Java Technologies-I (Core Java)

Java Buzzwords

1. Simple
2. Object Oriented
3. Architecture Neutral
4. Portable
5. Robust
6. Multithreaded
7. Dynamic
8. Secure
9. High Performance
10. Distributed

Java is Simple Programming Language.

- Java language is derived from C and C++.
- Syntax of java is simpler than C/C++:
    1. No header files and no global definitions.
    2. No Copy constructor and operator overloading.
    3. No default arguments and constructor member initializer list.
    4. No delete operator and destructor.
    5. No friend function and class.
    6. No multiple implementation inheritance, Diamond problem and virtual base class.
    7. No private and protected mode of inheritance.
    8. No advanced typecasting operators.
    9. No pointers and pointer arithmetic.
- Size of software required to develop Java application is small.

Java is Object Oriented Programming Language.

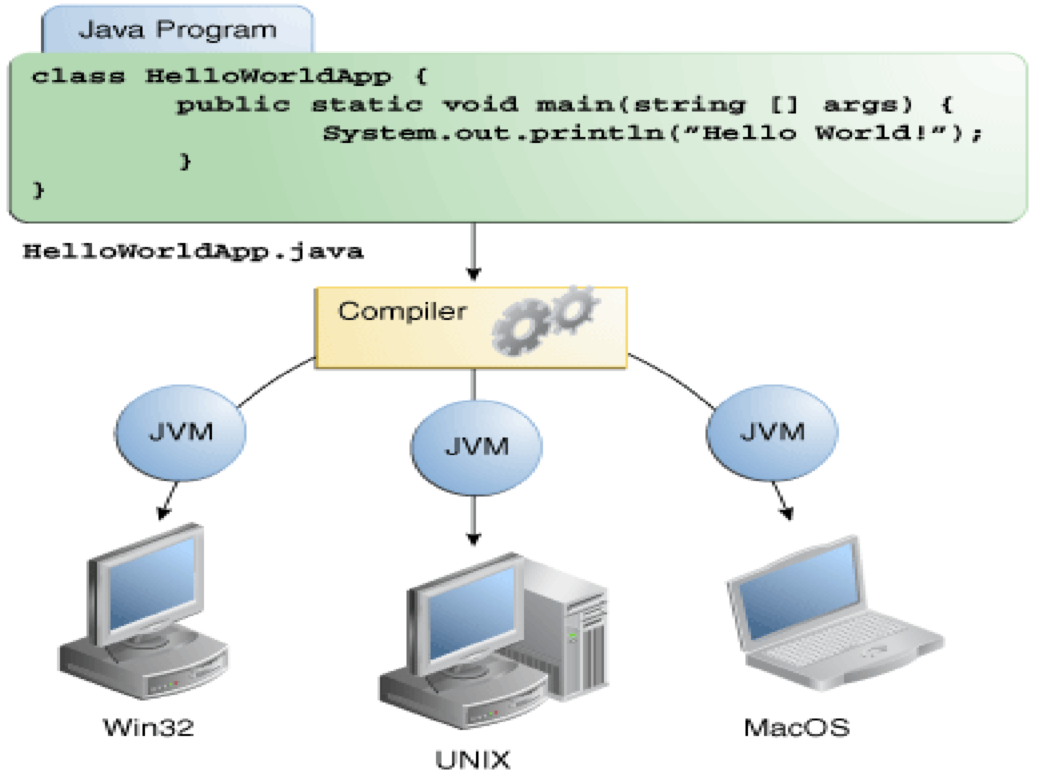
- According to Grady Booch, every object oriented programming language support to at least major pillars and some of the minor pillars of oops.
- Four major pillars of oops:
    1. Abstraction
    2. Encapsulation
    3. Modularity
    4. Hierarchy
- Three minor pillars of oops
    1. Typing / Polymosphism
    2. Concurrency
    3. Persistence.
- Java support to all major and minor pillars of oops.

Java is Architecture Neutral Prog. Language.

- Processor architecture's
  1. ARM
  2. Power PC
  3. X86 Intel's – IA32
  4. Alpha
  5. MIPS
  6. SPARC
- Java compiler do not generate native CPU code.
- Java compiler generates code for virtual machine that we can execute any where in presence of JVM.
- Bytecode makes java application architecture neutral.

Java is Portable Programming Language.

- Size of data types on all the platforms is constant.
- Since java is portable, it doesn't require sizeof operator.
- Since java is architecture neutral it is portable too.



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

Java's slogan is "**Write Once Run Anywhere**"

Java is Robust Programming Language.

1. It is architecture neutral
2. It is object oriented
3. It is having robust memory management
4. It’s Exception Handling

Java is Multithreaded Programming Language.

- When JVM starts execution of execution of application, it also starts execution of main thread and garbage collector(GC).
    1. Main thread is responsible for calling main method.
    2. Garbage collector(GC) / Finalizer is responsible for deallocating/releasing memory of unused objects.

- Thread is Non java resource/OS resource.

- To use OS thread, java application developer need not to do native coding. Java has given built in support to thread. Just import package and start using threads:
    1. java.lang.*;
    2. java.util.concurrent.*

Java is Dynamic Programming Language.

- Except constructor and static method all the methods are by default virtual.
- It was designed to adapt to an evolving environment.
- Libraries can freely add new methods and variable without affecting client.
- Reflection helps to get information of any type or instance anytime.

Java is Secure Programming Language.

- Java is intended to be used in networked/distributed environments. Toward that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems.
- From the beginning, Java was designed to make certain kinds of attacks impossible, among them:

    1. A common attack of worms and viruses
    2. Corrupting memory outside its own process space
    3. Reading or writing files without permission

Java is High Performance Programming Language.

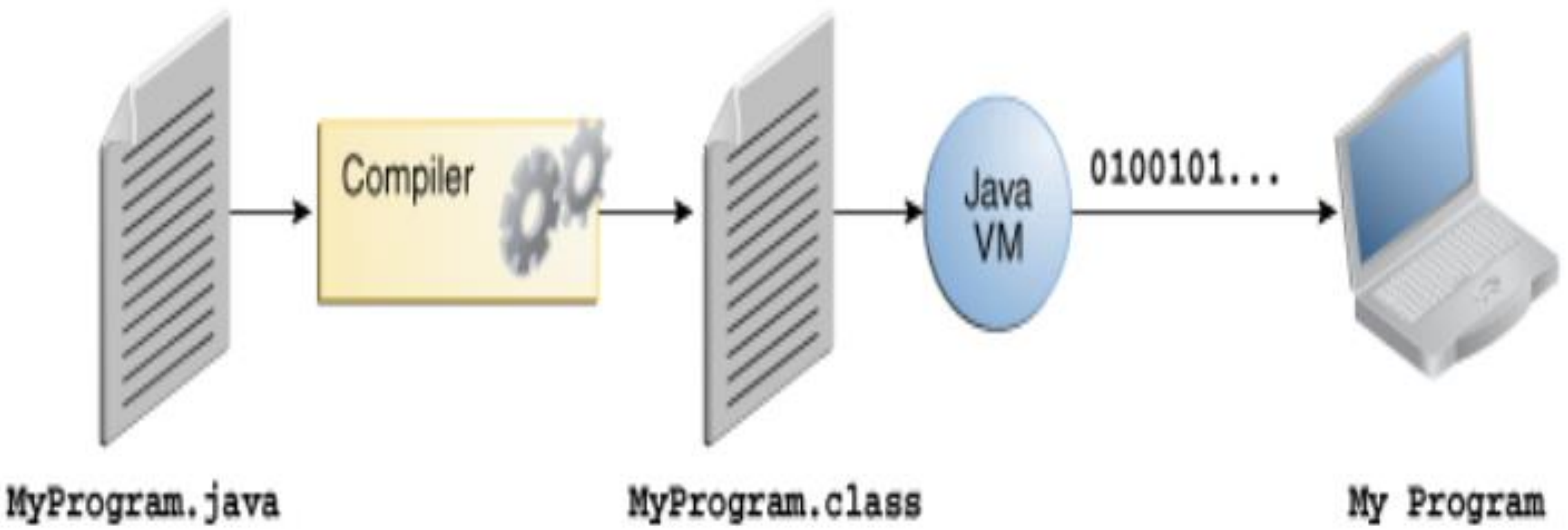
1. Dynamic class loading.
2. On the fly converting Bytecode into machine code.
3. Use of JIT to cache the machine code.

Java is Distributed Programming Language.

- Java has an extensive library of routines for coping with TCP/IP protocols like HTTP and FTP. Java applications can open and access objects across the Net via URLs with the same ease as when accessing a local file system.

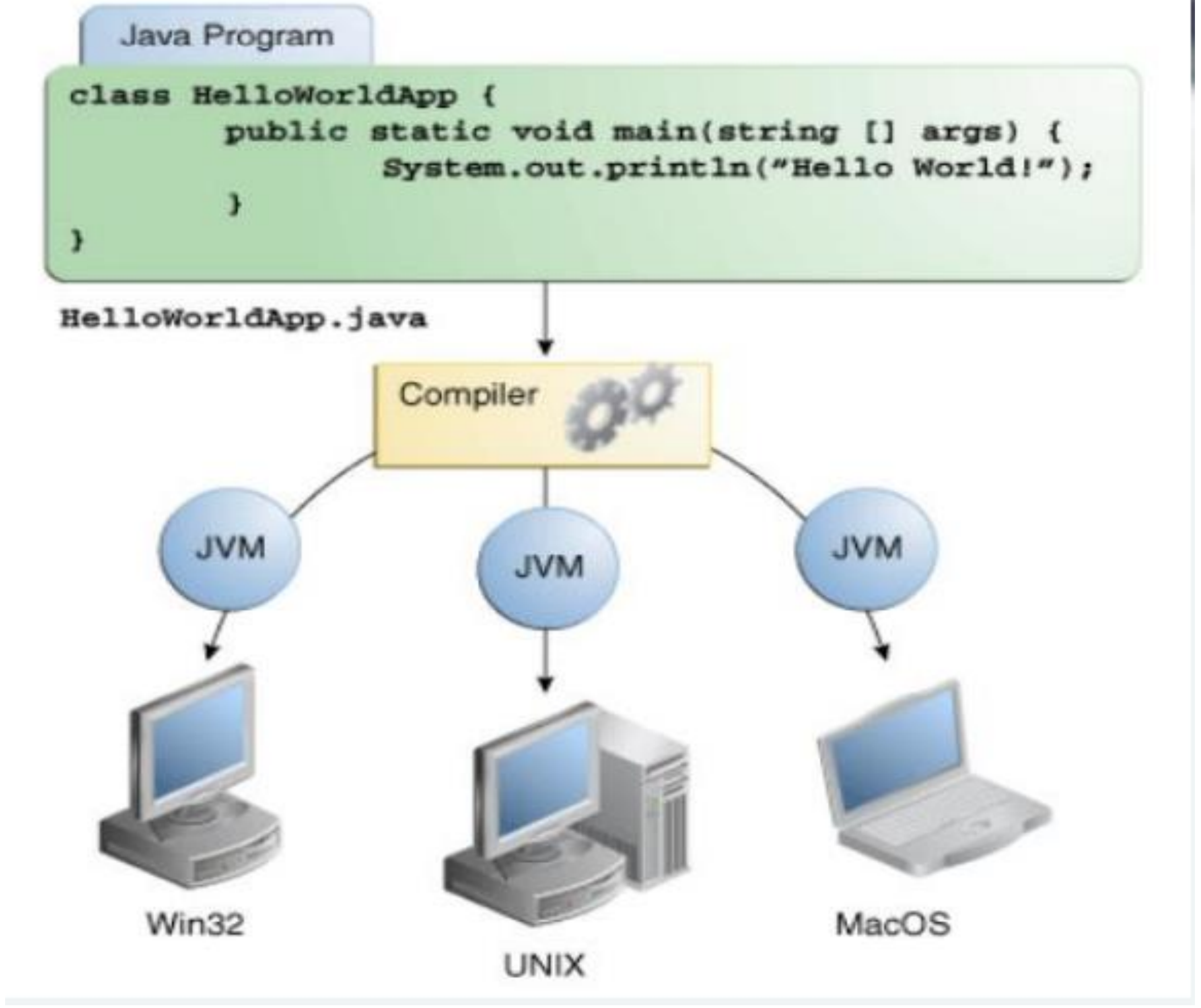
- Since Java supports RMI, it is distributed.

