Programming in Java [TCSCCS0302T]

~ by Asst. Prof. Shivkumar Chandey
Department of Computer Science
Thakur College of Science & Commerce (Autonomous)

Objective

- The objective of this course is to teach the learner how to use building blocks of Core Java Language to implement a real-world and functional CUI and GUI based Application by using Object Oriented paradigm.

Syllabus of Unit II

Unit II	Java I/O System[A]: Scope of Java I/O System, Input and Output[A], Streams, The System Class [java.lang], Byte Streams[A], Character Streams[A]. Object-Oriented Programming: OOPs Concept[A]: Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation, Methods, Abstract Classes and Interfaces, Packages and Imports, Garbage Collection, Enumerations, Autoboxing and Unboxing, Exception Handling, Wrapper Classes.

Expected Learning Outcomes

1. Learner will get a strong knowledge of the structure and model of the Java programming language
2. Implement Various Java Applications and Software by using concepts of Java programming language.
3. Develop a strong foundation of OOPs Concept, AWT, Threads, Java APIs.
4. Evaluate user requirements and Design responsive GUI based applications

Java I/O

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- We can perform file handling in Java by Java I/O API.

Stream

- A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.

  1) System.out: standard output stream

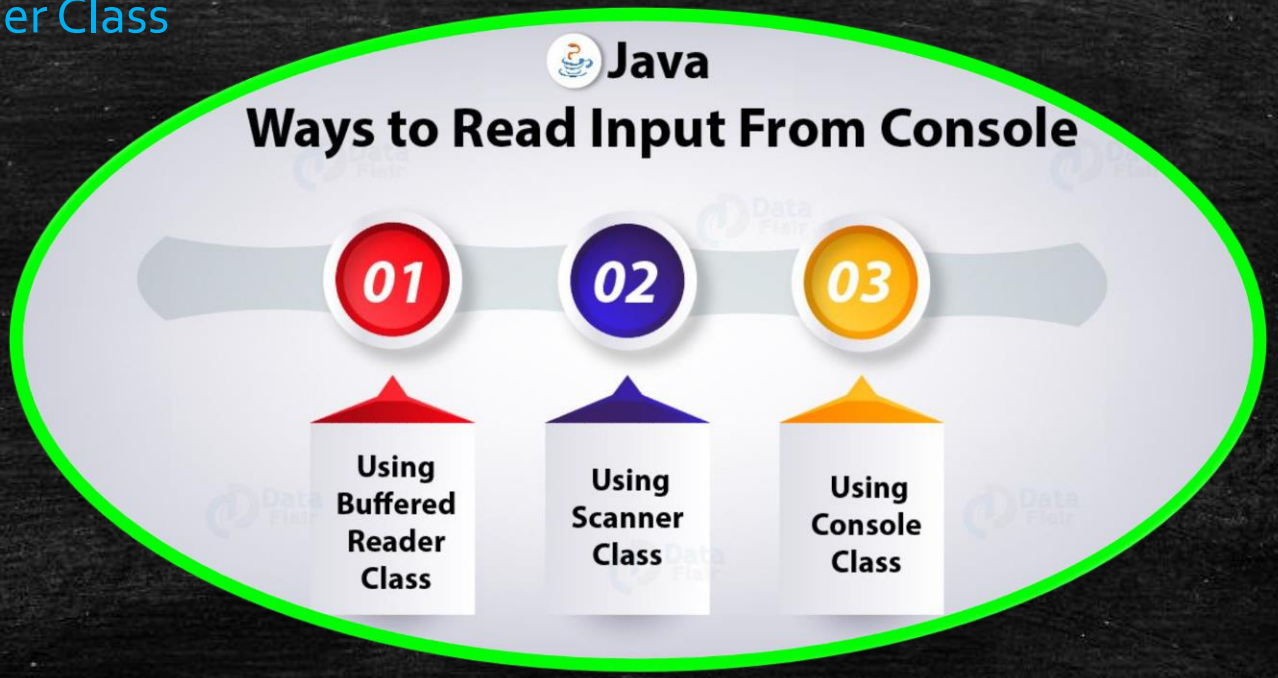
  2) System.in: standard input stream

  3) System.err: standard error stream

Ways to read input from console in Java

- In Java, there are three different ways for reading input from the user in the command line environment(console).

1) Using Buffered Reader Class
2) Using Scanner Class
3) Using Console Class



Buffered Reader Class

- Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.
- **Advantages**
  - The input is buffered for efficient reading.
- **Drawback:**
  - The wrapping code is hard to remember.

BufferedReader class declaration

- Let's see the declaration for Java.io.BufferedReader class:

  public class BufferedReader **extends** Reader

BufferedReader class constructors

Constructor	Description
BufferedReader(Reader rd)	It is used to create a buffered character input stream that uses the default size for an input buffer.
BufferedReader(Reader rd, int size)	It is used to create a buffered character input stream that uses the specified size for an input buffer.

BufferedReader class methods

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.
long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class BufferDemo
{
        public static void main(String[] args) throws IOException
        {
                //Enter data using BufferReader
                BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

                // Reading data using readLine
                String name = reader.readLine();

                // Printing the read line
                System.out.println(name);
        }
}
```

```
//Example 2
import java.io.*;
public class BufferedReaderExample
{
        public static void main(String args[])throws Exception
        {
                InputStreamReader r=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(r);
                System.out.println("Enter your name");
                String name=br.readLine();
                System.out.println("Welcome, "+name);
        }
}
```

Scanner Class

- This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it is also can be used to read input from the user in the command line.
- Advantages:
  - Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
  - Regular expressions can be used to find tokens.
- Drawbacks:
  - The reading methods are not synchronized

Scanner Class

- Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.
- The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

Scanner Class

- The Java Scanner class extends Object class and implements Iterator and Closeable interfaces.
- The Java Scanner class provides nextXXX() methods to return the type of value such as
  - nextInt(),
  - nextByte(),
  - nextShort(),
  - next(),
  - nextLine(),
  - nextBoolean(), etc.
- To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.

Java Scanner Class Declaration

```
public final class Scanner
    extends Object
    implements Iterator<String>
```

How to get Java Scanner

- To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:
  - Scanner in = **new** Scanner(System.in);
- To get the instance of Java Scanner which parses the strings, we need to pass the strings in the constructor of Scanner class. For Example:
  - Scanner in = **new** Scanner("Hello Students");

Java Scanner Class Constructors

SN	Constructor	Description
1)	Scanner(File source)	It constructs a new Scanner that produces values scanned from the specified file.
2)	Scanner(File source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified file.
3)	Scanner(InputStream source)	It constructs a new Scanner that produces values scanned from the specified input stream.
4)	Scanner(InputStream source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified input stream.
5)	Scanner(Readable source)	It constructs a new Scanner that produces values scanned from the specified source.
6)	Scanner(String source)	It constructs a new Scanner that produces values scanned from the specified string.
7)	Scanner(ReadableByteChannel source)	It constructs a new Scanner that produces values scanned from the specified channel.
8)	Scanner(ReadableByteChannel source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified channel.
9)	Scanner(Path source)	It constructs a new Scanner that produces values scanned from the specified file.
10)	Scanner(Path source, String charsetName)	It constructs a new Scanner that produces values scanned from the specified file.

Java Scanner Class Methods

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
close()	It is used to close this scanner.

