**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Fall 2024, Lab Manual - 01**

|  |  |
| --- | --- |
| **Course Code: SL3001** | **Course : Software construction and Development** |
| **Instructor(s) :** | **Yasir Arfat** |

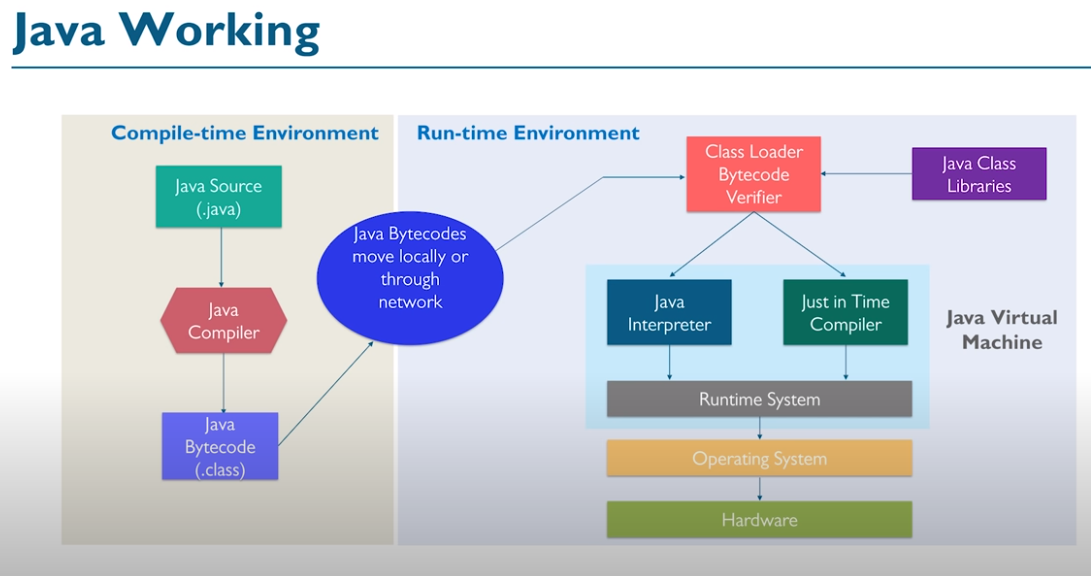
**Lab # 01**

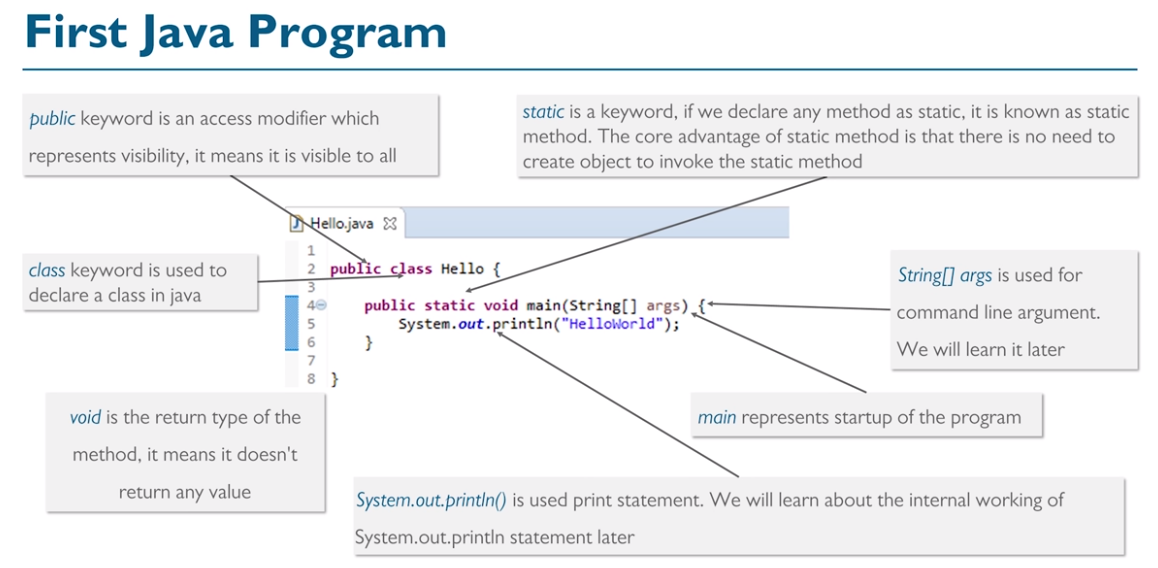
Java applications are called **WORA (Write Once Run Anywhere)**