JAVA Compilation and Execution



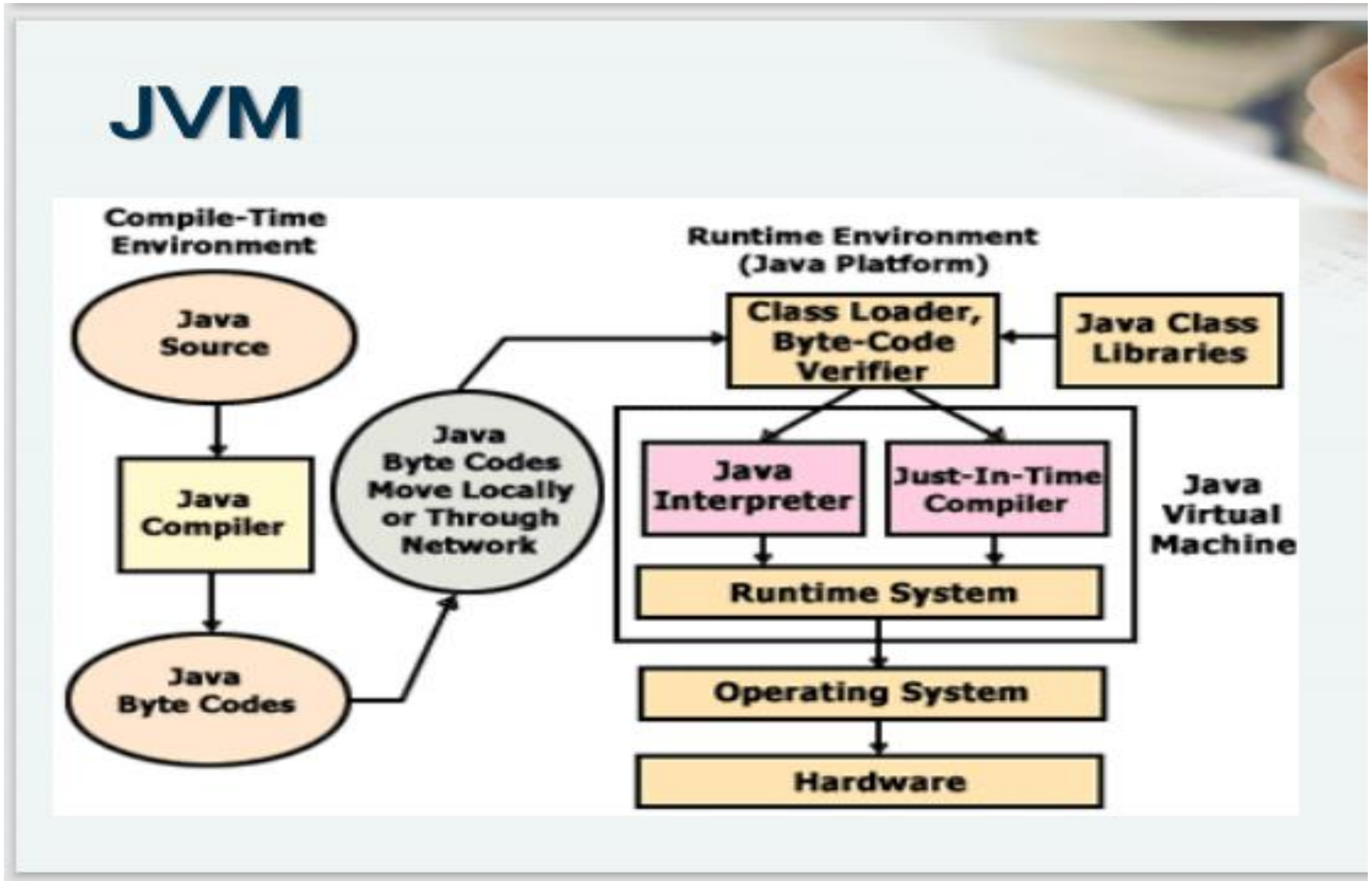
An overview of the software development process.

JAVA Compilation and Execution

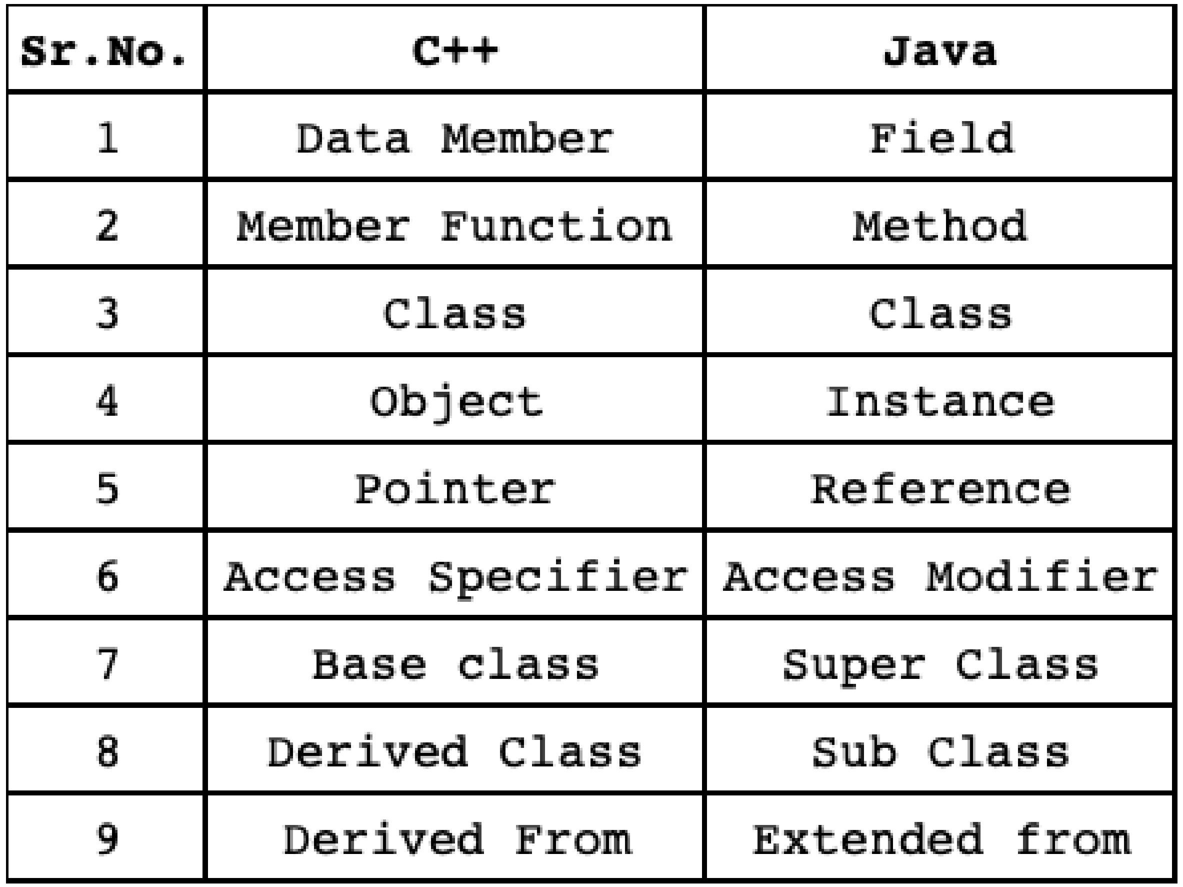


```
class HelloWorldApp {
        public static void main(string [] args) {
                System.out.println("Hello World!");
        }
}
```

Java Virtual Machine(JVM)



Some Java Terminologies



Sr.No.	C++	Java
1	Data Member	Field
2	Member Function	Method
3	Class	Class
4	Object	Instance
5	Pointer	Reference
6	Access Specifier	Access Modifier
7	Base class	Super Class
8	Derived Class	Sub Class
9	Derived From	Extended from

Path and Classpath

- path- path is operating system environment variable which is used to locate javac file
- Classpath- It is java platforms environment variable which is used to locate .class file

Simple Hello Application

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

Compile:
javac Program.java
Run:
java Program

Entry Point Method

- According to JVM specification, "**main**" is entry point method of Java application.
- Syntax:

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

- During execution of application JVM starts execution of 2 threads:
    1. Main thread
    2. Garbage Collector(GC). It is also called as finalizer.
- Main thread is responsible for calling main method.
- We can define main method per class but only one method can be considered as entry point method.
- We can overload main method in Java.

System.out.println

- System is a final class declared in java.lang package.
- Field's of System class:
  1. public static final InputStream in;
  2. public static final PrintStream out;
  3. public static final PrintStream err
- Following are overloaded methods of java.io.PrintStream class:
  1. public void print(String s);
  2. public PrintStream printf(String format, Object... args);
  3. public void println(String x)

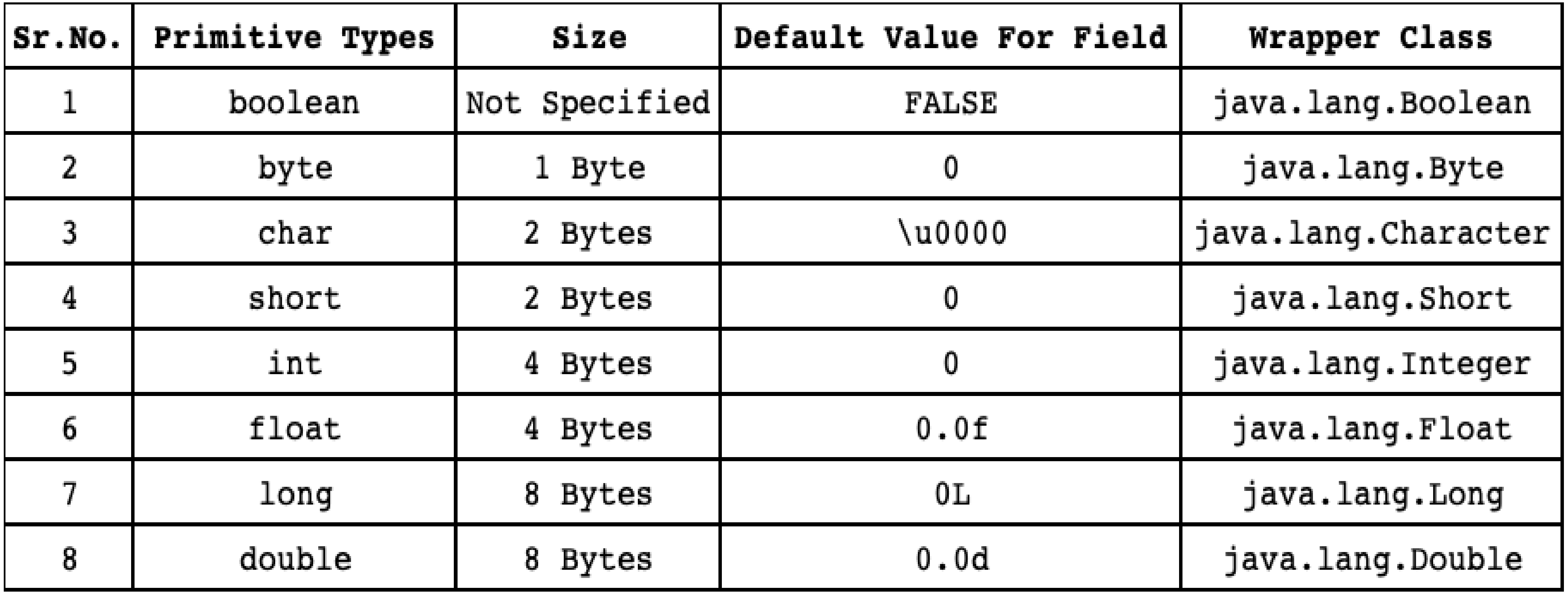
Data Types

- Types of data type in Java:
  1. Primitive Data Type (Also called as Value Type)
  2. Non Primitive Data Type (Also called as Reference Type)

- There are 8 primitive data types:
  - boolean, byte, char, short, int, long, float, double

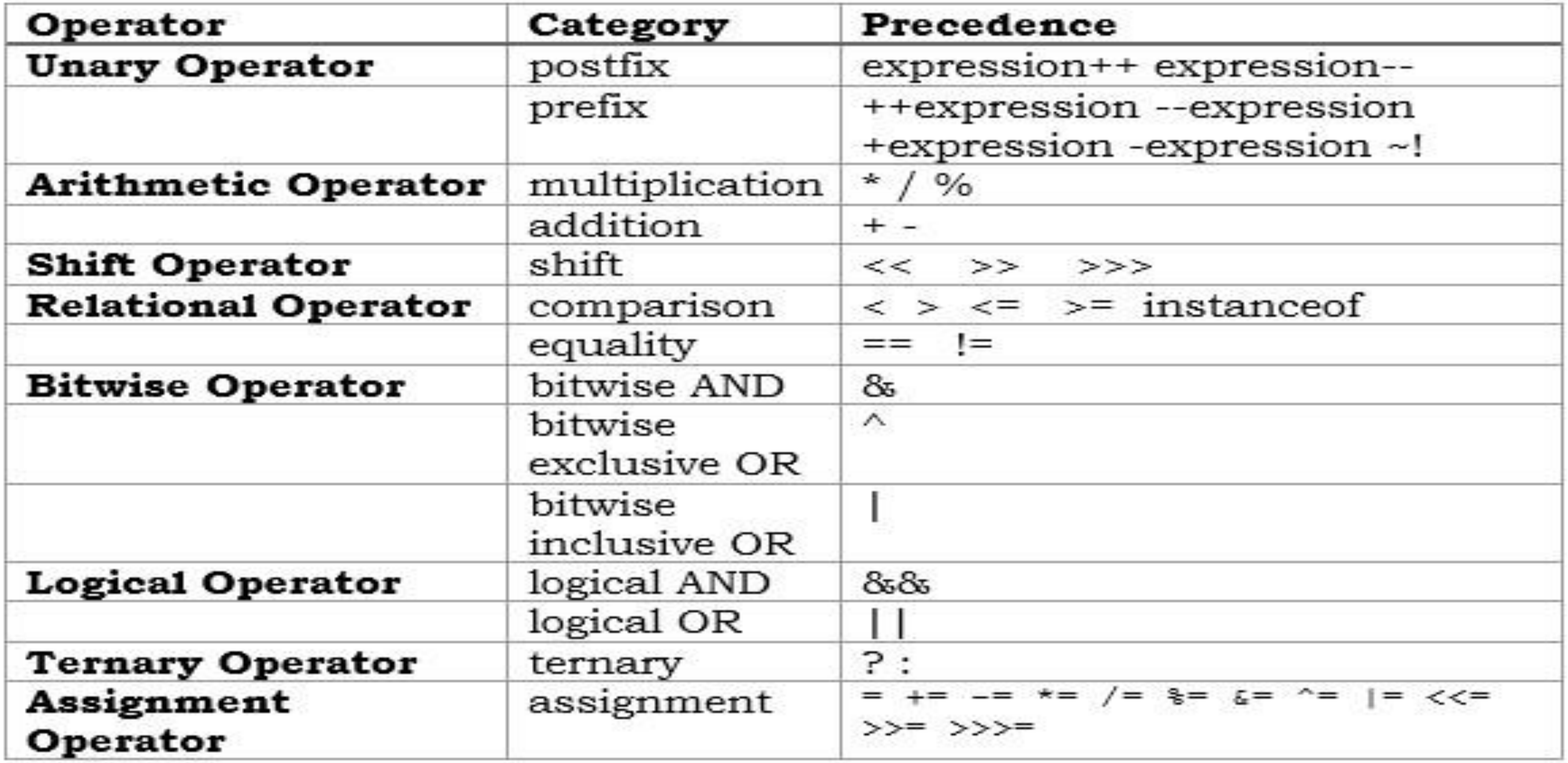
- There are 4 non primitive data types:
  - Interface, class, enum, array

