objects

Class	Object
Class is the blueprint of an object. It is used to create objects.	An object is an instance of the class.
No memory is allocated when a class is declared.	Memory is allocated as soon as an object is created.
A class is a group of similar objects.	An object is a real-world entity such as a book, car, etc.
Class is a logical entity.	An object is a physical entity.
A class can only be declared once.	Objects can be created many times as per requirement.
An example of class can be a car .	Objects of the class car can be BMW, Mercedes, Ferrari, etc.



A **class in Java** is a set of objects which shares common characteristics and common properties. It is a user-defined blueprint or prototype from which objects are created. For example, IceCream is a class while a particular icecream named cornetto is an object.





Classes

Class is a group of variables of different data types and a group of methods.

A Class in Java can contain:

Data member

Method

Constructor

Nested Class

Interface



Open Netbeans



Create a new Project

When the NetBeans main window is open, click on creating a new project. You can initialize the name of the project with capital letter as well.



Opening the Project

Once the project is opened, on the left side of the hierarchy, open the Source packages of the Project



Open the default class

The default class of the project is with the name of the project along with .java extension.



NetBeans

Create New Class

You can select the project and create the new class. Keep one thing in mind that the new class name should always start with the small letter.

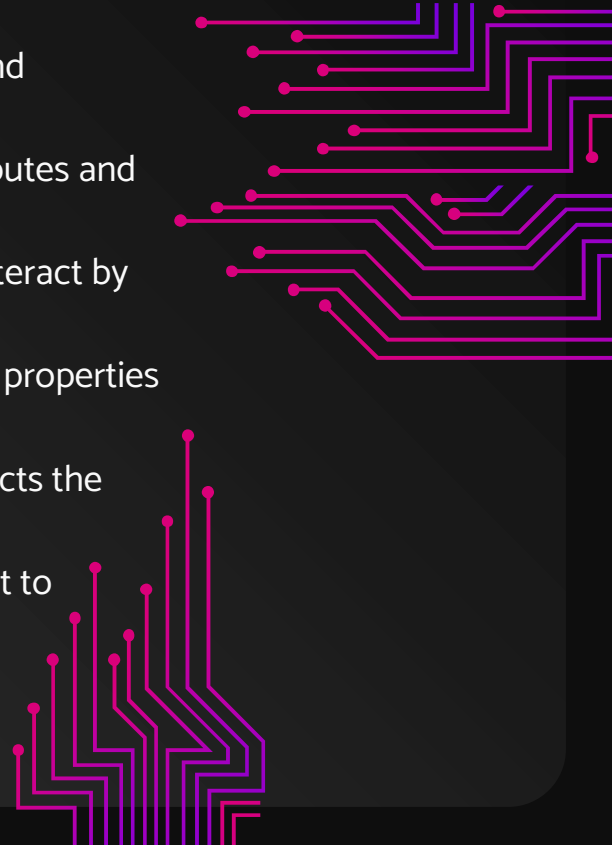
Main Function in the Main Default Class

The main function is just like the main function of the C++



Objects

- An **object in Java** is a basic unit of Object-Oriented Programming and represents real-life entities.
- Objects are the instances of a class that are created to use the attributes and methods of a class.
- A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:
 1. **State:** It is represented by attributes of an object. It also reflects the properties of an object.
 2. **Behavior:** It is represented by the methods of an object. It also reflects the response of an object with other objects.
 3. **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