**How Java is WORA:**

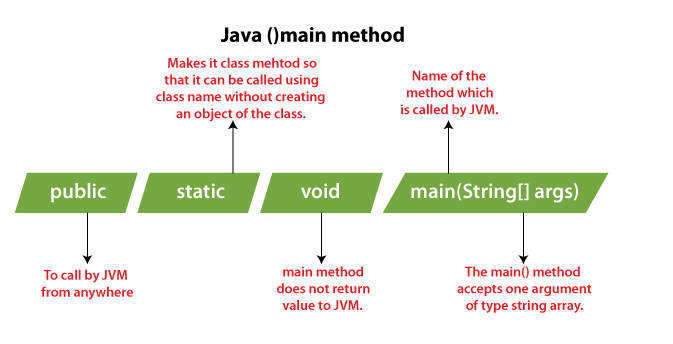
In traditional programming languages like C, C++ when programs were compiled, they used to be converted into the code understood by the particular underlying hardware, so If we try to run the same code at another machine with different hardware, which understands different code will cause an error, so you have to re-compile the code to be understood by the new hardware.

In Java, the program is not converted to code directly understood by Hardware, rather it is converted to [bytecode(.class file)](https://www.geeksforgeeks.org/java-class-file/), which is interpreted by JVM, so once compiled it generates bytecode file, which can be run anywhere (any machine) which has JVM( Java Virtual Machine) and hence it gets the nature of Write Once and Run Anywhere.





# We cannot modify the syntax of the **main()** method. The only thing which we can change is the name of the String array argument.



# How to Run Java Program in CMD Using Notepad

**Step 1:** Open the notepad

**Step 2:** Write a Java program that you want to compile and run

**tep 3:** To save a Java program press Ctrl + S key and provide the file name. Remember that the file name must be the same as the **class name** followed by the **.java** extension.

**Step 4:** To compile and run a Java program, open the **Command Prompt**

**Step 5:** In the Command Prompt window, write the following commands.

# CMD: basic commands you should know

<https://www.digitalcitizen.life/command-prompt-how-use-basic-commands/>

Text

Description automatically generated

**Step 6:** To compile the Java program type the following command:

**javac My.java**

Text

Description automatically generated

**Step 7:** To run the Java program, type the following command.

**java My**

Note that we have not written **.java** after the file name.

Text

Description automatically generated

# Java Basic Input and Output

1. **Command Line Arguments In Java**

**The console is an interface between the user and program**.

When a user enters the inputs on the console using commands, we sending the input as an argument to the main method in java that’s why in public static void main() we creating a string array to store values which work at executing time.

**Example code # 1 : With command line arguments :**

**class A**

**{**

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

**{**

**System.out.println(args[0]);**

**System.out.println(args[1]);**

**}**

**}**

**Output :Java**

**compilation:javac A.java**

**execution:java A 10 20**

**Output : 10 20**

### ****What Is Parse Method?****

# Syntax for a **command line arguments** for the conversion of a string into a number **(0, 1, 2, 3,N)** ” Parse Method “. From the following example, you will likely to learn how to pass the command line arguments as inputs with examples.

**PARSE #**

**PARSE : It is a method which take a string(input) as an argument and convert in other formats as like :**

* **Integer**
* **Float**
* **Double**

**THE FUNCTIONS OF PARSE METHODS :**

* **ParseInt();** – It is a member function in Integer Class;
* **ParseDouble();** – It is a member function in Double Class;
* **ParseFloat();** – It is a member function in Float Class;

**Syntax:Integer.parseInt(String s);**

**Example1:** String s=”156″; //156 is not a number it is string;

System.out.println(s+1);

//output:1561 :”156″+1=1561 string concatenation.