Data Types



Sr.No.	Primitive Types	Size	Default Value For Field	Wrapper Class
1	boolean	Not Specified	FALSE	java.lang.Boolean
2	byte	1 Byte	0	java.lang.Byte
3	char	2 Bytes	\u0000	java.lang.Character
4	short	2 Bytes	0	java.lang.Short
5	int	4 Bytes	0	java.lang.Integer
6	float	4 Bytes	0.0f	java.lang.Float
7	long	8 Bytes	0L	java.lang.Long
8	double	8 Bytes	0.0d	java.lang.Double

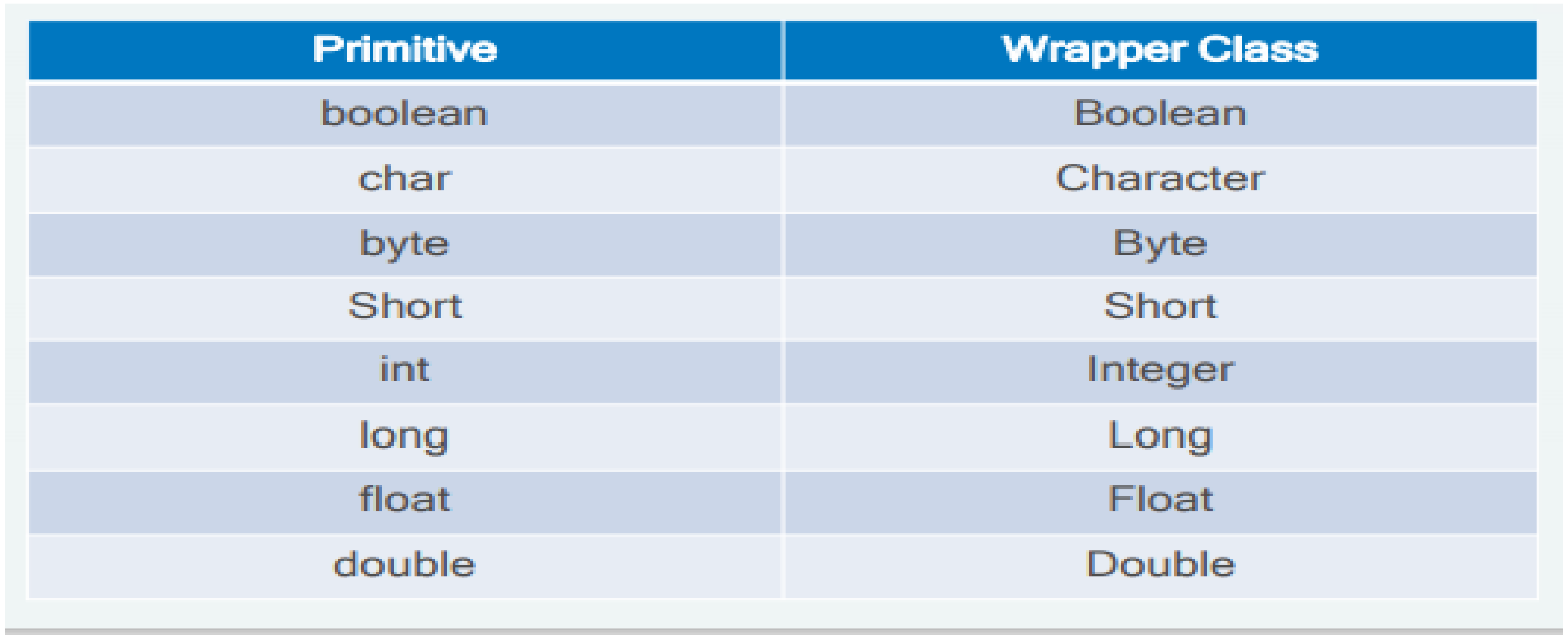
Operators



Operator	Category	Precedence
Unary Operator	postfix	expression++ expression--
	prefix	++expression --expression +expression -expression ~!
Arithmetic Operator	multiplication	* / %
	addition	+ -
Shift Operator	shift	<< >> >>>
Relational Operator	comparison	< > <= >= instanceof
	equality	== !=
Bitwise Operator	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical Operator	logical AND	&&
	logical OR	||
Ternary Operator	ternary	? :
Assignment Operator	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

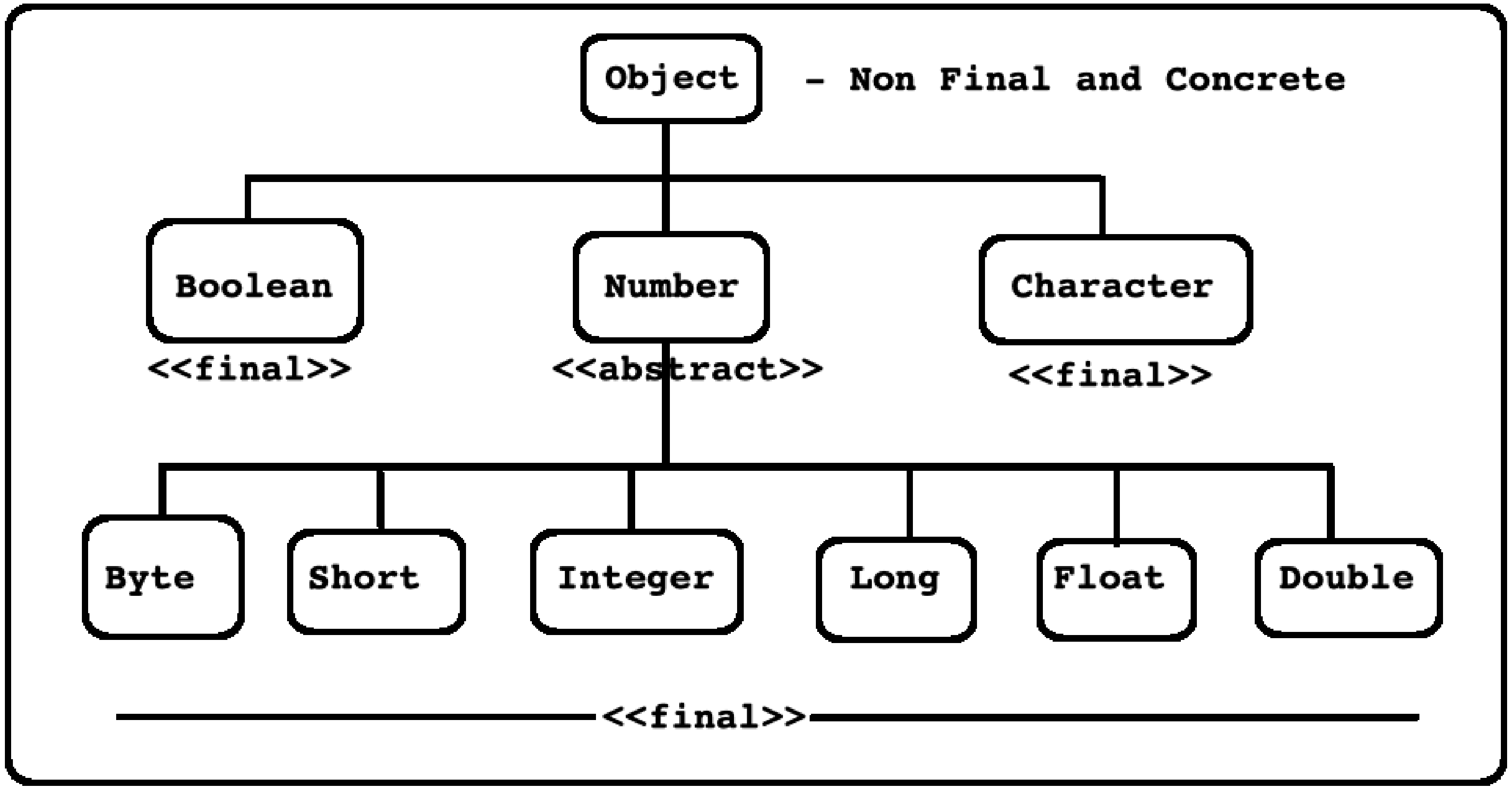
Wrapper Classes

- If we want to convert primitive data type into object type then we have to take help of wrapper classes



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

Wrapper Class



Boxing and Unboxing

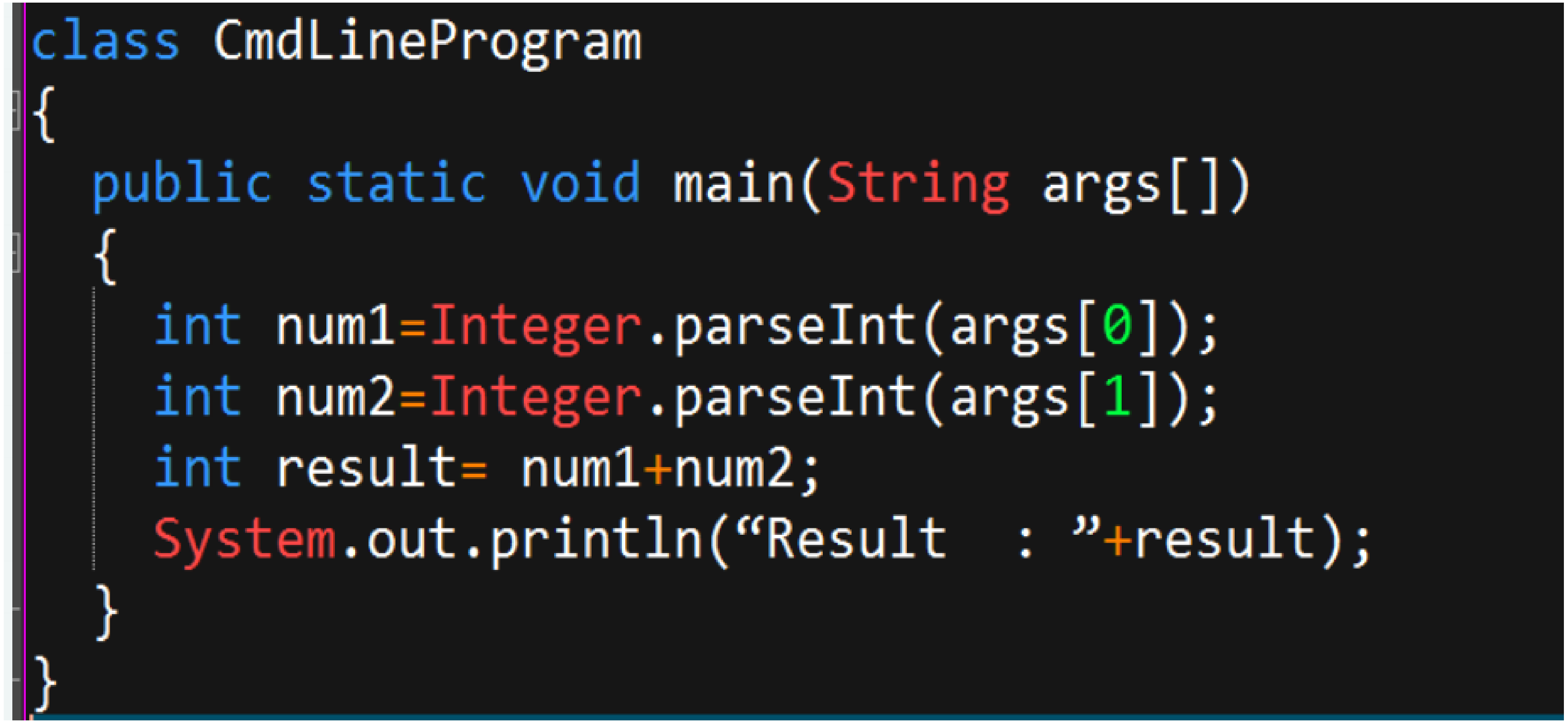
- Boxing – it is process of converting value of primitive data type into object data type.
- For Example: BoxingExample1.java
- Unboxing: It is process of converting object data type into primitive data type.
- For Example: UnboxingExample1.java
- The technique let us use primitive types and Wrapper class objects interchangeably and we do not need to perform any typecasting explicitly.

Command Line argument

- The arguments passed from command line are called command line arguments.
- These arguments are handled by main() function.
- 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 behaviour of the program for the different values.