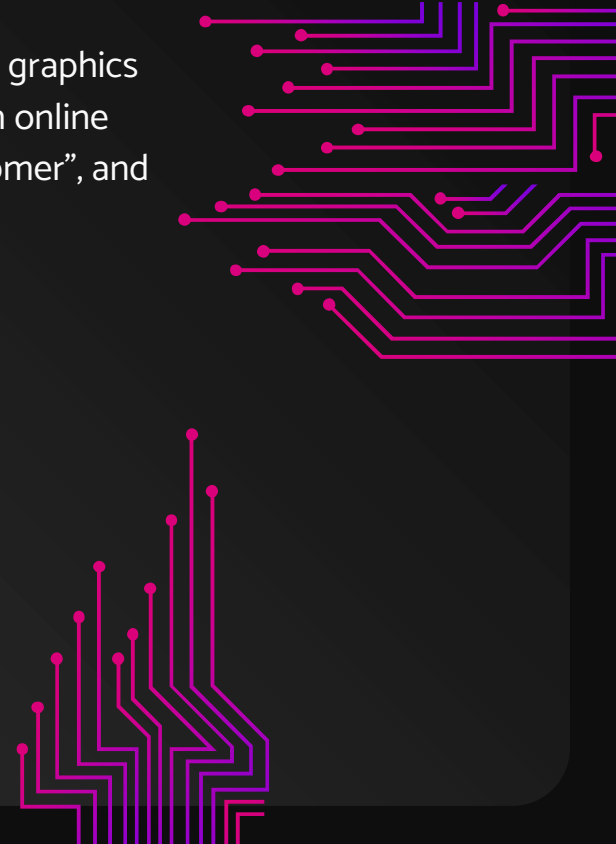


Example



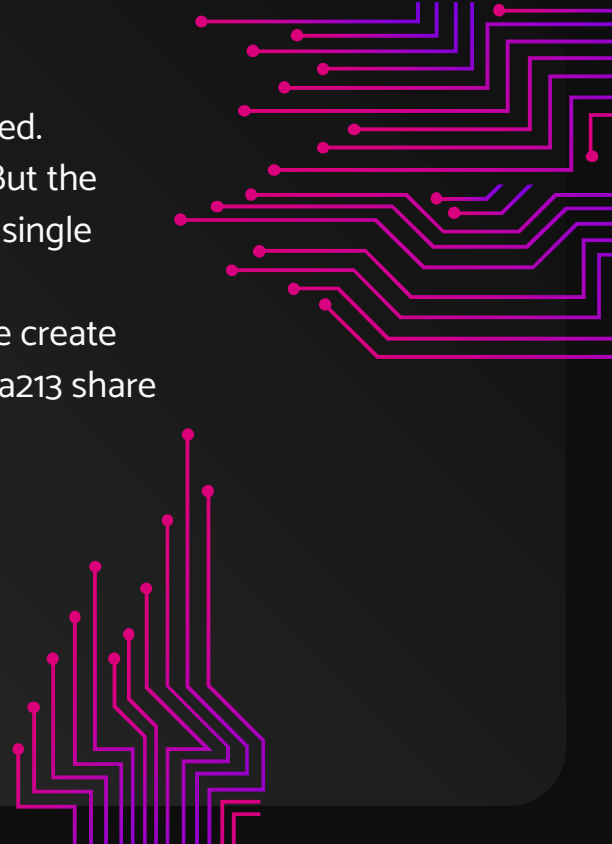
Objects

- Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, and “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.



Declaring Objects (Also called instantiating a Class)

- When an object of a class is created, the class is said to be instantiated.
- All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.
- For example in our lab1 class example, the object is b213. Suppose we create another object of lab1 class and name it as a213. Now both b213 and a213 share the same attributes and behaviors but each has the unique state.



The image features a dark gray background with a subtle gradient. In the top right and bottom left corners, there are decorative elements resembling circuit board traces. These lines are colored in a gradient from purple to pink and end in small circular dots. The main content is centered on the left side of the image.

02

Input and Output in Java

Java Programming

Variables and printing their values

1. Variable declaration and initialization is similar to the one in C++. Remember, a class will not be able to run if it does not contain the main function.

```
public class lab1 {  
    public void print()  
    {  
        int local = 0; // local variable  
    }  
}
```



Generating a simple output in NetBeans

- In order to print the content in NetBeans, instead of using the cout function like in C++, we use `System.out.println("Content which you want to print");`

```
public class lab1 {  
    public static void main(String[] args){  
        int local = 0; // local variable  
        System.out.println(x: "My first Program of Printing");  
        System.out.println(x: local);  
    }  
}
```


Another example

```
public class lab1 {  
    // declaration of the attributes of the classes associated with the lab  
    public int name;  
    private String password;  
    public int totalClasses;  
    public String[] subjectNames;  
    public String[] subjectSoftwares;  
}
```

Continued...

```
public static void main(String[] args){  
  
    //Wrong way  
    System.out.println(x: name);  
    System.out.println(x: password);  
}
```