**int x=Integer.parseInt(“156”);  
System.out.println(x+1);**

**//output:157 :156+1=157**

**Example2:**

**class New {**

**public static void main (String args[]){**

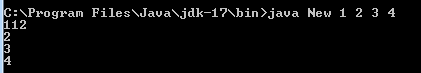
**System.out.println(args[0] + 12);**

**System.out.println(args[1]);**

**System.out.println(args[2]);**

**System.out.println(args[3]);**

**}}**



**Example3:**

**class New {**

**public static void main (String args[]){**

**int x = Integer.parseInt(args[0]);**

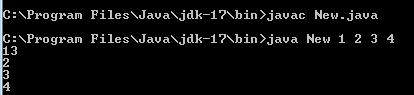
**System.out.println(x+ 12);**

**System.out.println(args[1]);**

**System.out.println(args[2]);**

**System.out.println(args[3]);**

**}}**



1. **Scanner class In Java**

After all that a new JDK was updated further by a scanner function was implemented in order to solve the problems much more confidently.

## Import Scanner Class

import java.util.Scanner;

## Java Scanner Methods to Take Input

The Scanner class provides various methods that allow us to read inputs of different types.



## Create a Scanner Object in Java

Scanner input = new Scanner(System.in);

Here, we have created an object of Scanner named input.

The System.in parameter is used to take input from the standard input. It works just like taking inputs from the keyboard.

import java.util.Scanner;

class Main {

public static void main(String[] args) {

// creates an object of Scanner

Scanner input = new Scanner(System.in);

System.out.print("Enter your name: ");

// takes input from the keyboard

String name = input.nextLine();

// prints the name

System.out.println("My name is " + name);

// closes the scanner

input.close();

}

}

import java.util.Scanner;

class Main {

public static void main(String[] args) {

// creates a Scanner object

Scanner input = new Scanner(System.in);

System.out.println("Enter an integer: ");

// reads an int value

int data1 = input.nextInt();

System.out.println("Using nextInt(): " + data1);

input.close();

}

}

To **read a [character in Java](https://www.javatpoint.com/post/java-character)**

, we use **next()** of the [Scanner class method](https://www.javatpoint.com/Scanner-class)

followed by chatAt() at method of the String class.