```
//Scanner Example
//Reading String from user.

import java.util.*;
public class ScannerExample
{
        public static void main(String args[])
        {
                Scanner in = new Scanner(System.in);
                System.out.print("Enter your name: ");
                String name = in.nextLine();
                System.out.println("Name is: " + name);
                in.close();
    }
}
```

Using Console Class

- The Java Console class is used to get input from console. It provides methods to read texts and passwords.
- If you read password using Console class, it will not be displayed to the user.
- The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.
- Let's see a simple example to read text from console.

```
String text=System.console().readLine();

System.out.println("Text is: "+text);
```

Console class declaration

- Let's see the declaration for Java.io.Console class:

  public final class Console **extends** Object **implements** Flushable

Console class methods

Method	Description
Reader reader()	It is used to retrieve the reader object associated with the console
String readLine()	It is used to read a single line of text from the console.
String readLine(String fmt, Object... args)	It provides a formatted prompt then reads the single line of text from the console.
char[] readPassword()	It is used to read password that is not being displayed on the console.
char[] readPassword(String fmt, Object... args)	It provides a formatted prompt then reads the password that is not being displayed on the console.
Console format(String fmt, Object... args)	It is used to write a formatted string to the console output stream.
Console printf(String format, Object... args)	It is used to write a string to the console output stream.
PrintWriter writer()	It is used to retrieve the PrintWriter object associated with the console.
void flush()	It is used to flushes the console.

Using Console Class

- Advantages:
  - Reading password without echoing the entered characters.
  - Reading methods are synchronized.
  - Format string syntax can be used.
- Drawback:
  - Does not work in non-interactive environment (such as in an IDE).

```
//Console Class Example
import java.io.Console;
class consoleDemo
{
        public static void main(String args[])
        {
                Console c=System.console();
                System.out.println("Enter your name: ");
                String n=c.readLine();
                System.out.println("Welcome "+n);
        }
}
```

```
//Reading password using Console Class

import java.io.Console;
class ReadPasswordTest
{
        public static void main(String args[])
        {
                Console c=System.console();
                System.out.println("Enter password: ");
                char[] ch=c.readPassword();
                String pass=String.valueOf(ch);//converting char array into string
                System.out.println("Password is: "+pass);
        }
}
```

The System Class

- The System class of java contains several useful class fields and methods. It also provides facilities like standard input, standard output, and error output Streams. It can't be instantiated
- The Java System class comes in the module of "java.base" & in the package of "java.lang".
- In Java System Class, we have 3 different types of field and 28 different types of method.
- Java System Class consists of following fields:-

Fields of System Class

SN	Modifier and Type	Field	Description
1	static PrintStrean	err	The "standard" error output stream.
2	static InputStream	in	The "standard" input stream.
3	static PrintStream	out	The "standard" output stream.

Java Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

Example of command-line argument in java

- Simple example of command-line argument in java

```
class CommandLineExample
{
        public static void main(String args[])
        {
                System.out.println("Your first argument is: "+args[0]);
        }
}
```

Passing arguments at runtime

- compile by:

  javac CommandLineExample.java

- run by:

  java CommandLineExample Shiv

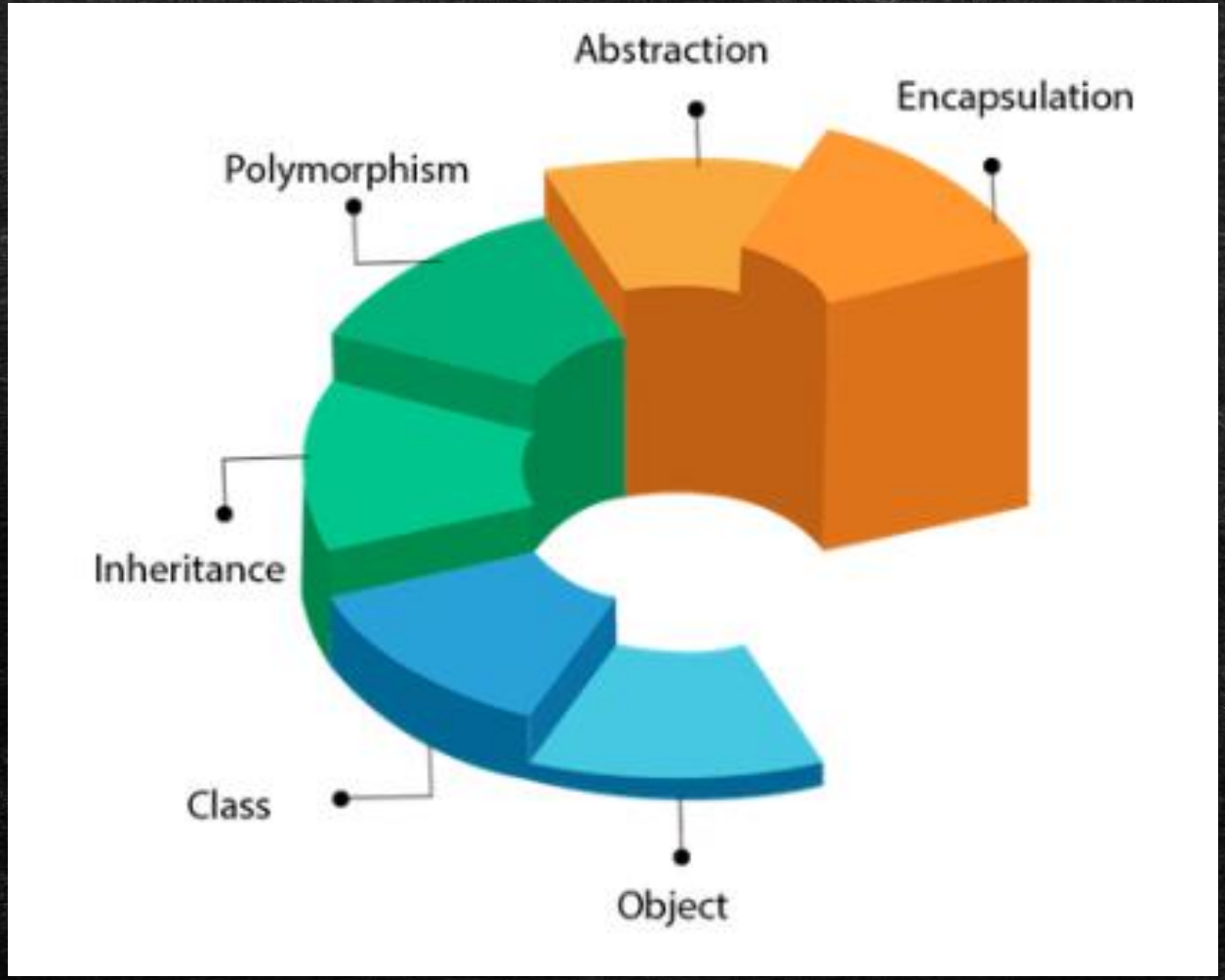
Reading multiple arguments