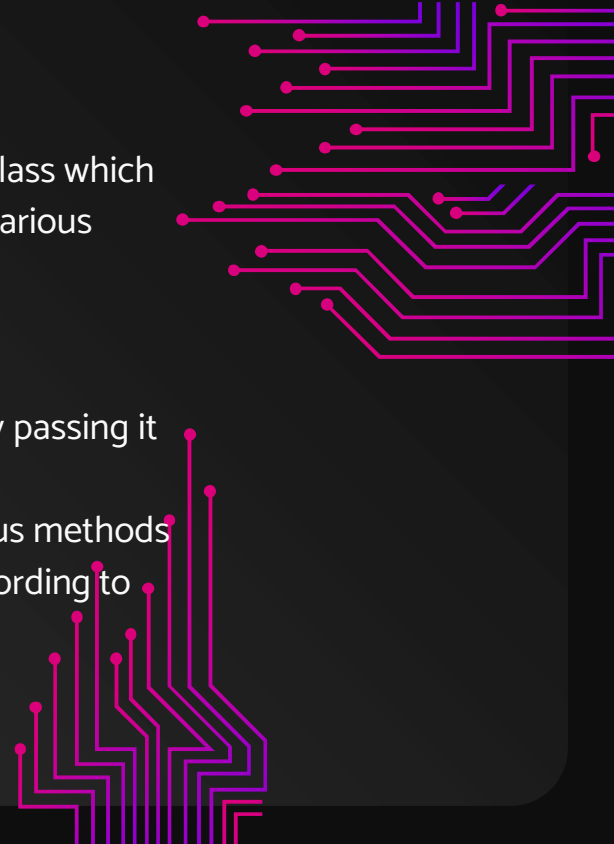
Continued...

```
public static void main(String[] args){  
  
    //Create the object of the class first  
    lab1 b213 = new lab1();  
    //Accessing the variables of the class through the object  
    System.out.println(x: b213.name);  
    System.out.println(x: b213.password);  
}
```



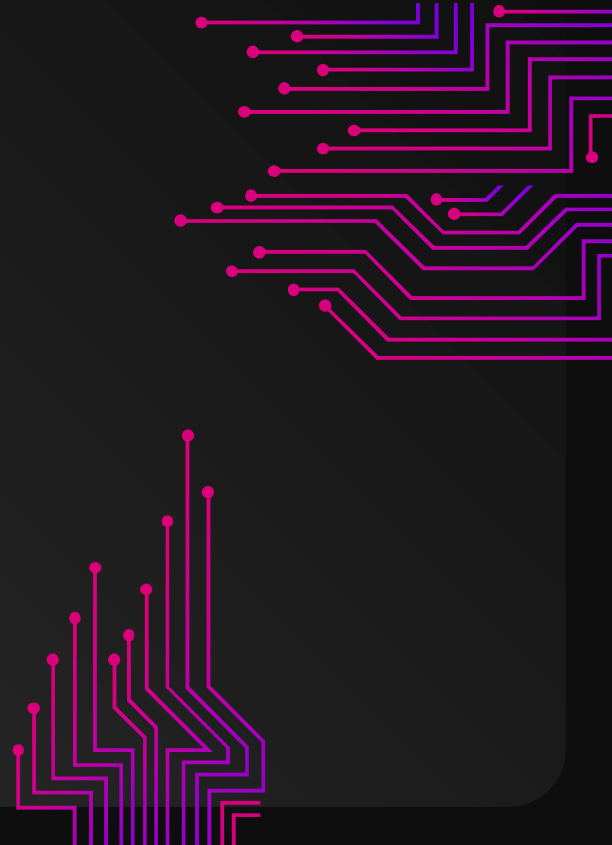
Taking Input in Java

- The most common way to take user input in Java is using **Scanner** class which is part of **java.util package**. The scanner class can read input from various sources like console, files or streams.
- **Follow these steps to take user input using Scanner class:**
 1. Import the Scanner class using ***import java.util.Scanner;***
 2. **Create the Scanner object** and connect Scanner with **System.in** by passing it as an argument i.e. ***Scanner scn = new Scanner(System.in);***
 3. Print a message to prompt for user input and you can use the various methods of Scanner class like `nextInt()`, `nextLine()`, `next()`, `nextDouble` etc according to your need.