**import java.util.Scanner;**

**public class CharacterInputExample1**

**{**

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

**{**

**Scanner sc = new Scanner(System.in);**

**System.out.print("Input a character: ");**

**// reading a character**

**char c = sc.next().charAt(0);**

**//prints the character**

**System.out.println("You have entered "+c);**

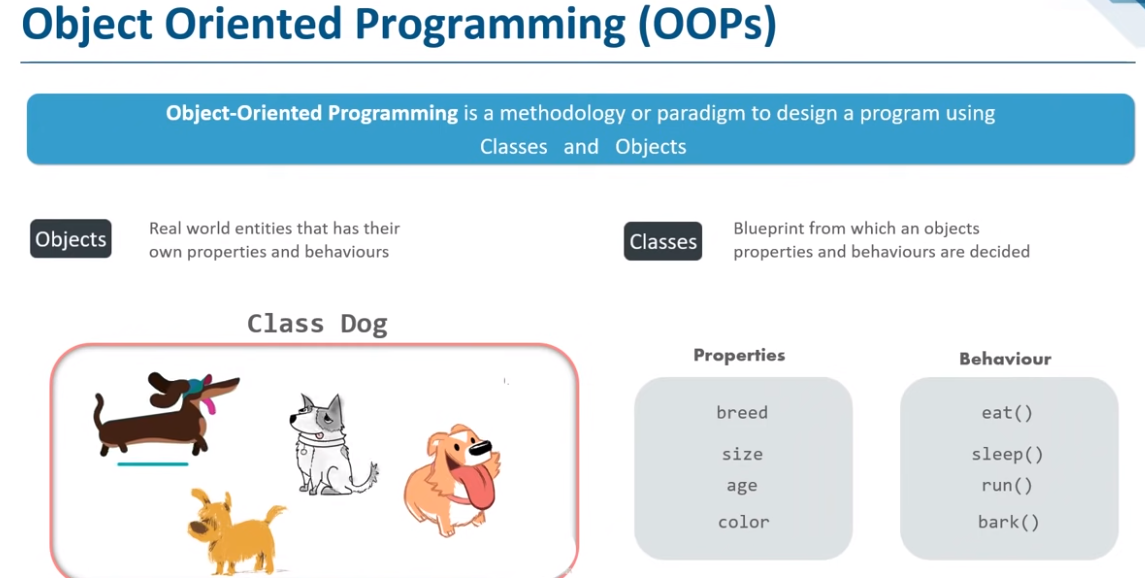
**}**

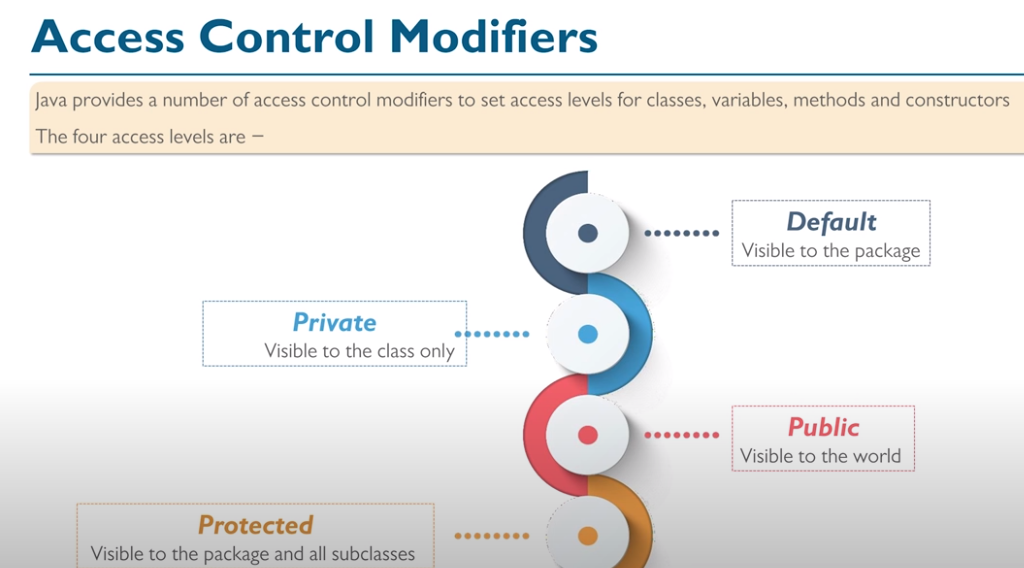
**}**

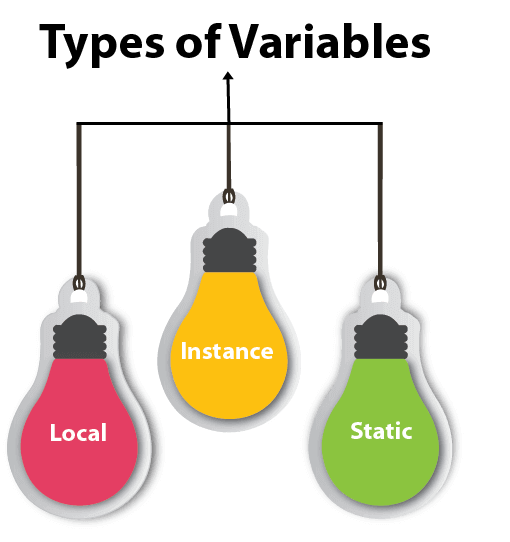
**OOP Concepts in Java**

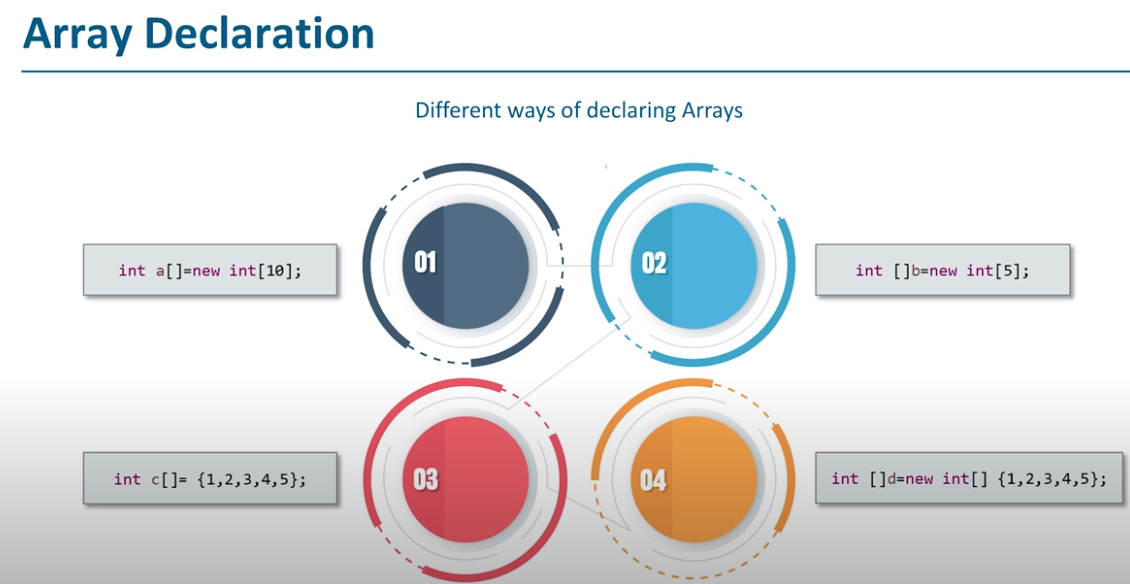