Command Line argument

For Example: CmdlineargDemo1.java



```
class CmdLineProgram
{
  public static void main(String args[])
  {
    int num1=Integer.parseInt(args[0]);
    int num2=Integer.parseInt(args[1]);
    int result= num1+num2;
    System.out.println(“Result  : ”+result);
  }
}
```

How to take input using Scanner Class

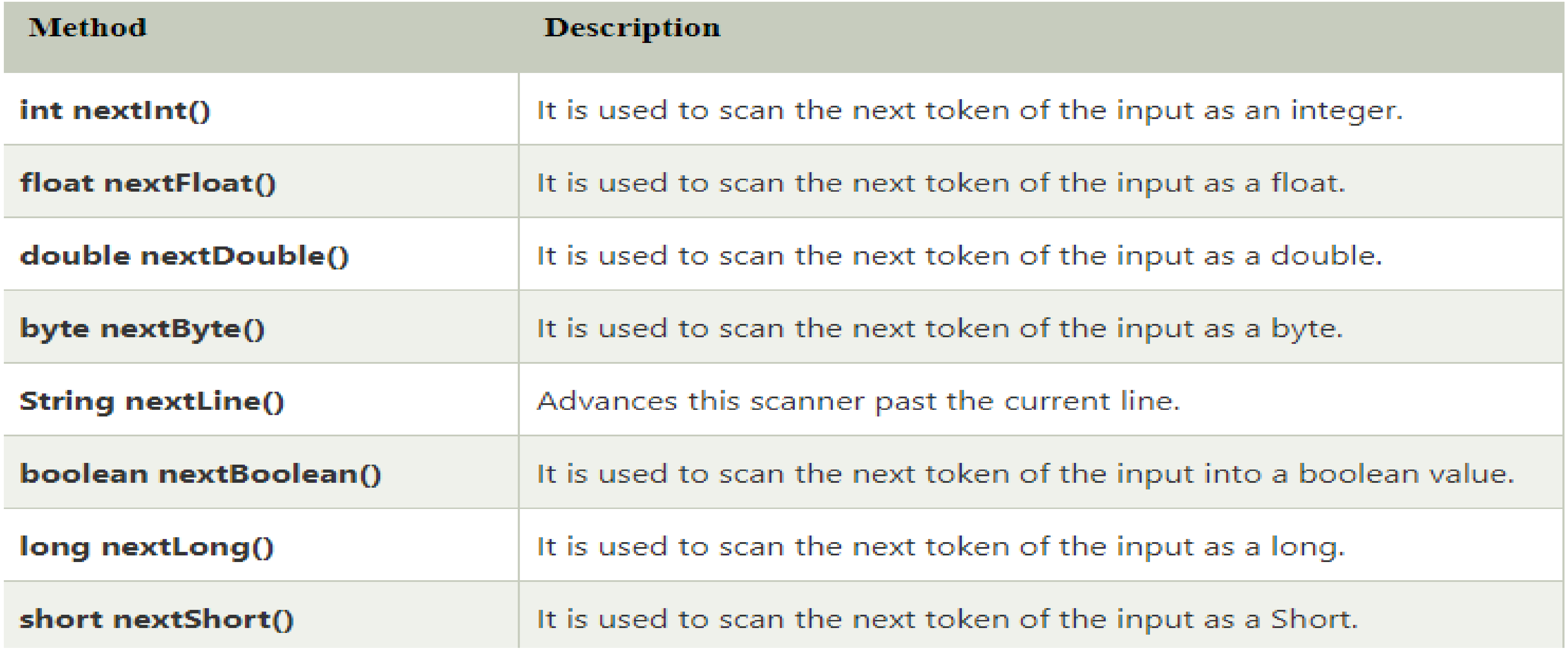
- **Scanner Class**
- Java **Scanner class** allows the user to take input from the console.
- It belongs to **java.util** package. It is used to read the input of primitive types like int, double, long, short, float, and byte.
- It is the easiest way to read input in Java program.

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

- The above statement creates a constructor of the Scanner class having **System.in** as an argument.
- It means it is going to read from the standard input stream of the program. The **java.util** package should be import while using Scanner class.

Methods of Java Scanner Class

For Example: Inputdemo.java



Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.

Type Casting in Java

- In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically.
- The automatic conversion is done by the compiler and manual conversion performed by the programmer.
- Types

1. Widening Type Casting
2. Narrowing Type Casting

Widening Type Casting

- Converting a lower data type into a higher one is called **widening** type casting.
- It is also known as **implicit conversion** or **casting down**.
- It is done automatically. It is safe because there is no chance to lose data.
- It takes place when:
  - Both data types must be compatible with each other.
  - The target type must be larger than the source type.

  byte -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**
- For Example: WideningTypeCastingExample.java

Narrowing Type Casting

- Converting a higher data type into a lower one is called **narrowing** type casting.
- It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer.
- If we do not perform casting then the compiler reports a compile-time error.

double -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

- For Example:
- NarrowingTypeCastingExample.java

Variables

- A variable is a container which holds the value while the Java program is executed.
- A variable is assigned with a data type.
- Variable is a name of memory location.
- There are three types of variables in java: local, instance and static.
- There are two types of data types in Java: primitive and non-primitive.
- A variable is the name of a reserved area allocated in memory.
- In other words, it is a name of the memory location.

Variables

1) Local Variable

- A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

2) Instance Variable

- A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.
- It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

- A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.
- Example: VariableDemo1.java

Access Modifiers

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.
- There are four types of Java access modifiers:

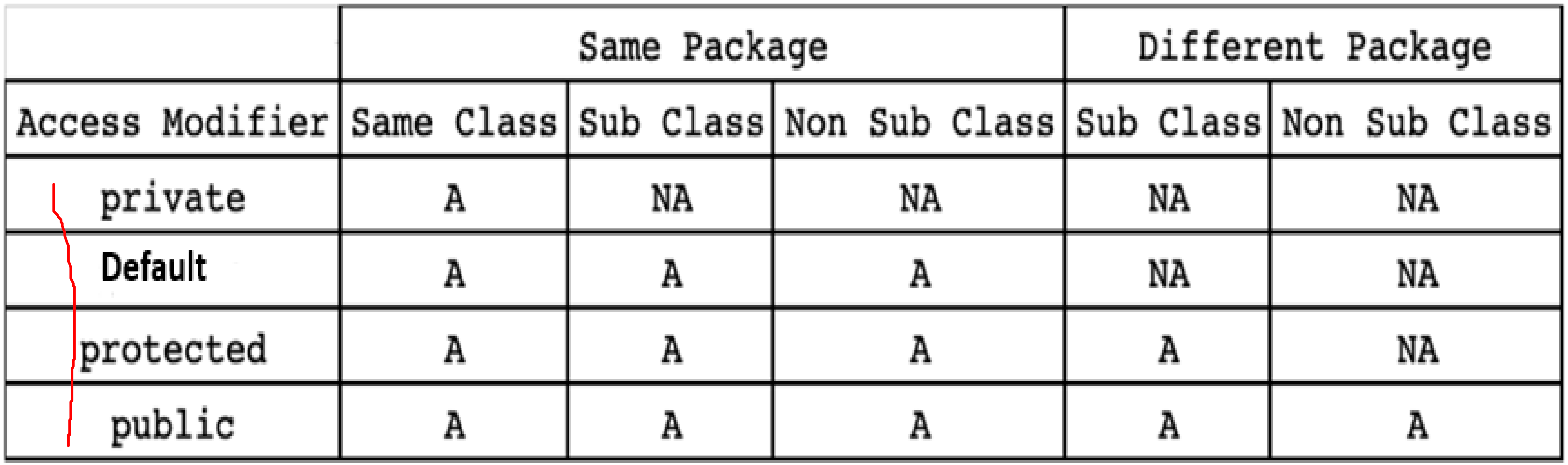
1.Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class. For Example: AccessModiDemo.java

2.Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3.Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

4.Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifiers



	Same Package			Different Package	
Access Modifier	Same Class	Sub Class	Non Sub Class	Sub Class	Non Sub Class
private	A	NA	NA	NA	NA
Default	A	A	A	NA	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

A : Accessible NA : Not Accessible

Constructor

- 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.
- Constructor name must match class name, and it cannot have a return type(like void)

Types of Constructors

1. Default Constructor :

- Java automatically creates default constructor if there is no parameterless or parameterized constructor written by user.
- Initializes member data variable to default values (numeric values are initialized as 0, booleans are initialized as false and references are initialized as null).
- For example: ConstructorDemo.java

Types of Constructors

2. Parameterless Constructor :

- A constructor that takes no parameters is called a parameterless constructor.
- Parameterless constructors are invoked whenever an object is instantiated by using the new operator and no arguments are provided to new .
- For example: NoarguConstructorDemo.java

Types of Constructors

3. Parameterized Constructor :

- A constructor that has parameters is known as parameterized constructor.
- If we want to initialize fields of the class with your own values, then use a parameterized constructor.
- For example: ArguConstructorDemo.java

Array

- An array is a collection of similar type(homogenous) of elements which has contiguous memory location.
- In java **arrays are object**, which contains elements of a similar data type.
- The elements of an array are stored in a contiguous memory location.
- It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
- Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

Array

- In Java, array is an object of a dynamically generated class.
- Java array inherits the **Object class**, and implements the Serializable as well as Cloneable interfaces.
- We can store primitive values or objects in an array in Java.
- Types of array

1. Single Dimensional
2. Multi-Dimensional
3. Jagged/Ragged array

Array

1.Single Dimensional:

- The general form of a single dimensional array declaration is

```
datatype[] arrayName;
Or
datatype arrayName[];
Or
datatype []arrayName;
```

```
int[] num;
Or
int num[];
Or
int []num;
```

•**datatype** can be a **primitive data type**(int, char, Double, byte etc.)or **Non-primitive data** (Objects).

•**arrayName** is the name of an array

Array

1.Single Dimensional:

- For the creation of an array **new** keyword is used with a data type of array.
- You must specify the size of the array. The size should be an integer value or a variable or expression that contains an integer value. How many elements you can store an array directly depends upon the size of an array.

```
arrayName = new DataType[size];
```

```
num = new int[10];
```

- new Datatype[size]: It creates an array in heap memory. Because an array is an object, so it stored in heap memory.
- arrayName: Assignment operator assigns the newly created array to the reference of variable arrayName.
- We can access the elements of an array by use of array names.

Array

1.Single Dimensional:

- To initialize the Array we have to put the values at each index of array.
- For example: num[0]=10;

  num[1]=20; and so on.
- You can also construct an array by initializing the array at declaration time.

```
dataType arrayName[] = {value1, value2, …valueN}

int number[] = {11, 22, 33 44, 55, 66, 77,
                88, 99, 100}
```

- For Example: OndDArrayDemo.java

Array

2. Multi-Dimensional A Multi-dimensional array is the array of arrays because they store in tabular form.

- As like single dimensional array we must declare the two-dimensional array variable. We must specify the array name followed by two square brackets called **subscript([][])**.
- Declaration:

```
datatype[][] arrayName;
          Or
datatype arrayName[][];
          Or
datatype [][]arrayName;
```

```
int[][] num;
     Or
int num[][];
     Or
int [][]num;
```

Array

2. Multi-Dimensional

- Creation/construction:
- creation of a Multi-dimensional array, the new keyword is used with the data type of array.
- There are two subscripts for the size of an array. The first subscript is used to define the number of rows in the matrix and the second one is for the number of columns.
- You must specify the size of the array because how many number elements you can store an array directly depends upon the size of the array.

```
arrayName = new DataType[size][size];
```

```
num = new int[3][3];
```

Array

2. Multi-Dimensional

- Initialization:

```
dataType arrayName[][] = {
{value1, value2, …valueN},
{value1, value2, …valueN},
{value1, value2, …valueN},
};
```

```
int[][] num = {
{11, 22, 33},
{44, 55, 66},
{77, 88, 99},
};
```

- For Example: ExampleOfMultiDimentionalArray.java

Array

3. Jagged array

- A Jagged array is also called as Ragged Array. A jagged array is an array of arrays.
- Member arrays can be of different sizes, i.e. we can create a 2-D array but with a variable number of columns in each row.

```
data_type array_name[][] = new data_type[n][]; //n: no. of rows
array_name[] = new data_type[n1] //n1= no. of colmuns in row-1
array_name[] = new data_type[n2] //n2= no. of colmuns in row-2
array_name[] = new data_type[n3] //n3= no. of colmuns in row-3
.......
........
........
array_name[] = new data_type[nk] //nk=no. of colmuns in row-n
```

Array

3. Jagged array

- Initialization:

```
int arr_name[][] = new int[][]  {
                    new int[] {10, 20, 30 ,40},
                    new int[] {50, 60, 70, 80, 90, 100},
                    new int[] {110, 120}
                      };
```

OR

```
int[][] arr_name = { new int[] {10, 20, 30 ,40},
                     new int[] {50, 60, 70, 80, 90, 100},
                     new int[] {110, 120} };
```