Taking Input in Java

```
package com.mycompany.opplab1;  
import java.util.Scanner;
```

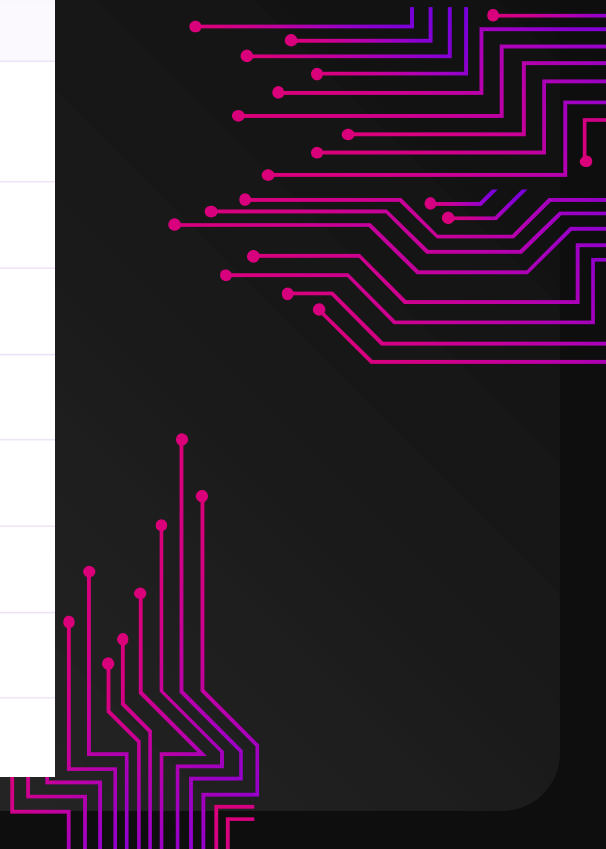


Taking Input in Java

```
public static void main(String[] args){  
    lab1 b213 = new lab1();  
    Scanner scanInt = new Scanner(source: System.in);  
    Scanner scanString = new Scanner(source: System.in);  
  
    System.out.println (x: "Enter the name of the lab");  
    b213.name = scanString.nextLine();  
    System.out.println("The name of the lab is " + b213.name);  
  
    System.out.println(("Enter the total number of the classes conducted in the lab"));  
    b213.totalClasses = scanInt.nextInt();  
    System.out.println("The total number of the classes are "+b213.totalClasses);  
  
    System.out.println(x: "Enter the password you want to set for the lab");  
    b213.password = scanString.nextLine();  
    System.out.println("The password you set is " + b213.password);  
}
```

Scanner functions

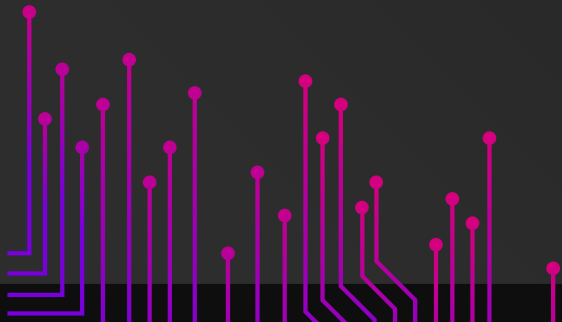
Method	Description
<u>nextBoolean()</u>	Used for reading Boolean value.
<u>nextByte()</u>	Used for reading Byte value.
<u>nextDouble()</u>	Used for reading Double value.
<u>nextFloat()</u>	Used for reading Float value.
<u>nextInt()</u>	Used for reading Int value.
<u>nextLine()</u>	Used for reading Line value.
<u>nextLong()</u>	Used for reading Long value.
<u>nextShort()</u>	Used for reading Short value.