Java is a class-based object-oriented programming (OOP) language that is built around the concept of objects. OOP concepts (OOP) intend to improve code readability and reusability by defining how to structure a Java program efficiently. The main principles of object-oriented programming are:

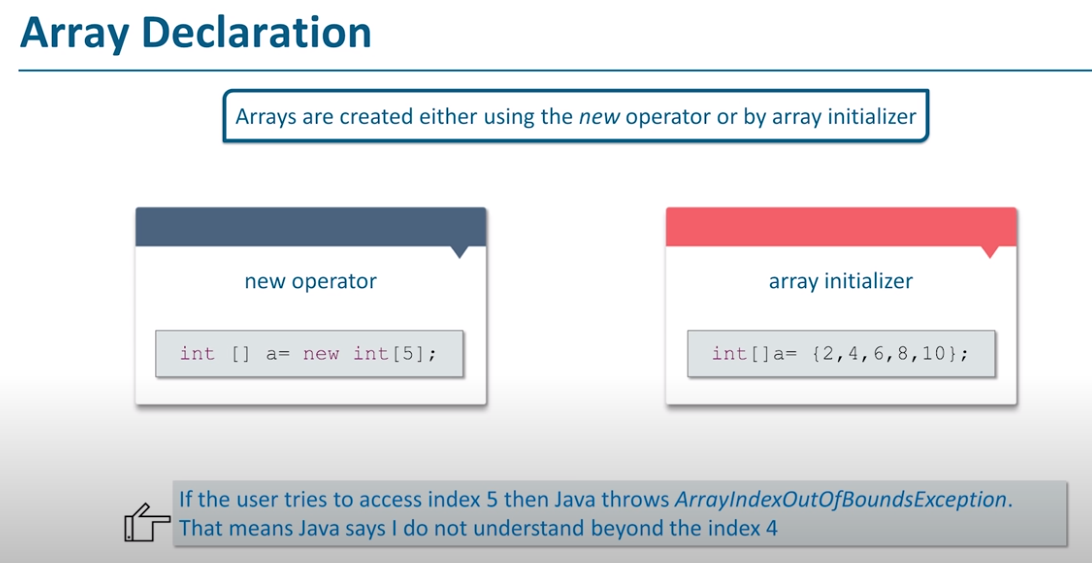
1. Object
2. Class
3. [Abstraction](https://raygun.com/blog/oop-concepts-java/#abstraction)
4. [Encapsulation](https://raygun.com/blog/oop-concepts-java/#encapsulation)
5. [Inheritance](https://raygun.com/blog/oop-concepts-java/#inheritance)
6. [Polymorphism](https://raygun.com/blog/oop-concepts-java/#polymorphism)