```
class CLA
{
        public static void main(String args[])
        {
                for(int i=o;i<args.length;i++)
                {
                        System.out.println(args[i]);
                }
        }
}
```

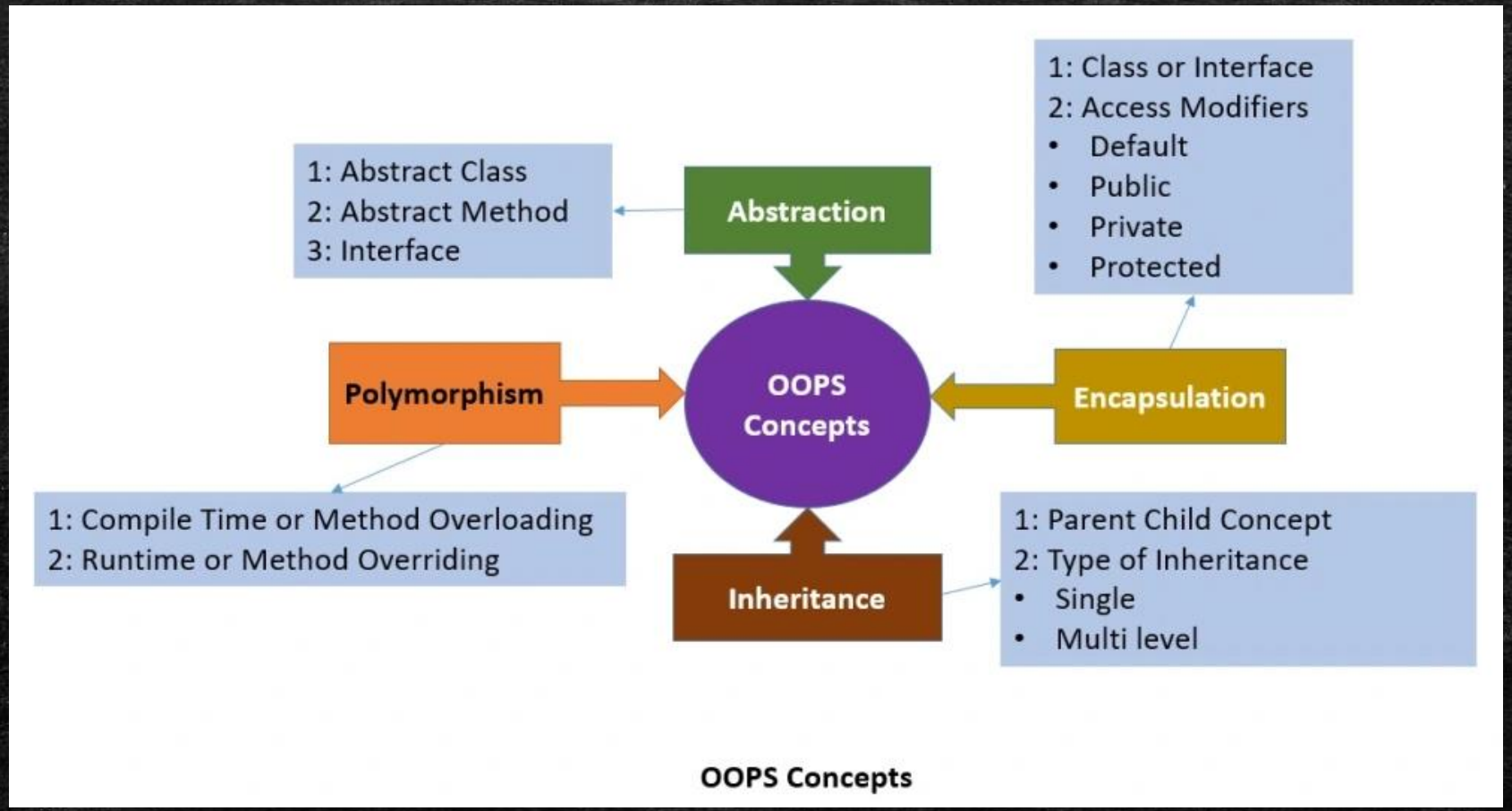
What is OOPs?

- Object-oriented programming is a method used for designing a program using classes and objects.
- Object-oriented programming organizes a program around objects and well-defined interfaces. This can also be characterized as data controlling for accessing the code.
- This implies software development and maintenance by using some of the concepts:
  - Object
  - Class
  - Abstraction
  - Inheritance
  - Polymorphism
  - Encapsulation

OOPs Concept



OOPs Concept



OOPS Concepts

Objects

- Objects are always called as instances of a class.
- Objects are created from class in java or any other languages.
- Objects are those that have state and behavior. Objects are abstract data types (i.e., objects behavior is defined by a set of values and operations).
- These objects always correspond to things found in the real world, i.e., real entities. So, they are also called a run time entity of the world.
- These are self–contained which consists of methods and properties which makes data useful.

Objects

- Objects can be both physical and logical data. It contains addresses and takes up some space in memory. Some of the examples of objects are a dog, chair, tree etc.
- When we treat animals as objects, it has states like colour, name, breed etc., and behaviours such as eating, wagging the tail etc.
- Suppose, we have created a class called My book, we specify the class name followed by the object name, and we use the keyword new.

Example

```
Public class Mybook
{
    int x=10;
    Public static void main (String args [])
     {
       Mybook Myobj= new Mybook ();
       System.out.println(MyObj.x);
     }
}
```

In this example, a new object is created, and it returns the value of x which may be the number of books.

Mybook Myobj= new Mybook ();

This is the statement used for creating objects.

System.out.println(Myobj.x);

This statement is used to return the value of x of an object.

Class

- Classes are like object constructors for creating objects.
- The collection of objects is said to be a class. Classes are said to be logical quantities.
- Classes don't consume any space in the memory.
- Class is also called a template of an object.
- Classes have members which can be fields, methods and constructors. A class has both static and instance initializers.

A class declaration consists of:

- Modifiers: Can be public or default access.
- Class name: Initial letter.
- Superclass: A class can only extend (subclass) one parent.
- Interfaces: A class can implement more than one interface.
- Body: Body surrounded by braces, { }.
- A class keyword is used to create a class. A simplified general form of the class definition is given below:

```
class classname
{
      type instance variable 1;
      type instance variable 2;
      .
      .
      .
      type instance variable n;
      type methodname 1 (parameter list)
      {
            // body od method
      }
      type methodname 2 (parameter list)
      {
            // body od method
      }
      type methodnamen (parameter list)
      {
            // body od method
      }
}
```

The variables or data defined within a class are called as instance variables.

Code is always contained in the methods. Therefore, the methods and variables defined within a class are called members of the class.

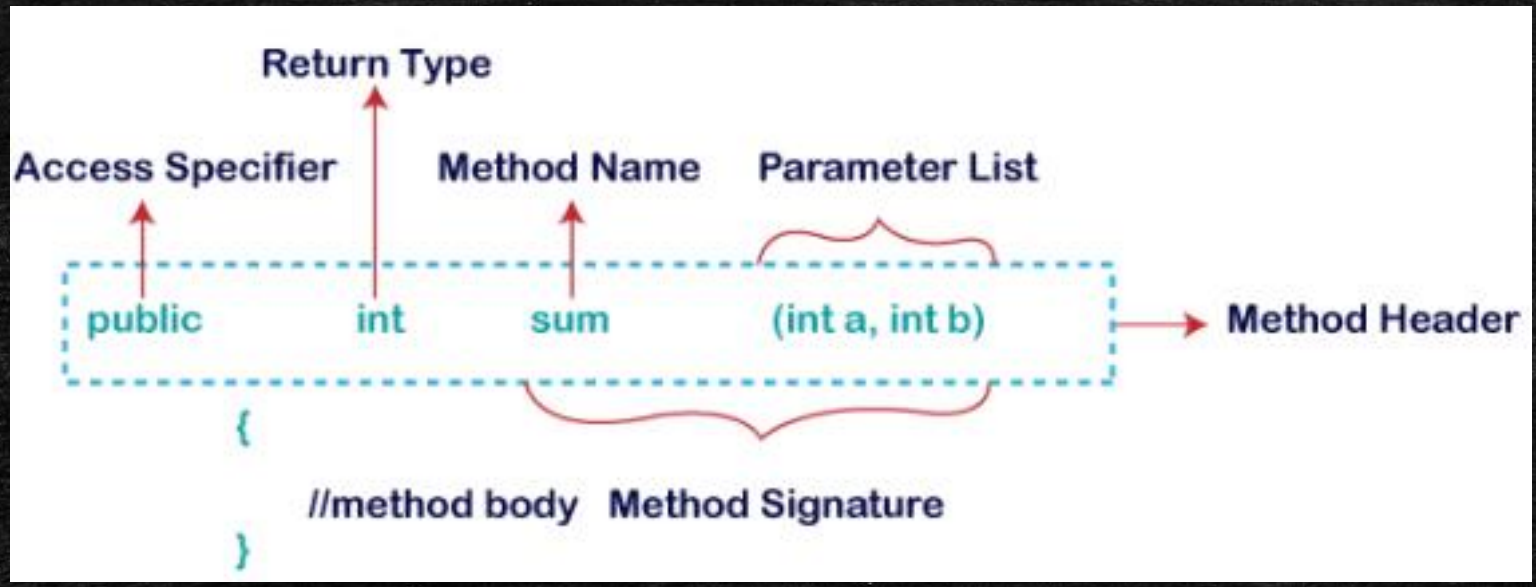
All the methods have the same form as main () these methods are not specified as static or public.

What is a method in Java?

- A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- It is used to achieve the reusability of code.
- It also provides the easy modification and readability of code, just by adding or removing a chunk of code.
- The method is executed only when we call or invoke it.
- The most important method in Java is the main() method.

Method Declaration

- The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.
- It has six components that are known as method header, as we have shown in the following figure.



Method Signature

- Every method has a method signature.
- It is a part of the method declaration.
- It includes the method name and parameter list.

Access Specifier

- Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides four types of access specifier:
  - Public: The method is accessible by all classes when we use public specifier in our application.
  - Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.
  - Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
  - Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type

- Return type is a data type that the method returns.
- It may have a primitive data type, object, collection, void, etc.
- If the method does not return anything, we use void keyword.

Method Name

- It is a unique name that is used to define the name of a method.
- It must be corresponding to the functionality of the method.
- Suppose, if we are creating a method for subtraction of two numbers, the method name must be subtraction().
- A method is invoked by its name.

Parameter List

- It is the list of parameters separated by a comma and enclosed in the pair of parentheses.
- It contains the data type and variable name.
- If the method has no parameter, leave the parentheses blank.

Method Body

- It is a part of the method declaration.
- It contains all the actions to be performed.
- It is enclosed within the pair of curly braces.

Naming a Method

- While defining a method, remember that the method name must be a verb and start with a lowercase letter.
- If the method name has more than two words, the first name must be a verb followed by adjective or noun.
- In the multi-word method name, the first letter of each word must be in uppercase except the first word.
- For example:
  - Single-word method name: sum(), area()
  - Multi-word method name: areaOfCircle(), stringComparision()
- It is also possible that a method has the same name as another method name in the same class, it is known as method overloading.

Types of Method

- There are two types of methods in Java:

1. Predefined Method
2. User-defined Method

Predefined Method

- In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.
- It is also known as the standard library method or built-in method.
- We can directly use these methods just by calling them in the program at any point.
- Some pre-defined methods are length(), equals(), compareTo(), sqrt(), etc.
- When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.
- Each and every predefined method is defined inside a class. Such as print() method is defined in the java.io.PrintStream class. It prints the statement that we write inside the method.
- For example, print("Java"), it prints Java on the console.

User-defined Method

- The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.
- Let's create a user defined method that checks the number is even or odd. First, we will define the method.

user defined method definition

```
public static void findEvenOdd(int num)
{
//method body
if(num%2==0)
        System.out.println(num+" is even");
else
        System.out.println(num+" is odd");
}
```

How to Call or Invoke a User-defined Method?

- Once we have defined a method, it should be called.
- When we call or invoke a user-defined method, the program control transfer to the called method.

Static Method

- A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method.
- We can also create a static method by using the keyword static before the method name.
- The main advantage of a static method is that we can call it without creating an object.
- It can access static data members and also change the value of it.
- It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the main() method.

Example of static method
Display.java