03

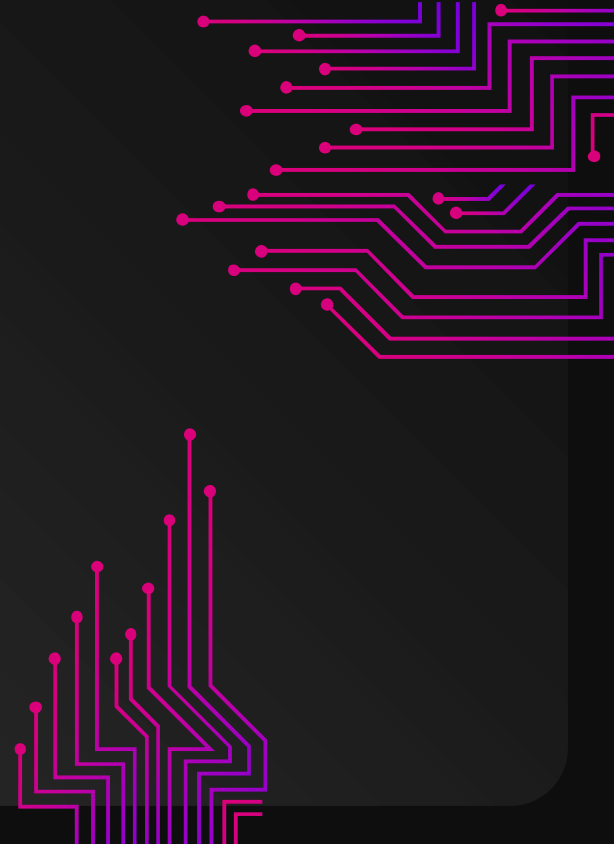
Constructors

OPP Concepts



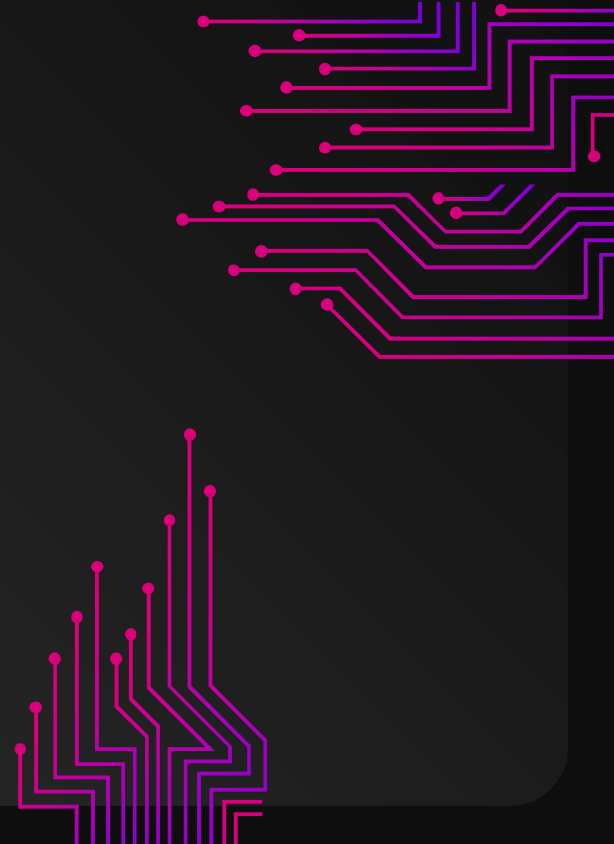
Constructors

- Java constructors or constructors in Java is a terminology used to construct something in our programs.
- 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.



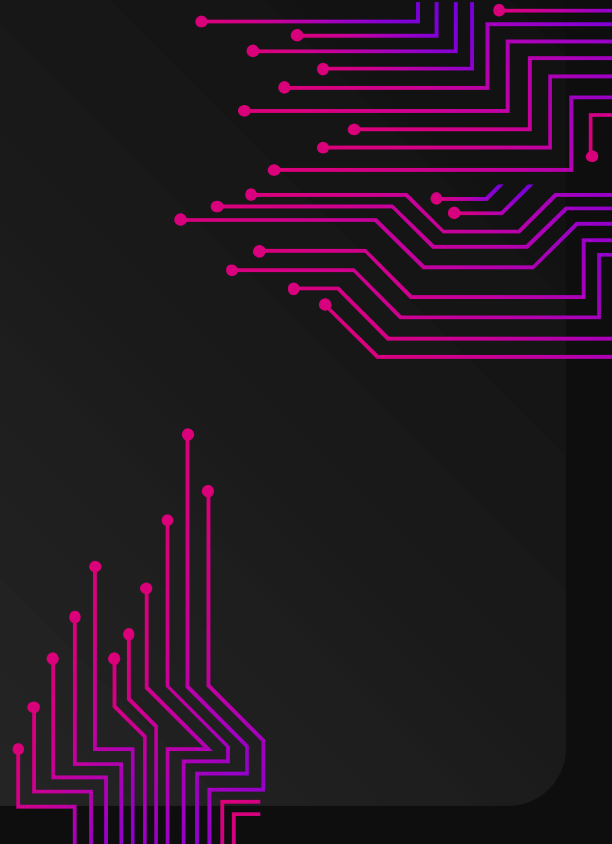
Constructors

- In Java, a Constructor is a block of codes similar to the method. It is called when an instance of the class is created.
- At the time of calling the constructor, memory for the object is allocated in the memory.
- It is a special type of method that is used to initialize the object.
- Every time an object is created using the `new()` keyword, at least one constructor is called.