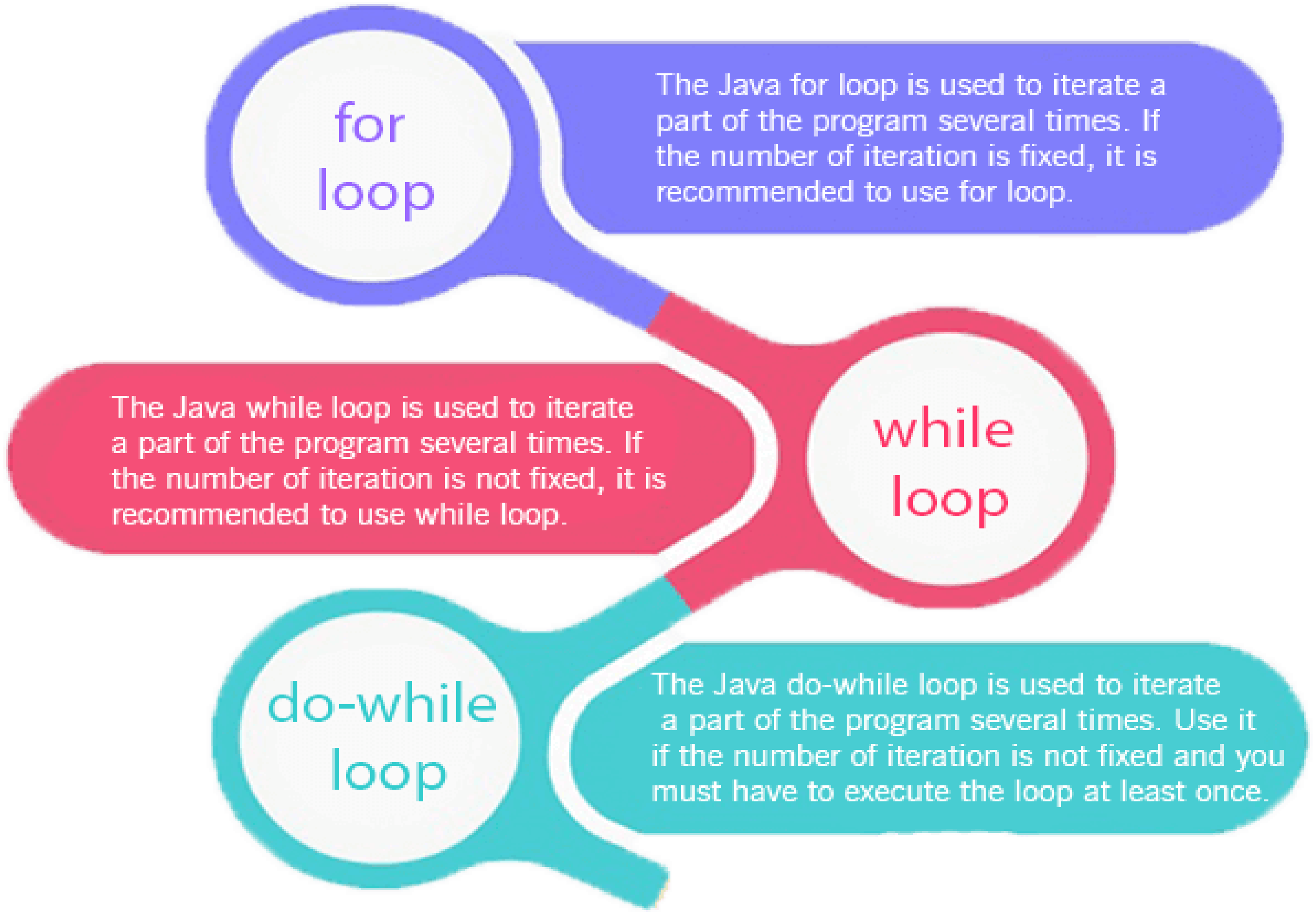
```
int[][] arr_name = { {10, 20, 30 ,40},
                     {50, 60, 70, 80, 90, 100},
                     {110, 120} };
```

- For example: JaggedArrayDemo.java

Types of Loop

- 4 Types of loop

1. for
2. for-each
3. while
4. do-while


for loop
The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
while loop
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.
do-while loop
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Types of Loop

- for each loop:

- The for-each loop is used to traverse array or collection in Java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

- It works on the basis of elements and not the index. It returns element one by one in the defined variable.

```
for(data_type variable : array_name)
{
//code to be executed
}
```

```
int arr[]={12,23,44,56,78};
   //Printing array using for-each loop
   for(int i:arr){
      System.out.println(i);
   }
```

String Class

- String is reference type not a primitive type
- String class belongs to java.lang package.
- String class is final class.
- string is basically an object that represents sequence of char values. An array of characters works same as Java string.

```
char[] ch={‘h','’e',’l',’l’,’o'};
String s=new String(ch);
Is same as
String s=“hello";
```

String Class

- We can create object of string in two ways:

```
1. String s1 = new String(“Hello”); //heap

2. String s2 = “Hello”; //String literal pool
```

- Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

String Class

- How to create a string object?
- There are two ways to create String object:

A. By string literal

B. By new keyword

String Class

String Literal:

- Java String literal is created by using double quotes.
- For Example: String s="welcome";
- Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.
- For example:

```
1.String s1="Welcome";
2.String s2="Welcome";//It doesn't create a new instance
```

- In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object.
- After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

String Class

By new Keyword:

1.String s=**new** String("Welcome");

- In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Methods for String Class

- **charAt(int index)=**method returns *a char value at the given index number*.
- **compareTo(String str)**
- **compareToIgnoreCase(String str)**
- **concat(String str)=**method *combines specified string at the end of this string*. It returns a combined string. It is like appending another string.
- **contains(CharSeqiences)=**method searches the sequence of characters in this string. It returns *true* if the sequence of char values is found in this string otherwise returns *false*.
- **length()=**method length of the string. It returns count of total number of characters.
- **toString()=**to represent any object as a string

String Compare

- We can compare String in Java on the basis of content and reference.
- There are three ways to compare String in Java:

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

String Compare

- By Using equals() Method
- The String class equals() method compares the original content of the string.
- It compares values of string for equality. String class provides the following two methods:
- **public boolean equals(Object another)** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)** compares this string to another string, ignoring case.
- For Example: Stringcomparison1.java

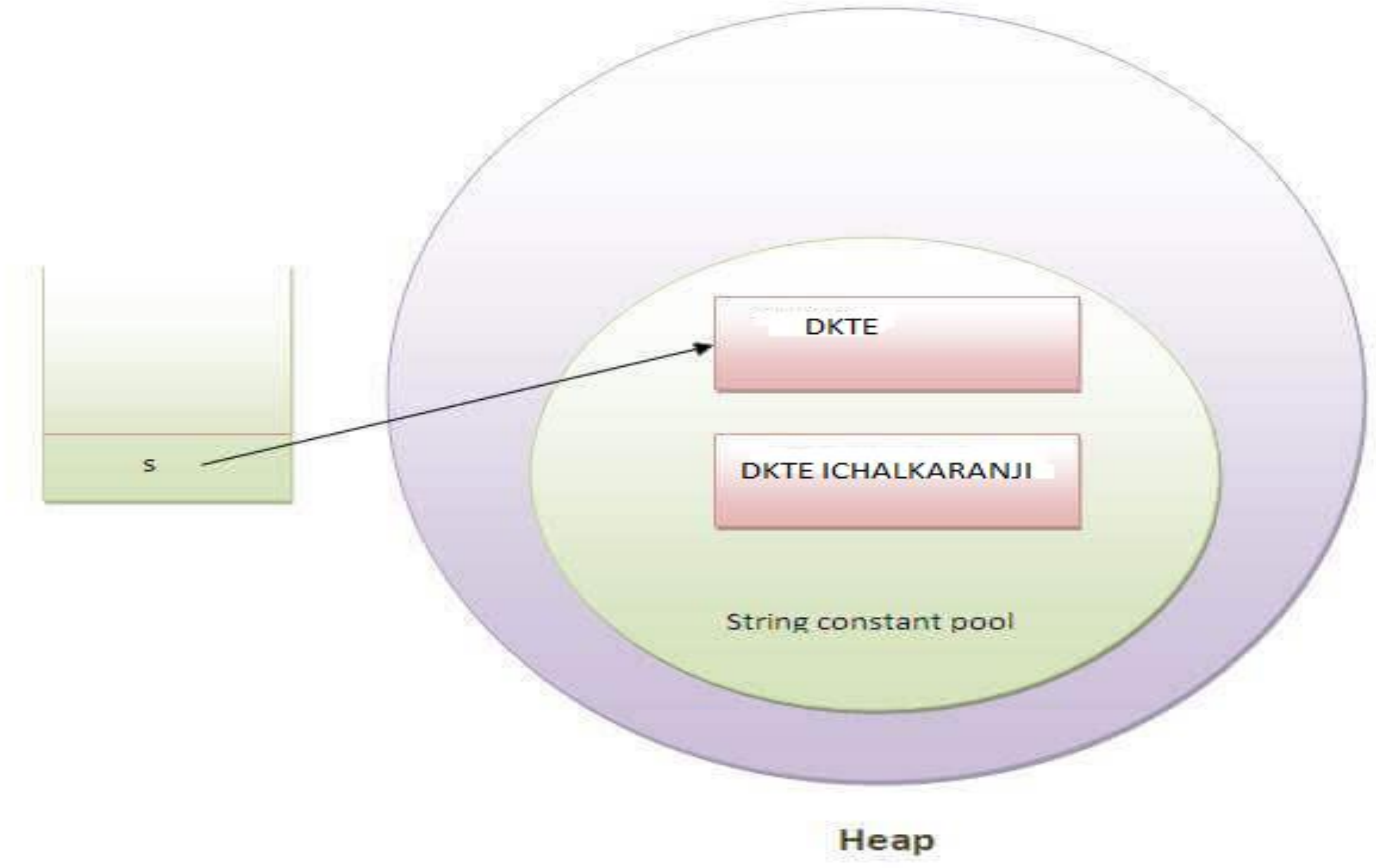
toString() Method

- toString() method
- If you want to represent any object as a string, **toString() method is used.**
- The toString() method returns the string representation of the object.
- If you print any object, java compiler internally invokes the toString() method on the object.
- So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

String are Immutable

- Immutable simply means unmodifiable or unchangeable.
- Once string object is created its data or state can't be changed but a new string object is created.
- For Example:Testimmutablestring.java

String are Immutable



StringBuffer class

- Java StringBuffer class is used to create mutable (modifiable) String objects.
- The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.
- It belongs to java.lang package
- It is final class
- For Example: StringBufferExample.java

Nested Classes and Inner Classes

- In Java, it is possible to define a class within another class, such classes are known as nested classes.
- They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and creates more readable and maintainable code.
- Nested classes are divided into two categories:

1. static nested class : Nested classes that are declared static are called static nested classes.
2. inner class : An inner class is a non-static nested class.

- Syntax:

```
class OuterClass
{ ...
        class NestedClass
        { ...
        }
}
```

Nested Classes and Inner Classes

- The scope of a nested class is bounded by the scope of its enclosing class. Thus in above example, class *NestedClass* does not exist independently of class *OuterClass*.
- A nested class has access to the members, including private members, of the class in which it is nested. However, the reverse is not true i.e., the enclosing class does not have access to the members of the nested class.
- A nested class is also a member of its enclosing class.
- As a member of its enclosing class, a nested class can be declared *private*, *public*, *protected*, or *package private*(default).
- Nested classes are divided into two categories:
  - **static nested class :** Nested classes that are declared *static* are called static nested classes.
  - **inner class :** An inner class is a non-static nested class.

Nested Classes and Inner Classes

- Need of Java Inner class
- Sometimes users need to program a class in such a way so that no other class can access it. Therefore, it would be better if you include it within other classes.
- For Example:

1. StaticNestedClassDemo.java
2. InnerClassDemo.java

Inheritance

- Inheritance is an important pillar of OOP(Object-Oriented Programming).
- It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
- 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.
- classes can be derived from other classes, thereby inheriting fields and methods from those classes.
- It is Process or journey from generalization to Specialization.

Inheritance

- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.
- It is also called as “kind-of ” relationship.
- Why use inheritance in java?
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Inheritance

- 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.

Inheritance

The syntax of Java Inheritance

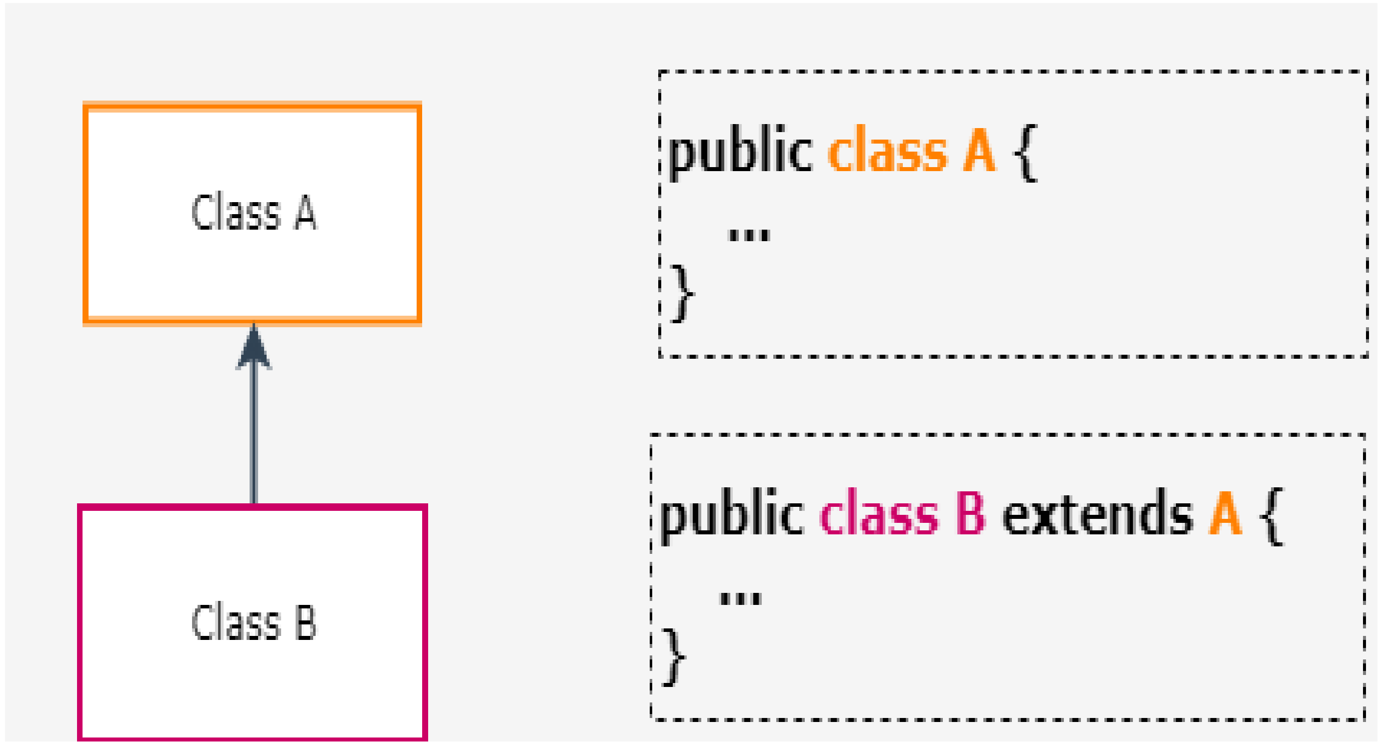
```
Class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

- 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

1. A class that is derived from another class is called a subclass.
2. The class from which the sub class is derived is called a super class.(also a base class or a parent class).

Example:



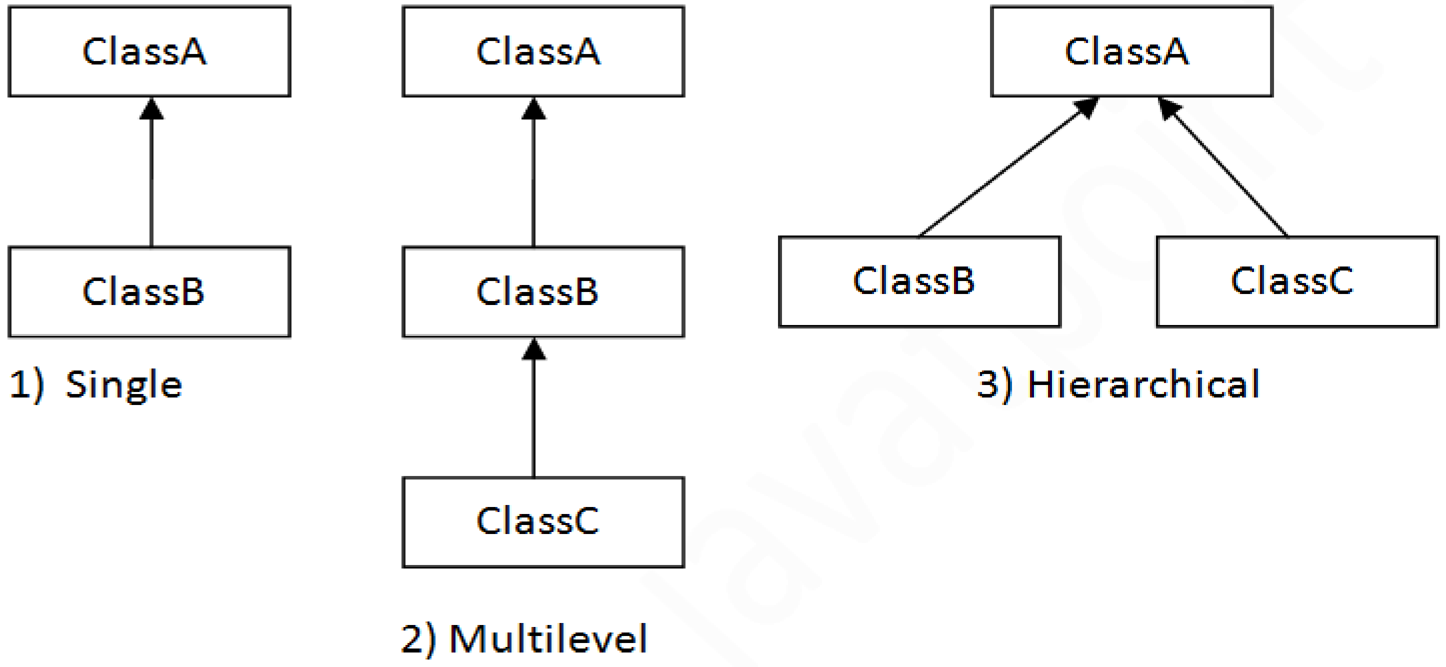
```
public class A {
    ...
}
```