```
public class Display
{
	public static void main(String[] args)
	{
		show();
	}
	static void show()
	{
		System.out.println("It is an example of static method.");
	}
}
```

Instance Method

- The method of the class is known as an instance method.
- It is a non-static method defined in the class.
- Before calling or invoking the instance method, it is necessary to create an object of its class.
- Let's see an example of an instance method.

Abstraction

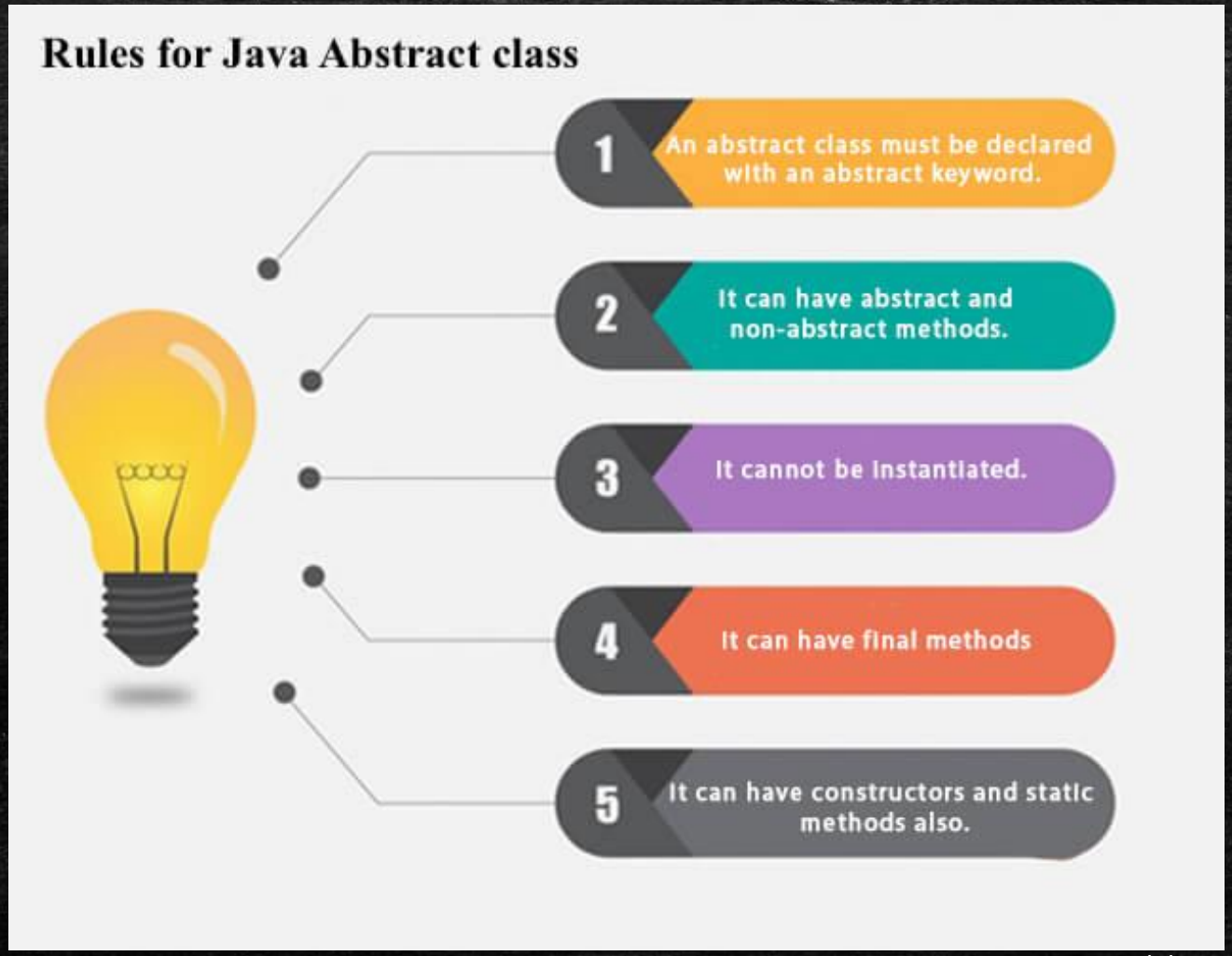
- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
- Abstraction lets you focus on what the object does instead of how it does it.
- Ways to achieve Abstraction
- There are two ways to achieve abstraction in java
  - Abstract class (0 to 100%)
  - Interface (100%)

Abstract class

- A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods.
- It needs to be extended and its method implemented. It cannot be instantiated.
- Example of abstract class

```
abstract class A
{
}
```

Rules for Abstract class



Abstract Method in Java

- A method which is declared as abstract and does not have implementation is known as an abstract method.
- Example of abstract method

```
abstract void printStatus();//no method body and abstract
```

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 constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Constructors

- There are two types of constructors in Java: no-arg constructor, and parameterized constructor.
- Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

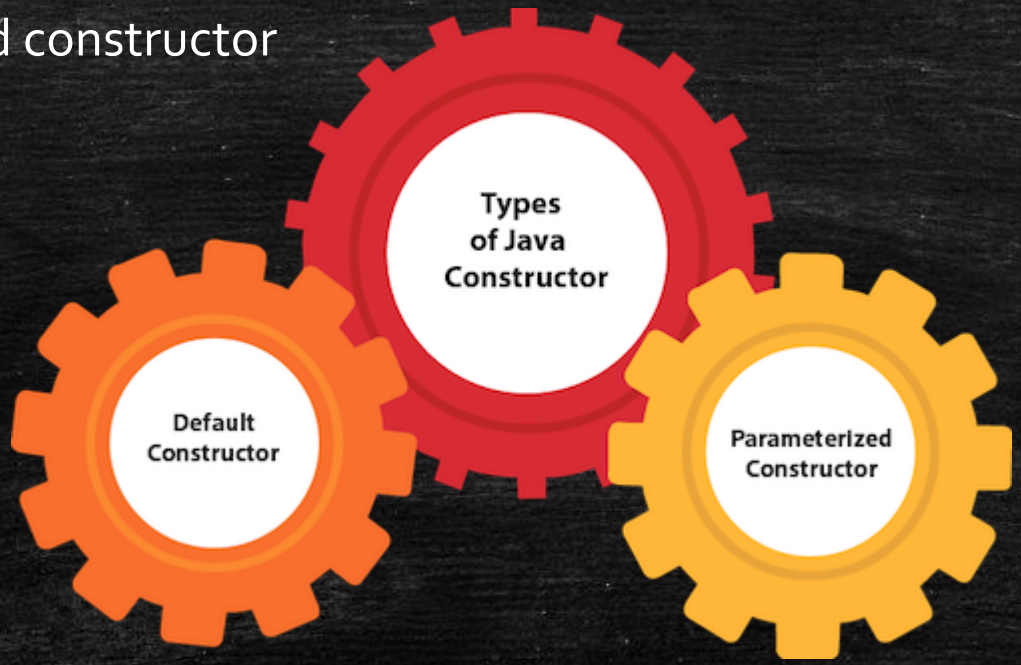
- There are few rules defined for the constructor.

1. Constructor name must be the same as its class name.
2. A Constructor must have no explicit return type.
3. A Java constructor cannot be abstract, static, final, and synchronized.

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

- There are two types of constructors in Java:
  - Default constructor (no-arg constructor)
  - Parameterized constructor



Java Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- Syntax of default constructor:

```
<class_name>()
{
}
```

//Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1
{
        //creating a default constructor
        Bike1()
        {
                System.out.println("Bike is created");
        }
        //main method
        public static void main(String args[])
        {
                //calling a default constructor
                 Bike1 b=new Bike1();
        }
}
```

Java Parameterized Constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.
- Why use the parameterized constructor?
  - The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
   int id;
   String name;
   //creating a parameterized constructor
   Student4(int i,String n){
   id = i;
   name = n;
   }
   //method to display the values
   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){
   //creating objects and passing values
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   //calling method to display the values of object
   s1.display();
   s2.display();
  }
}
```

Difference between constructor and method in Java

Constructor		Method
A constructor is used to initialize the state of an object.	1	A method is used to expose the behavior of an object.
A constructor must not have a return type.	2	A method must have a return type.
The constructor is invoked implicitly.	3	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	4	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	5	The method name may or may not be same as class name.

Inheritance

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Why use inheritance in java?

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

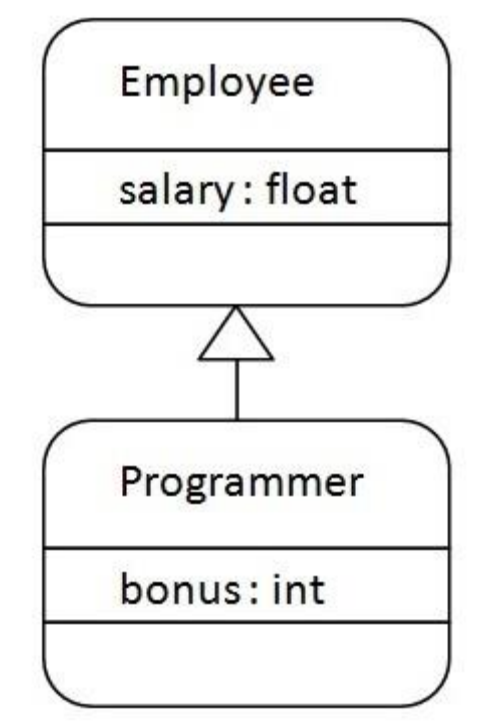
- Syntax:

```
class Subclass-name extends Superclass-name
{
        //methods and fields
}
```

- The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
- In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Inheritance Example

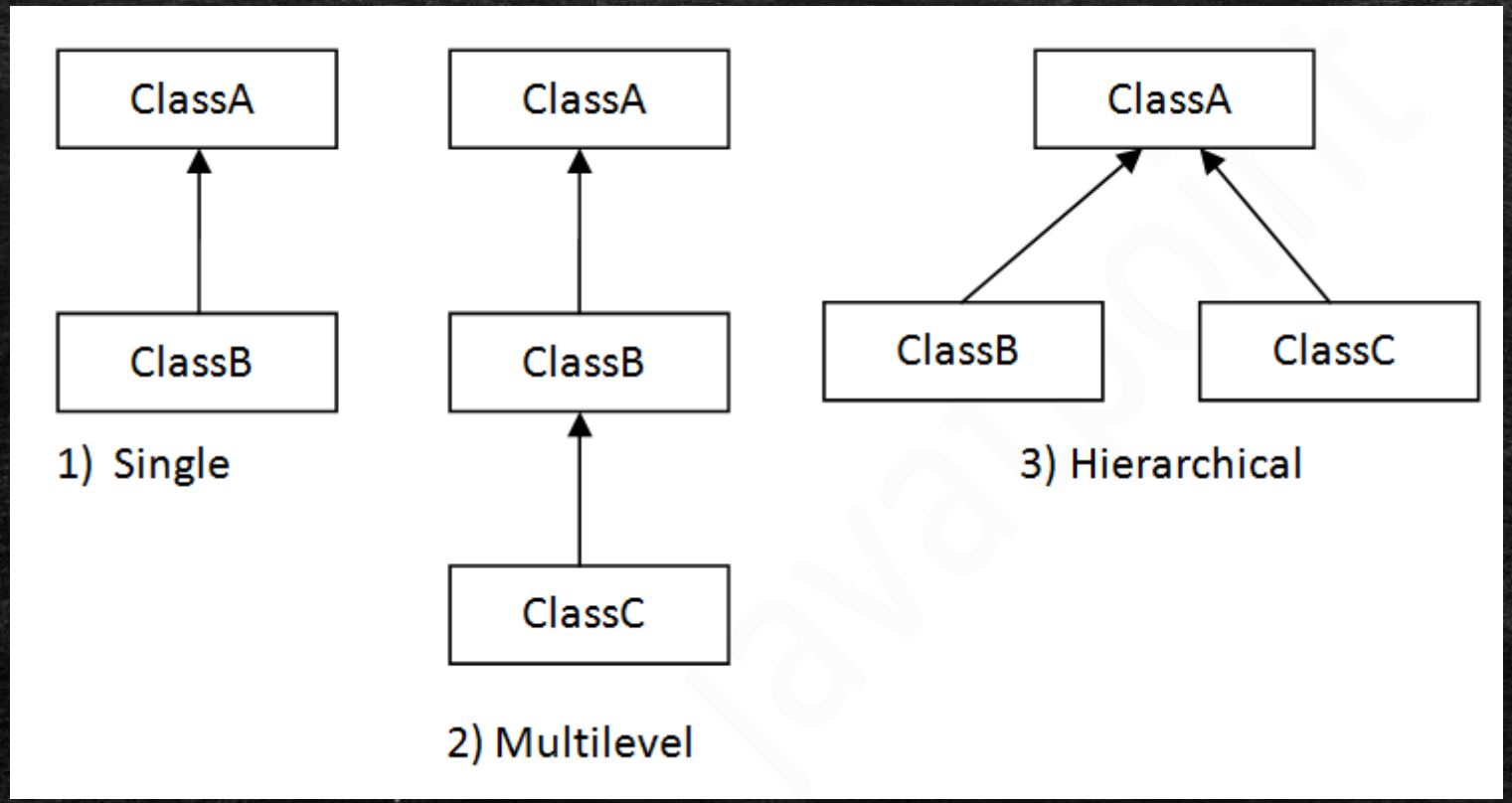
- As displayed in the figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.



Types of inheritance in Java

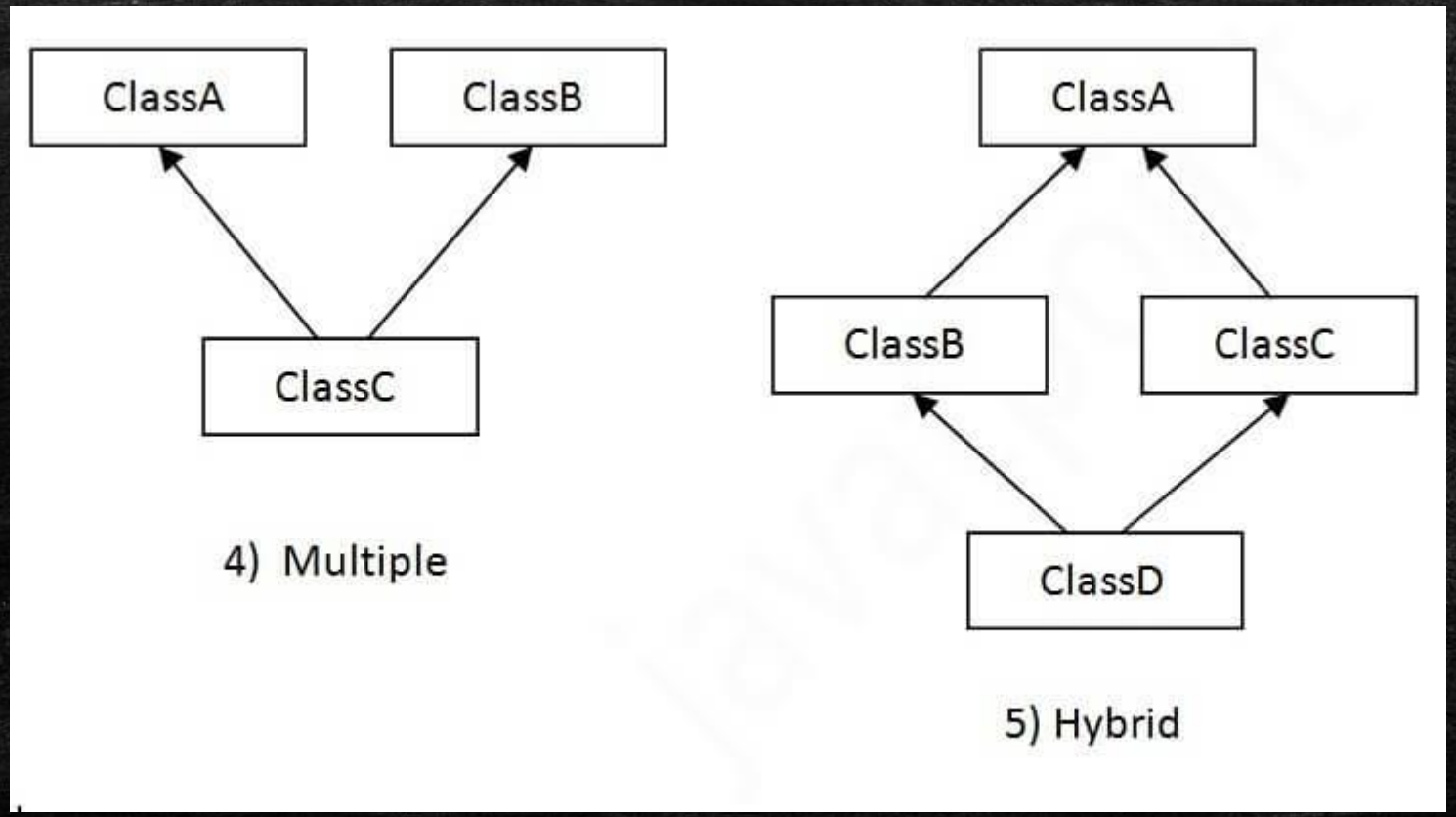
- On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.
- In java programming, multiple and hybrid inheritance is supported through interface only.
- Note: Multiple inheritance is not supported in Java through class.

Types of inheritance in Java



Multiple Inheritance

- When one class inherits multiple classes, it is known as multiple inheritance. For Example:



Single Inheritance Example

- When a class inherits another class, it is known as a single inheritance.

Multilevel Inheritance Example

- When there is a chain of inheritance, it is known as multilevel inheritance.

Hierarchical Inheritance Example

- When two or more classes inherits a single class, it is known as hierarchical inheritance.

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
- Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Polymorphism in Java

- Polymorphism in Java is a concept by which we can perform a single action in different ways.
- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.
- If you overload a static method in Java, it is the example of compile time polymorphism.

Runtime Polymorphism in Java

- Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- In this process, an overridden method is called through the reference variable of a superclass.

Method overloading v/s Method overriding

No.	Method Overloading	Method Overriding
1)	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2)	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3)	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4)	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5)	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