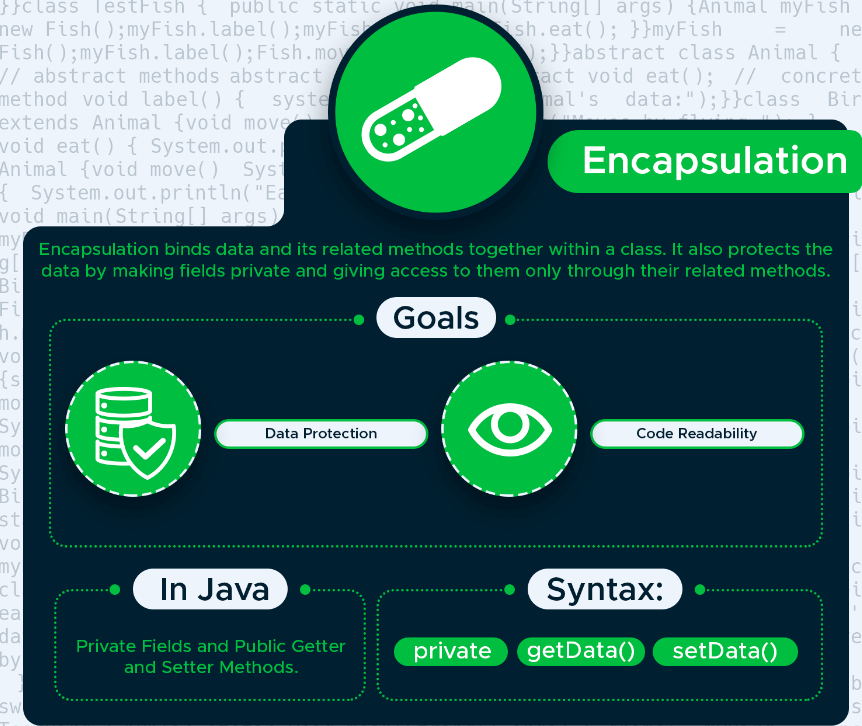






### Encapsulation

Encapsulation allows us to protect the data stored in a class from system-wide access. As its name suggests, it safeguards the internal contents of a class like a real-life capsule. You can implement encapsulation in Java by keeping the fields (class variables) private and providing public getter and setter methods to each of them. Java Beans are examples of fully encapsulated classes.



With encapsulation, you can protect the fields of a class. To do so, declare the fields as private and providing access to them with getter and setter methods.

The Animal class below is fully encapsulated. It has three private fields and each of them has its own set of getter and setter methods.

class Animal {

private String name;

private double averageWeight;

private int numberOfLegs;

// Getter methods

public String getName() {

return name;

}

public double getAverageWeight() {

return averageWeight;

}

public int getNumberOfLegs() {

return numberOfLegs;

}

// Setter methods

public void setName(String name) {

this.name = name;

}

public void setAverageWeight(double averageWeight) {

this.averageWeight = averageWeight;

}

public void setNumberOfLegs(int numberOfLegs) {

this.numberOfLegs = numberOfLegs;

}

}

The TestAnimal class first sets a value for each field with the setter methods, then prints out the values using the getter methods.

public class TestAnimal {

public static void main(String[] args) {

Animal myAnimal = new Animal();

myAnimal.setName("Eagle");

myAnimal.setAverageWeight(1.5);

myAnimal.setNumberOfLegs(2);

System.out.println("Name: " + myAnimal.getName());

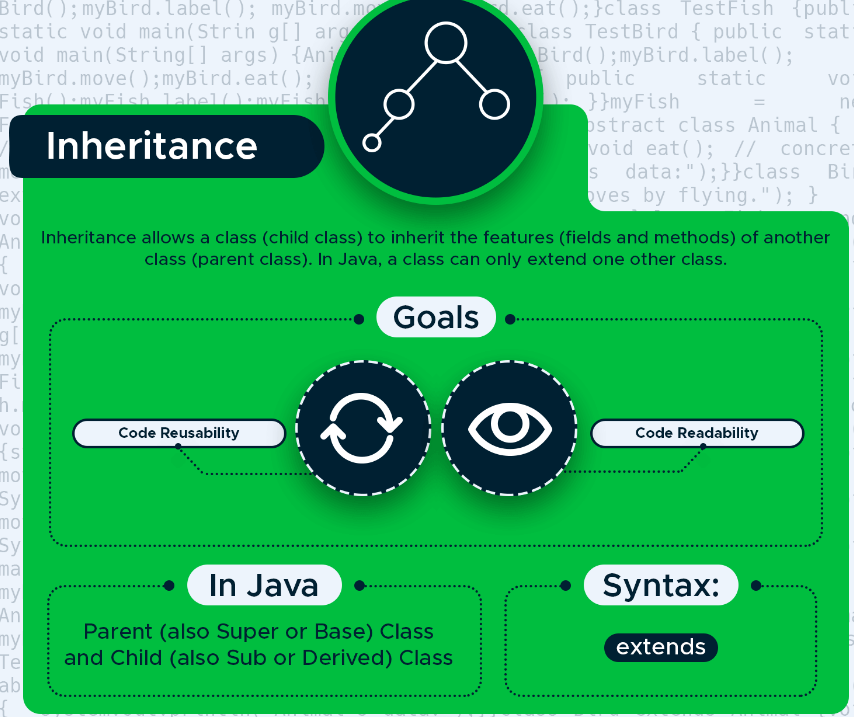
System.out.println("Average weight: " + myAnimal.getAverageWeight() + "kg");