```
public class B extends A {
    ...
}
```

Types of Inheritance

- On the basis of class, there can be three types of inheritance in java:

1. Single,
2. Multilevel and
3. Hierarchical.



- In java programming, multiple and hybrid inheritance is supported through interface only.

Types of Inheritance

1. Single Inheritance

- When a class inherits another class, it is known as a single inheritance. only one child is derived from the parent class.
- SingleInherDemo.java

2. Multilevel Inheritance

- It means that one child class is derived from one parent class. Now, this child class act as a parent class and another child class is derived from it. This process goes on.
- MultilevelInherDemo.java

3. Hierarchical Inheritance

- In hierarchical Inheritance, more than one child class can be derived from the base class. All the child classes can inherit similar properties from the parent class.
- HierarchicalDemo.java

this keyword

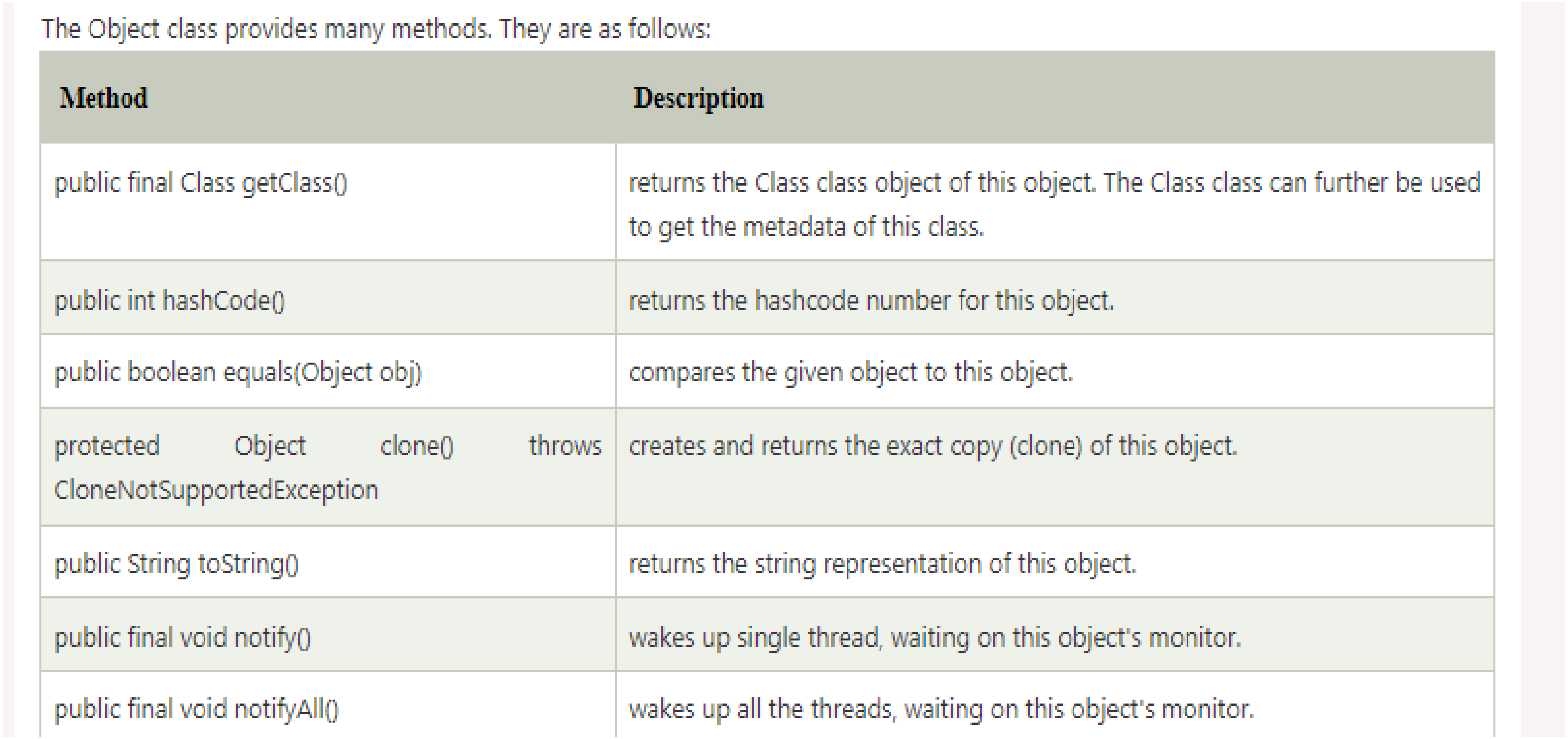
- Private members of a class can't be inherited by another class. For Example: SingleInherDemo.java
- this keyword :
- **this** is a reserved keyword in java i.e., we can't use it as an identifier.
- **this** is used to refer **current-class's instance as well as static members.**
- **this** can be used in various contexts as given below:

1. to refer instance variable of current class. For Example: TestThis1.java
2. to invoke or initiate current class constructor. For Example: TestThis2.java
3. can be passed as an argument in the method call. For Example: S2.java
4. can be passed as argument in the constructor call. For Example: A4.java
5. can be used to return the current class instance. For Example: Test1.java
6. If we want to call any constructor from any other constructor within same class. It must be first statement inside constructor. For Example: TestThis3.java

Object Class

- The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java. i.e. In java, all the classes are by default extended from java.lang.Object class.
- It is also called Super cosmic base class or Universal Base class.
- The Object class is beneficial if you want to refer any object whose type you don't know.
- For Example : there is getObject() method that returns an object but it can be of any type like Employee, Student etc, we can use Object class reference to refer that object.
- For example:
- Object obj=getObject(); //we don't know what object will be returned from this method

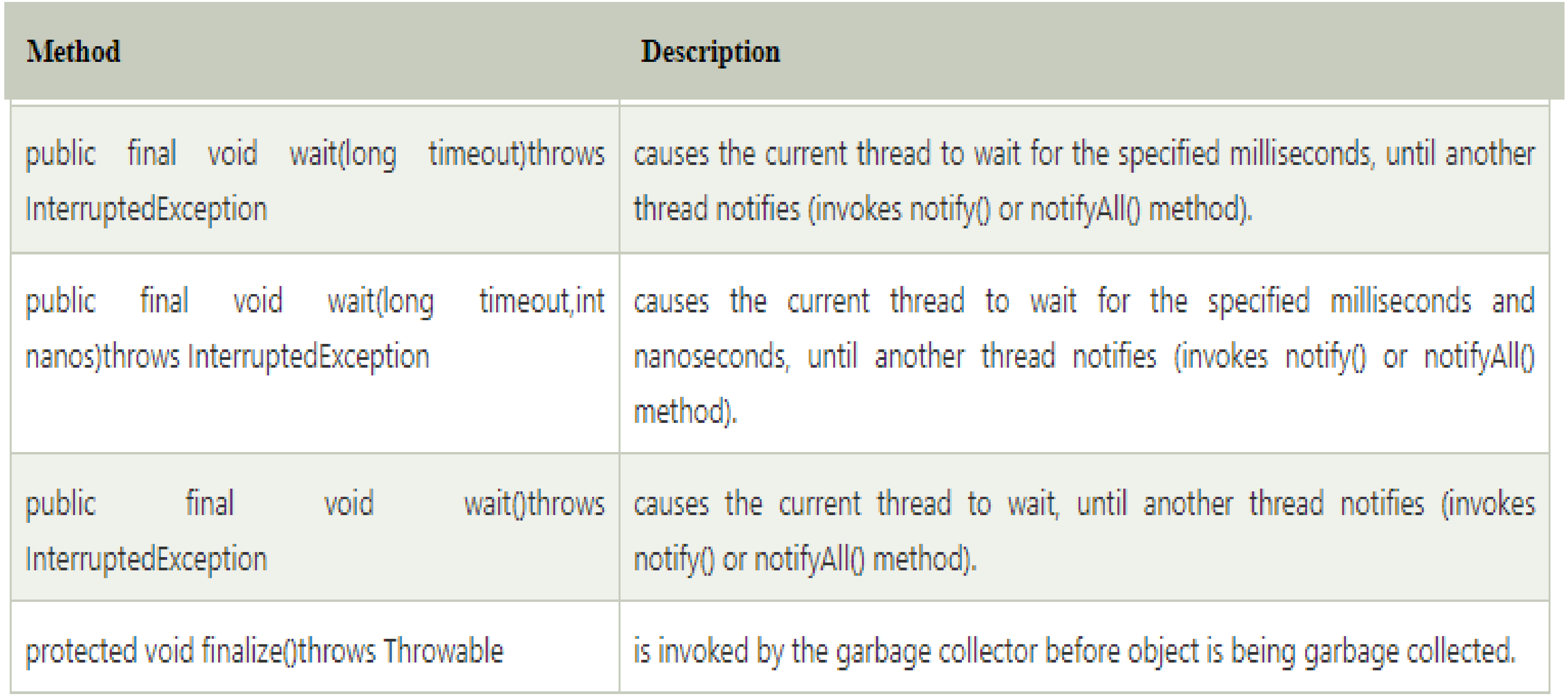
Methods of Object Class



The Object class provides many methods. They are as follows:

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.

Methods of Object Class



Method	Description
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Super Keyword

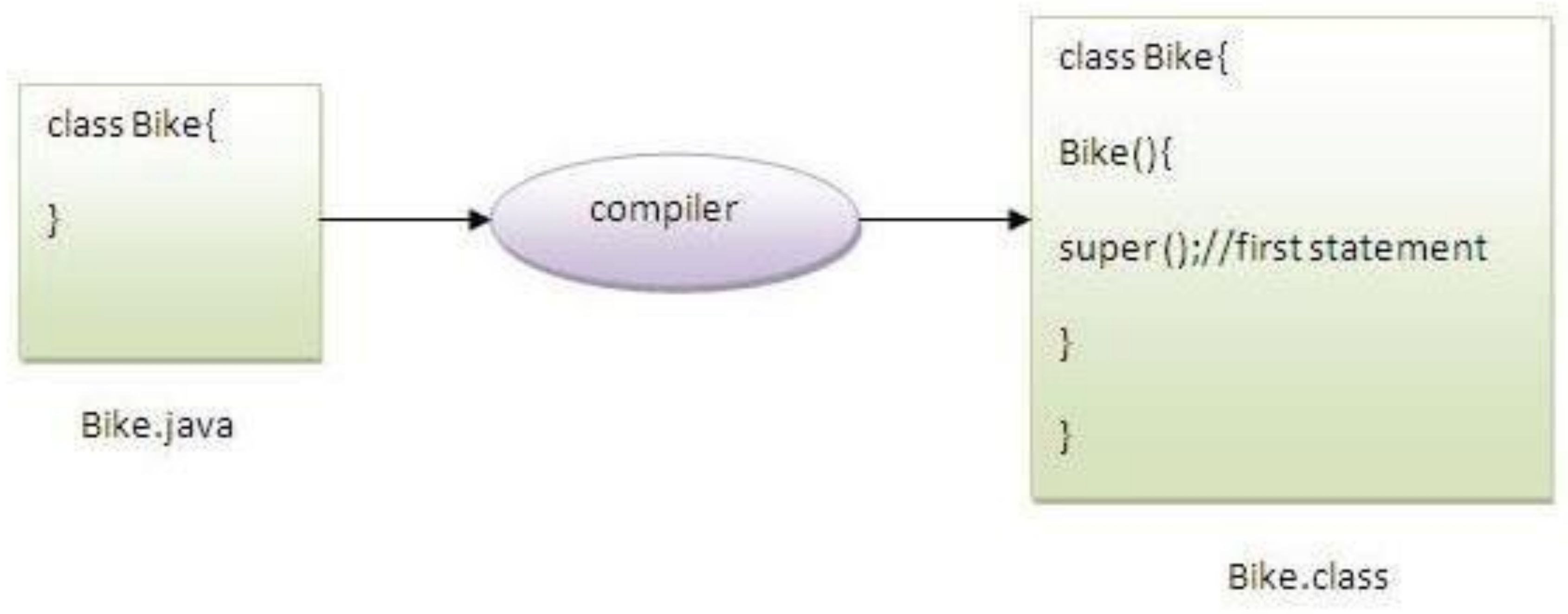
- The super keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
- Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable. TestSuper1.java
2. super can be used to invoke immediate parent class method. TestSuper2.java
3. super() can be used to invoke immediate parent class constructor. TestSuper2.java

- Super statement must be first statement inside the constructor.
- We can call super class members inside the subclass method by using super keyword.

Super Keyword

- super() is added in each class constructor automatically by compiler if there is no super() or this().