```
class Bike
{
      void run()
      {
      System.out.println("running");
      }
}
class Splendor extends Bike
{
       void run()
      {
      System.out.println("running safely with 60km");
      }

       public static void main(String args[])
      {
        Bike b = new Splendor();//upcasting
        b.run();
       }
}
```

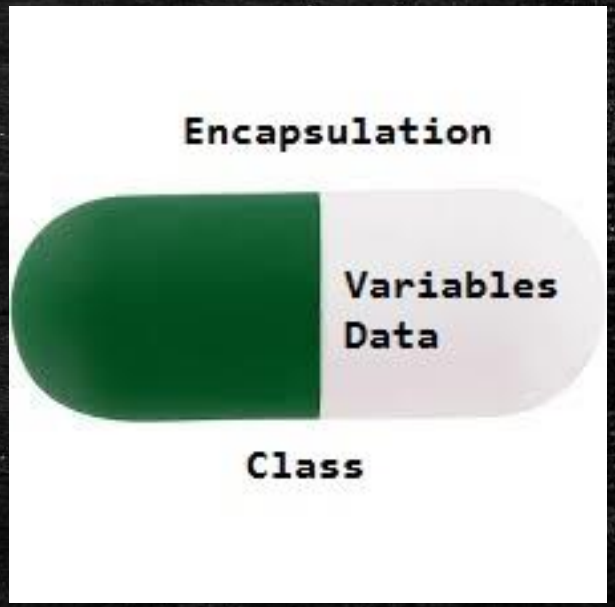
In this example, we are creating two classes Bike and Splendor. Splendor class extends Bike class and overrides its run() method.

We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, the subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

Encapsulation in Java

- Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.
- We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.
- The Java Bean class is the example of a fully encapsulated class.



Benefits of Encapsulation

- Data Hiding: Here, a user will have no idea about the inner implementation of the class. Even user will not be aware of how the class is storing values in the variables.
- Increased Flexibility: Here, we can make the variables of the class as read-only or write-only depending on our requirement.
- Reusability: It also improves the re-usability and easy to change with new requirements.

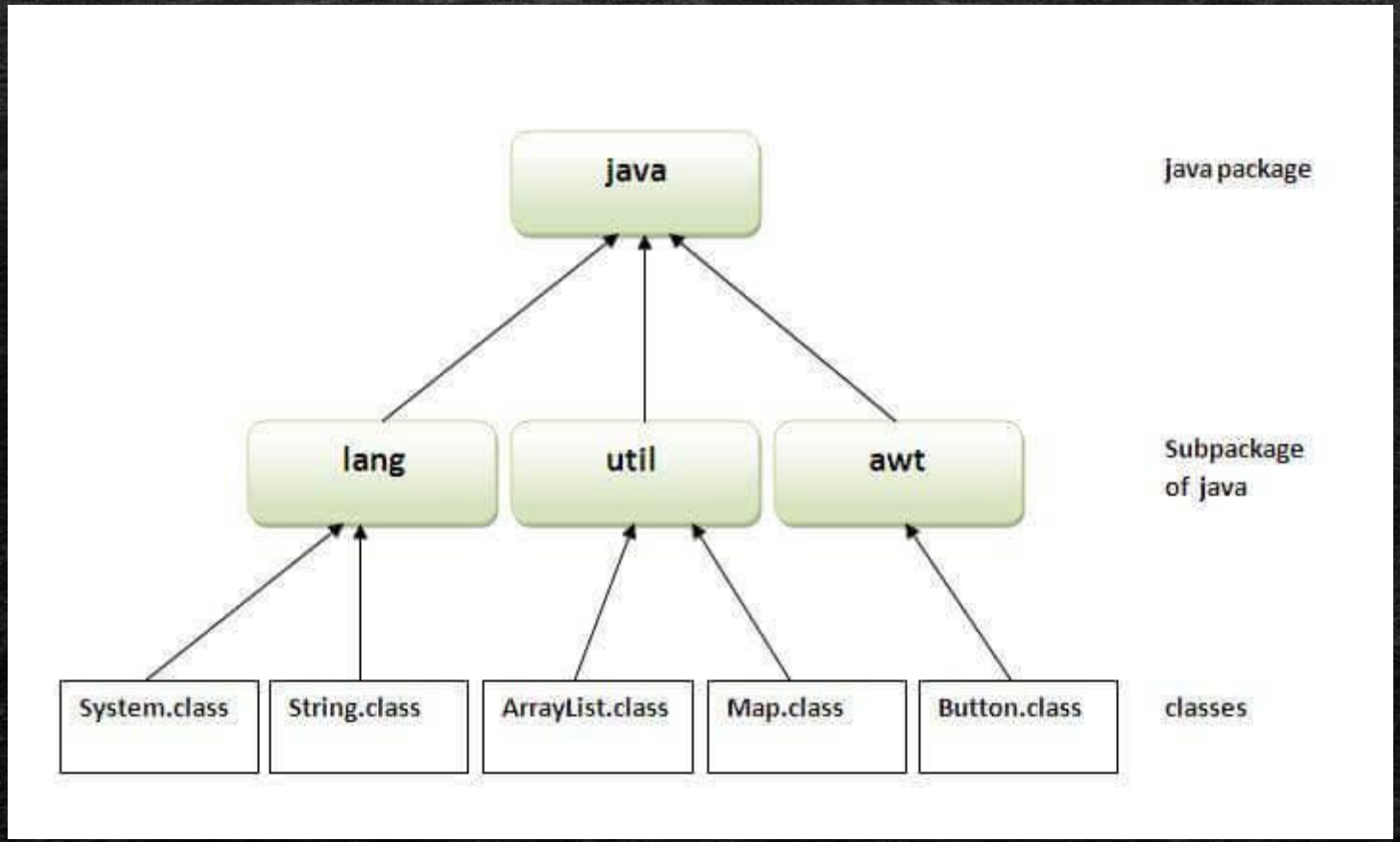
Package

- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2. Java package provides access protection.
3. Java package removes naming collision.

Understand hierarchy of packages



How to access package from another package?

- There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

Wrapper classes in Java

- The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.
- Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Wrapper classes in Java

- The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing

- The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.
- Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

Wrapper class Example: Primitive to Wrapper

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1
{
public static void main(String args[])
{
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}
}
```