System.out.println("Number of legs: " + myAnimal.getNumberOfLegs());

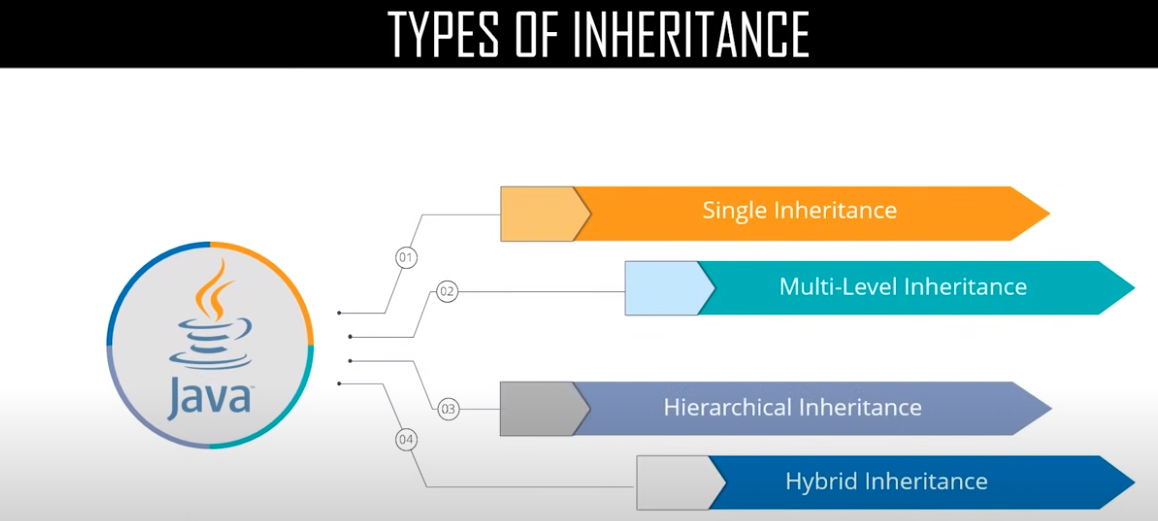
}

}

### Inheritance



Inheritance allows us to extend a class with child classes that inherit the fields and methods of the parent class. It’s an excellent way to achieve code reusability. In Java, we need to use the extends keyword to create a child class.



In the example, the Eagle class extends the Bird parent class. It inherits all of its fields and methods, plus defines two extra fields that belong only to Eagle.

class Bird {

public String reproduction = "egg";

public String outerCovering = "feather";

public void flyUp() {

System.out.println("Flying up...");

}

public void flyDown() {

System.out.println("Flying down...");

}

}

class Eagle extends Bird {

public String name = "eagle";

public int lifespan = 15;

}

The TestEagle class instantiates a new Eagle object and prints out all the information (both the inherited fields and methods and the two extra fields defined in the Eagle class).

class TestEagle {

public static void main(String[] args) {

Eagle myEagle = new Eagle();

System.out.println("Name: " + myEagle.name); System.out.println("Reproduction: " + myEagle.reproduction);

System.out.println("Outer covering: " + myEagle.outerCovering);

System.out.println("Lifespan: " + myEagle.lifespan);

myEagle.flyUp();

myEagle.flyDown();

}

}