```
class Bike{

}
```

```
class Bike{

Bike(){

super();//first statement

}

}
```

Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.
- Usage of Java Method Overriding

1. Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
2. Method overriding is used for runtime polymorphism.

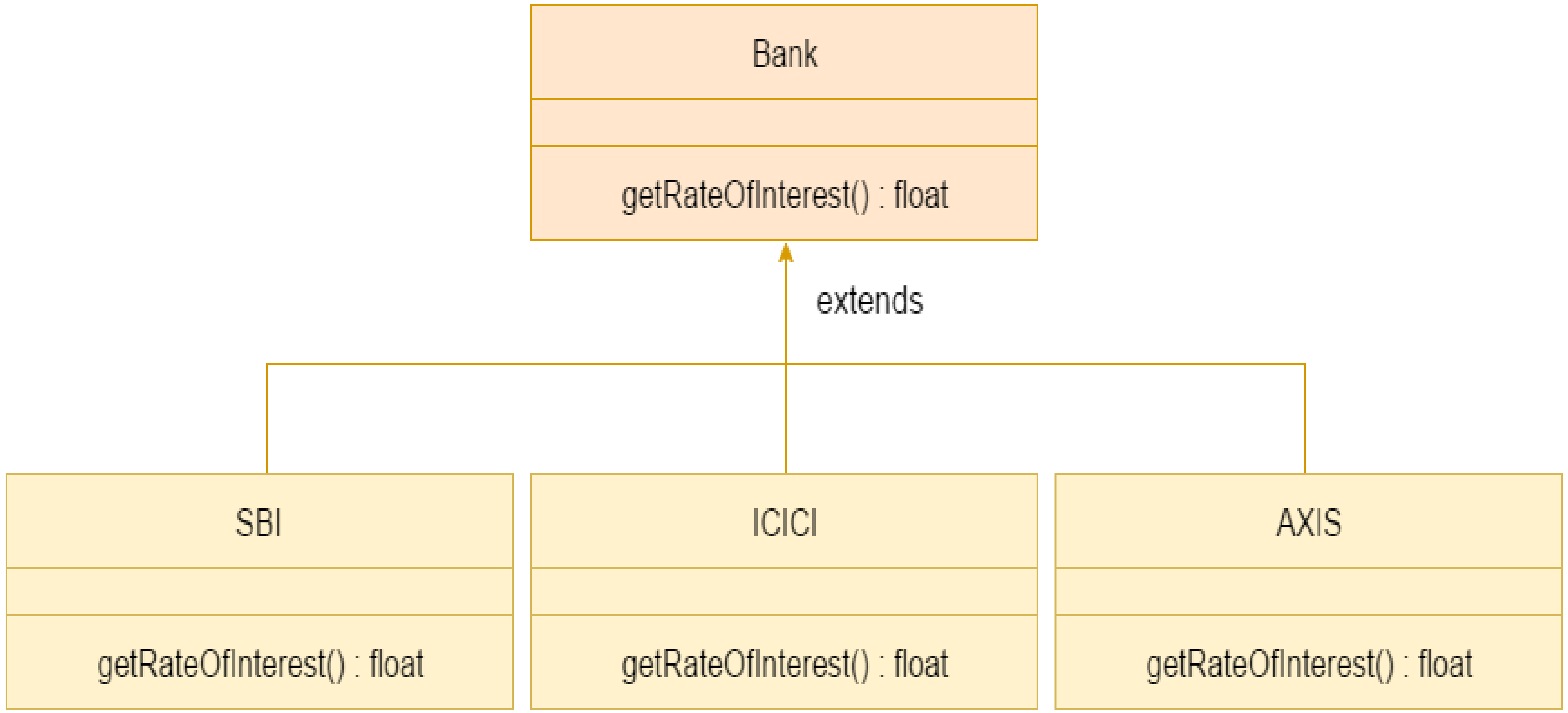
Method Overriding

- Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).
4. When we declare any method as final in super class at that time we can not override that method
5. for method overriding access specifier of the sub class method must be same or wider than super class.

- MethodOverridingDemo.java

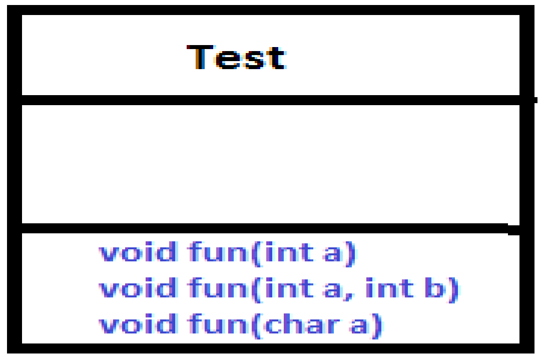
Method Overriding


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

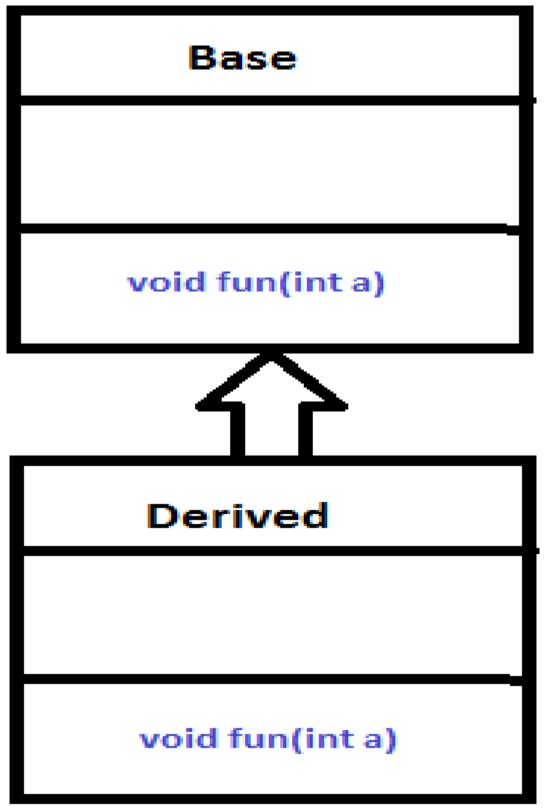
Method Overriding

- Static method cannot be overridden.
- It is because the static method is bound with class whereas instance method is bound with an object.
- Main method is static so we cannot override main method.
- Access Specifiers in Method Overriding
- The same method declared in the superclass and its subclasses can have different access specifiers. However, there is a restriction.
- We can only use those access specifiers in subclasses that provide larger access than the access specifier of the superclass.
- For example,
- Suppose, a method myClass() in the superclass is declared protected. Then, the same method myClass() in the subclass can be either public or protected, but not private.

Method Overriding



Overloading

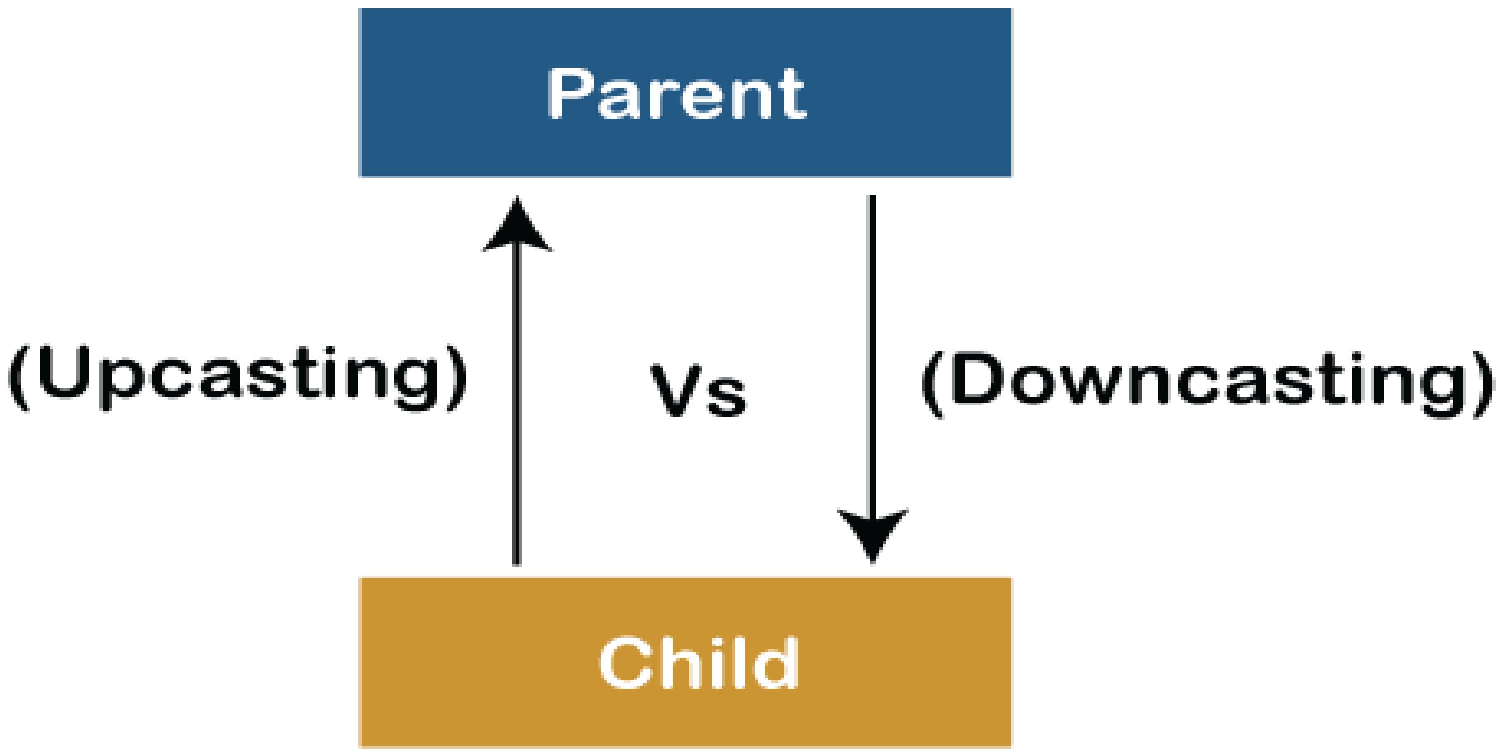


Overriding

Upcasting and Downcasting

- A process of converting one data type to another is known as **Typecasting** and
- **Upcasting** and **Downcasting** is the type of object typecasting.
- In Java, the object can also be typecasted like the datatypes.
- **Parent** and **Child** objects are two types of objects. So, there are two types of typecasting possible for an object, i.e., **Parent to Child** and **Child to Parent**
- **Typecasting** is used to ensure whether variables are correctly processed by a function or not
- In **Upcasting** and **Downcasting**, we typecast **a child object to a parent object** and **a parent object to a child object** simultaneously.
- We can perform Upcasting implicitly or explicitly, but downcasting cannot be implicitly possible.

Upcasting and Downcasting



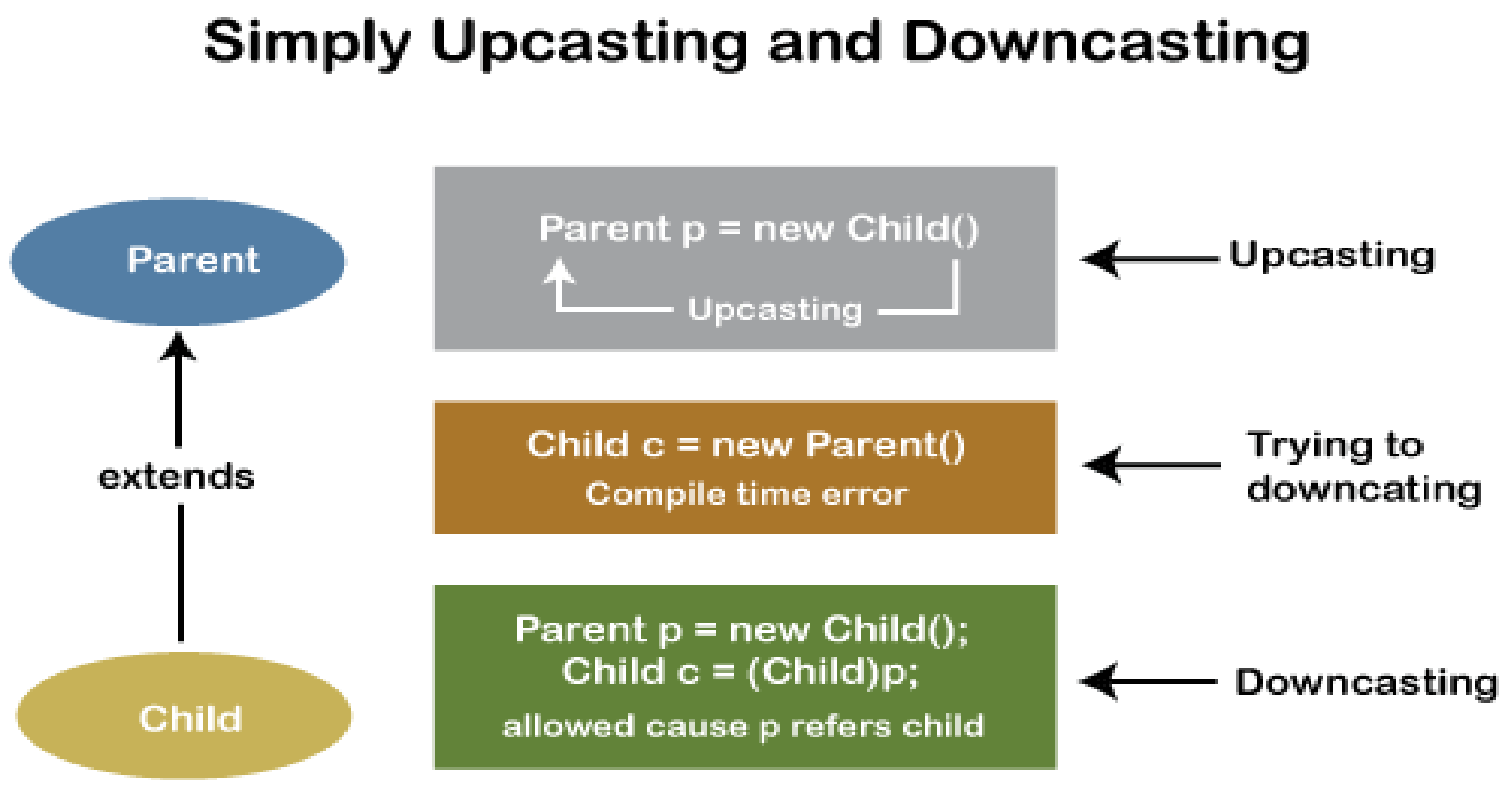
Upcasting

- **Upcasting** is a type of object typecasting in which a **child object** is typecasted to a **parent class object**.
- By using the Upcasting, we can easily access the variables and methods of the parent class to the child class.
- **Upcasting** is also known as **Generalization** and **Widening**.
- UpcastingExample.java

Downcasting

- **Downcasting** is another type of object typecasting.
- In Downcasting, we assign a parent class reference object to the child class.
- In Java, we cannot assign a parent class reference object to the child class, but if we perform downcasting, we will not get any compile-time error.
- DowncastingExample.java

Downcasting



Abstract Keyword

- The abstract keyword is used to achieve abstraction in Java.
- It is a non-access modifier which is used to create abstract class and method but not variables.
- The role of an abstract class is to contain abstract methods. However, it may also contain non-abstract methods.
- The method which is declared with abstract keyword and doesn't have any implementation is known as an abstract method.
- An abstract method is a method that is declared, but contains no implementation.