Unboxing

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.
- Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

Wrapper class Example: Wrapper to Primitive

```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2
{
public static void main(String args[])
{
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}
}
```

Exception Handling

- The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.
- **Dictionary Meaning:** Exception is an abnormal condition.
- **Exception:** In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Advantage of Exception Handling

- The core advantage of exception handling is to maintain the normal flow of the application.
- An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.
- Let's consider a scenario:

Example

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

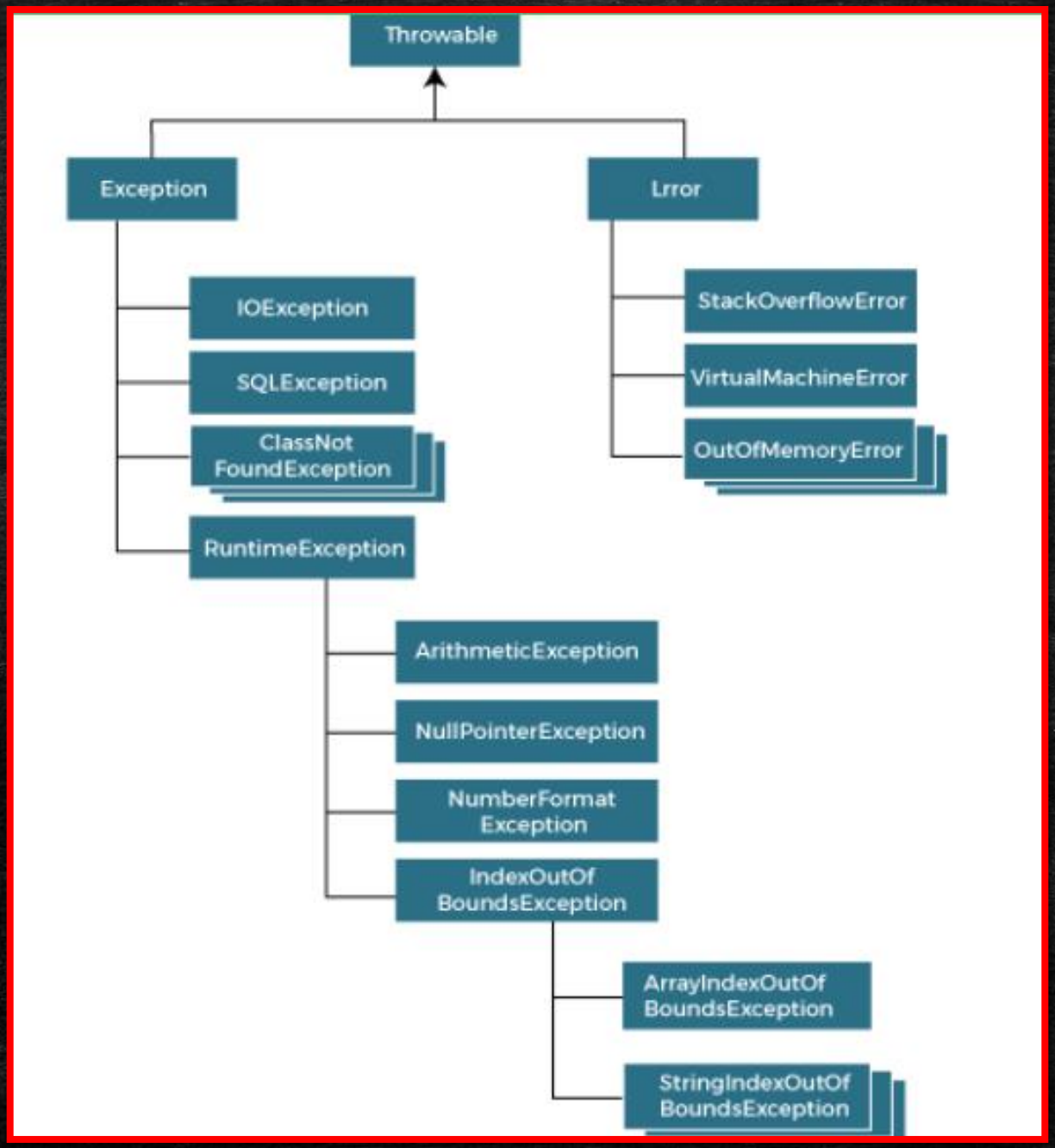
statement 10;

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed.

However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

Hierarchy of Java Exception classes

- The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:


Throwable
Exception
Lrror
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

Java Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

1) By nulling a reference

Employee e=new Employee();

e=null;

2) By assigning a reference to another

Employee e1=new Employee();

Employee e2=new Employee();

e1=e2;//now the first object referred by e1 is available for garbage collection

3) By anonymous object

new Employee();

Simple Example of garbage collection in java

```
public class TestGarbage1
{
 public void finalize()
{
                System.out.println("object is garbage collected");
}
 public static void main(String args[])
{
 TestGarbage1 s1=new TestGarbage1();
 TestGarbage1 s2=new TestGarbage1();
 s1=null;
 s2=null;
 System.gc();
 }
}
```

References

- Text book:
  - Core Java 8 for Beginners, by Sharanam Shah and Vaishali Shah, Shroff Publishers & Distributors PVT. LTD. First Edition, July 2015.
  - Java The Complete Reference, Ninth Edition, Herbert Schildt, McGraw-Hill Education, 2014.
- Websites:
  - https://www.javatpoint.com/java-tutorial
  - https://www.geeksforgeeks.org/java/
  - https://www.tutorialspoint.com/java/index.html

Google Classroom code for "Programming in Java"

- Join the Google Classroom by Using following Code:

x76ov4y

Thank You!!!Any Query?

asktoshivsir@gmail.com

Shivkumar Chandey

(+91 9987389441)

Scan QR Code to connect on LinkedIn