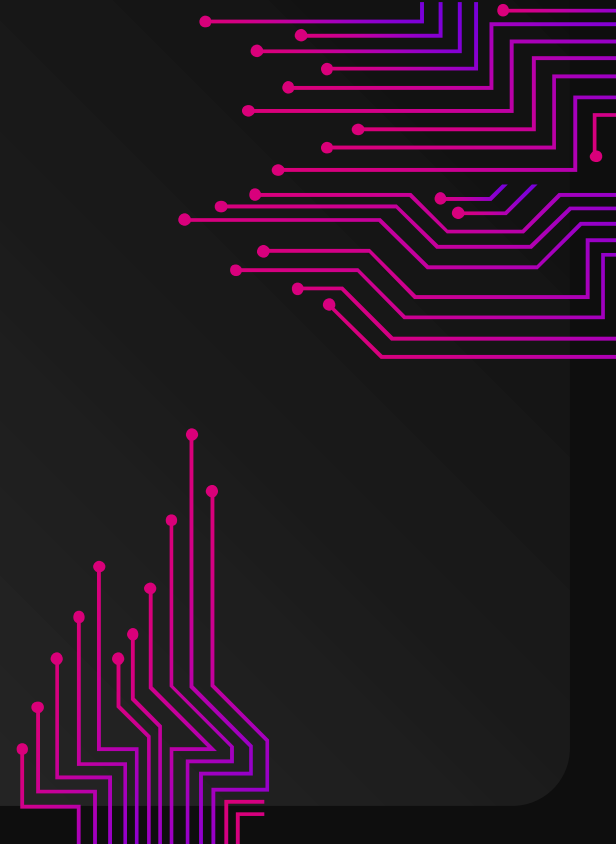
Constructors

- Constructors must have the same name as the class within which it is defined it is not necessary for the method in Java.
- Constructors do not return any type while method(s) have the return type or **void** if does not return any value.
- Constructors are called only once at the time of Object creation while method(s) can be called any number of times.



Types of Constructors in Java

- Default Constructor
- Parameterized Constructor
- Copy Constructor



Default Constructors

```
lab1()  
{  
    System.out.println(x: "This is the default Constructor");  
}
```

```
--- exec:3.1.0:exec (default-cli) @ oppLab1 ---  
This is the default Constructor
```

Parameterized Constructors

```
//Parameterized Constructor
lab1(String name1, String Password1, int tclasses){
    this.name = name1;
    this.password = Password1;
    this.totalClasses = tclasses;
}

public static void main(String[] args){
    lab1 b213 = new lab1(name1: "B209", Password1: "NewLab", tclasses: 10);
}
```



Copy Constructors

```
lab1(lab1 obj)
{
    this.name = obj.name;
    this.password = obj.password;
    this.totalClasses = obj.totalClasses;
}

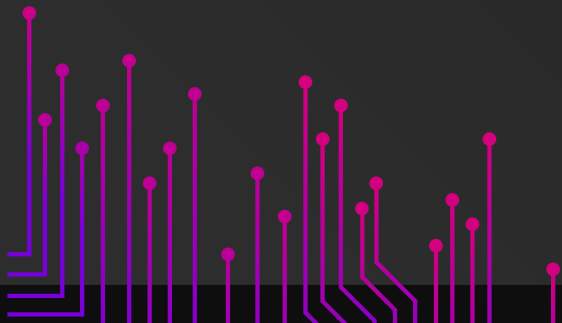
// declare the methods that will be using these attributes mentioned above
// calling Constructor

public static void main(String[] args){
    lab1 b213 = new lab1(name1: "B209", Password1: "NewLab", tclasses: 10);
    lab1 b209 = new lab1(obj:b213);
```

04

Access Modifiers

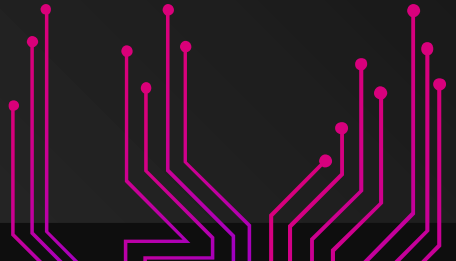
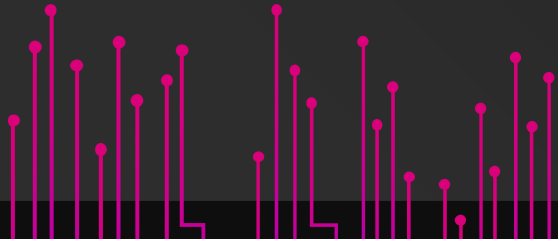
OOP Concepts



Access Modifiers

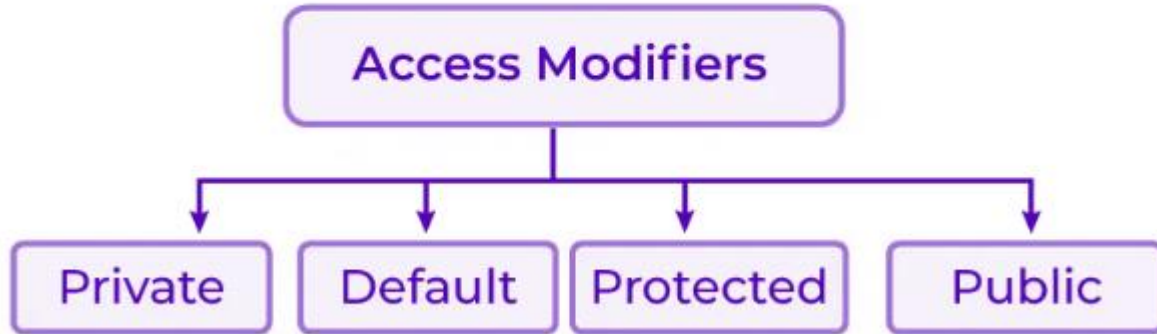
In Java, **Access modifiers** helps to restrict the **scope** of a **class, constructor, variable, method, or data member**.

It provides **security, accessibility**, etc. to the user depending upon the access modifier used with the element.



Types of Access Modifiers

Access Modifiers in Java



Default Access Modifiers

When no access modifier is specified for a class, method, or data member, it is said to be having the **default** access modifier by default. The default access modifiers are accessible *only within the same package*.

```
class inputInt {  
    int newone;
```

Now call this newone variable in the other class that is lab1. There will be an error provided that the inputInt class is present in the other package. If not there will be no error.

```
public static void main(String[] args){  
    inputInt integer = new inputInt();  
    integer.newone = 10;
```

Private Access Modifiers

The **private access modifier** is specified using the keyword **private**. The methods or data members declared as private are accessible *only within the class in which they are declared*.

```
class inputInt {  
    private int newone;
```

The access of the private data member is not allowed.

```
public static void main(String[] args){  
    inputInt integer = new inputInt();  
    integer.newone = 10;
```



Protected Access Modifiers

The **protected access modifier** is specified using the keyword **protected**. The methods or data members declared as protected are *accessible within the same package or subclasses in different packages*.

```
class inputInt {  
    protected int newone;
```

```
public static void main(String[] args){  
    inputInt integer = new inputInt();  
    integer.newone = 10;
```



Public Access Modifiers

The **public access modifier** is specified using the keyword **public**.

The public access modifier has the **widest scope** among all other access modifiers.

Classes, methods, or data members that are declared as public are ***accessible from everywhere*** in the program. There is no restriction on the scope of public data members.

```
class inputInt {  
    public int newone;
```



Difference of the access Modifiers

	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package Non-Subclass	No	No	No	Yes

Lab Task

Scenario: University Student Management System

Scenario Description:

A university wants to manage student records. Each student has a name, roll number, and department. Each department has a department name and department head. The students need to implement two classes:

Department Class: Stores department details.

Student Class: Stores student details.

The implementation should

1. Use access modifiers (private, public, and default).
2. Take user input for student and department details.
3. Create a display function in department class which will display the contents of the Student Class as well.
4. Implement default, parameterized, and copy constructors.
5. Use objects to establish a relationship between Student and Department.