```
abstract class Employee
{
        abstract void work();
}
```

Rules of Abstract Keyword

- Do’s
- An abstract keyword can only be used with class and method.
- An abstract class can contain constructors and static methods.
- If a class extends the abstract class, it must also implement the abstract method.
- An abstract class can contain the main method and the final method.
- An abstract class can contain overloaded abstract methods.
- We can declare the local inner class as abstract.
- We can declare the abstract method with a throw clause.

Rules of Abstract Keyword

- Don’ts
- An abstract keyword cannot be used with variables and constructors.
- If a class is abstract, it cannot be instantiated.
- If a method is abstract, it doesn't contain the body.
- We cannot use the abstract keyword with **the final**.
- We cannot declare abstract methods **as private**.
- We cannot declare abstract methods **as static.**

Abstract Class

- Abstract class-
- An abstract classis a class that is declared abstract—it may or may not include abstract methods.
- But, if class contains at least single abstract method class must declare abstract.
- Abstract classes cannot be instantiated, but they can be sub-classed.
- Abstraction:
- –Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- For Example:
- AbstractExample1.java // abstract method and constructor in abstract class
- AbstractExample2.java // abstract class containing overloaded abstract methods

final keyword

- The **final keyword** in java is used to restrict the user.
- The java final keyword can be used in many context. Like,

1. variable
2. method
3. Class

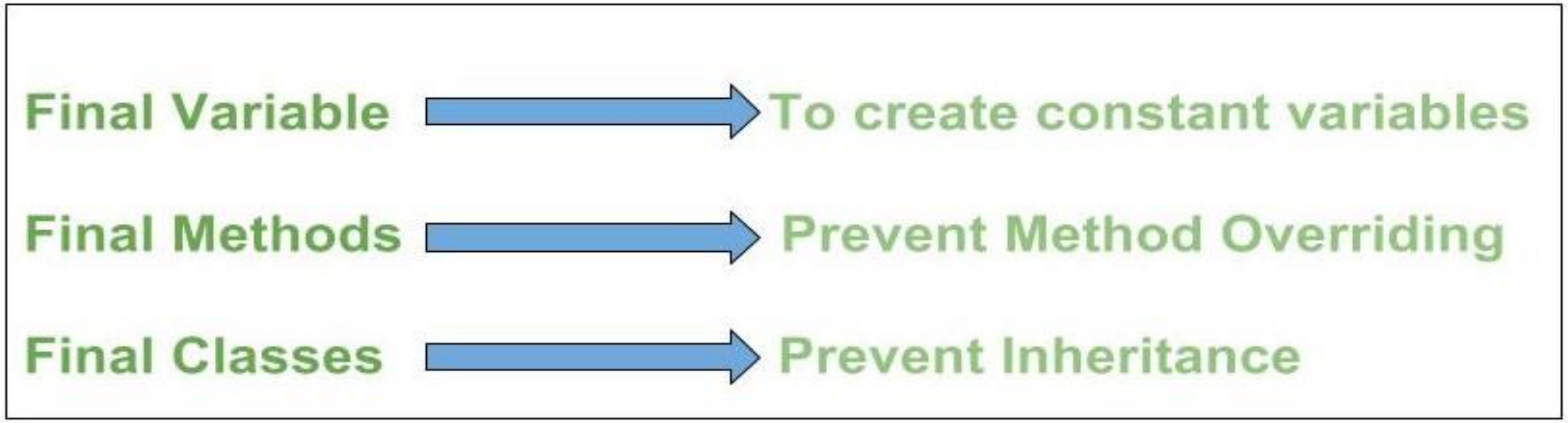
- The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable.
- It can be initialized in the constructor only.
- The blank final variable can be static also which will be initialized in the static block only.

final variable

- If you make any variable as final, you cannot change the value of final variable(It will be constant).
- For example: There is a final variable pi, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.
- AreaCal.java

final method and final class

- final method:
- if you make any method as final, you cannot override it.
- Circle.java
- final class:
- If you make any class as final, you cannot extend it.
- finalDemo.java



Interface

- Objects define their interaction with the outside world through the methods that they expose.
- Methods form the object's interface with the outside world;
- The buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off.
- Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

Interface

- It has static constants and abstract methods.
- The interface in Java is *a mechanism to achieve abstraction*.
- There can be only abstract methods in the Java interface, not method body.
- It is used to achieve multiple inheritance in Java.
- Java Interface also **represents the IS-A relationship**.
- It cannot be instantiated just like the abstract class.
- Since Java 8, we can have **default and static methods** in an interface.
- Since Java 9, we can have **private methods** in an interface.

Why use Java interface?

- There are mainly two reasons to use interface. They are given below.

1. It is used to achieve abstraction.

2. By interface, we can support the functionality of multiple inheritance.

- In its most common form, an interface is a group of related methods with empty bodies.

- An interface is not a class.

- Writing an interface is similar to writing a class, but they are two different concepts.

- A class describes the attributes and behaviours of an object.

- An interface contains behaviours that a class implements.

How to declare an interface?

- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

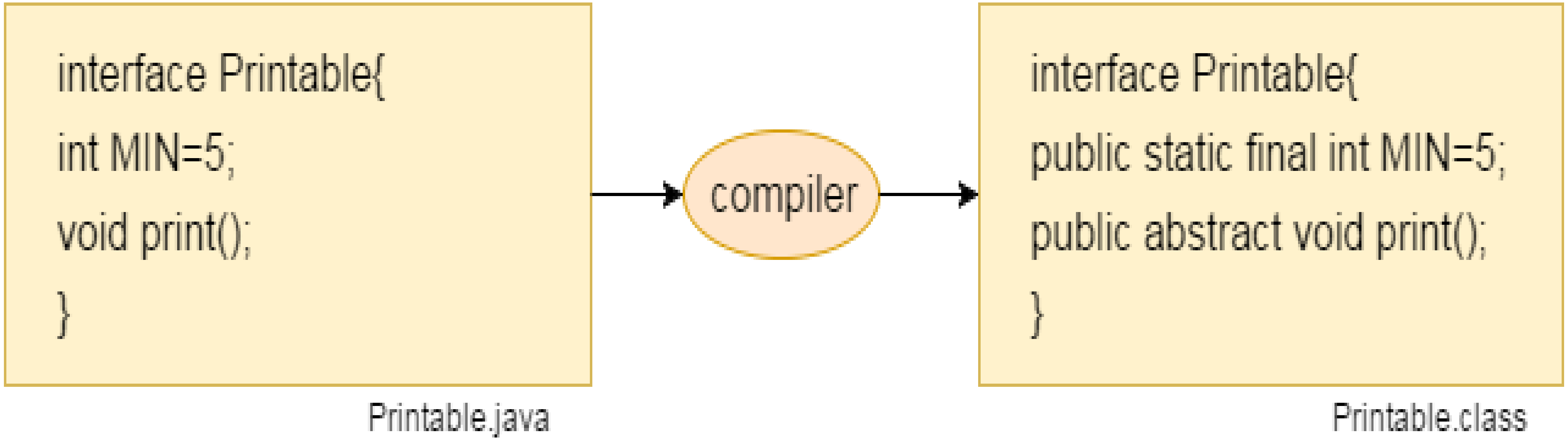
```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

```
interface Drawable{
void draw();
}
```

Interface

- The Java compiler adds public and abstract keywords before the interface method.
- It adds public, static and final keywords before data members.



```
interface Printable{
int MIN=5;
void print();
}
```

Printable.java

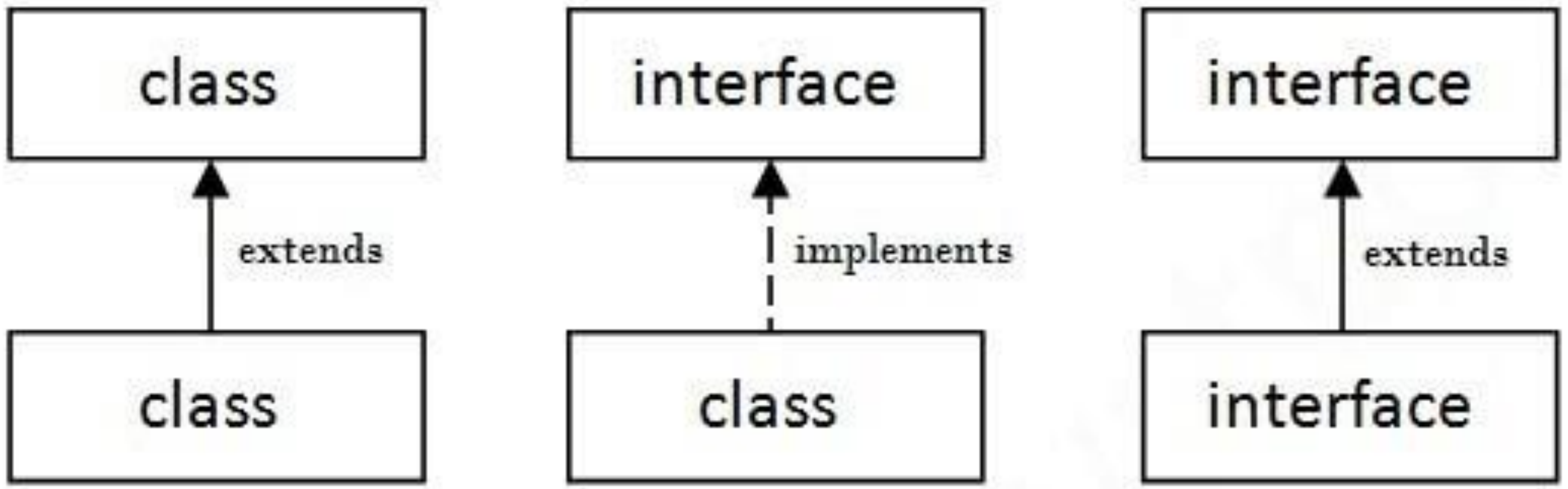
```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

Printable.class

Interface

The relationship between classes and interfaces

- As shown in the figure given below, a class extends another class, an interface extends another interface, but a class implements an interface.

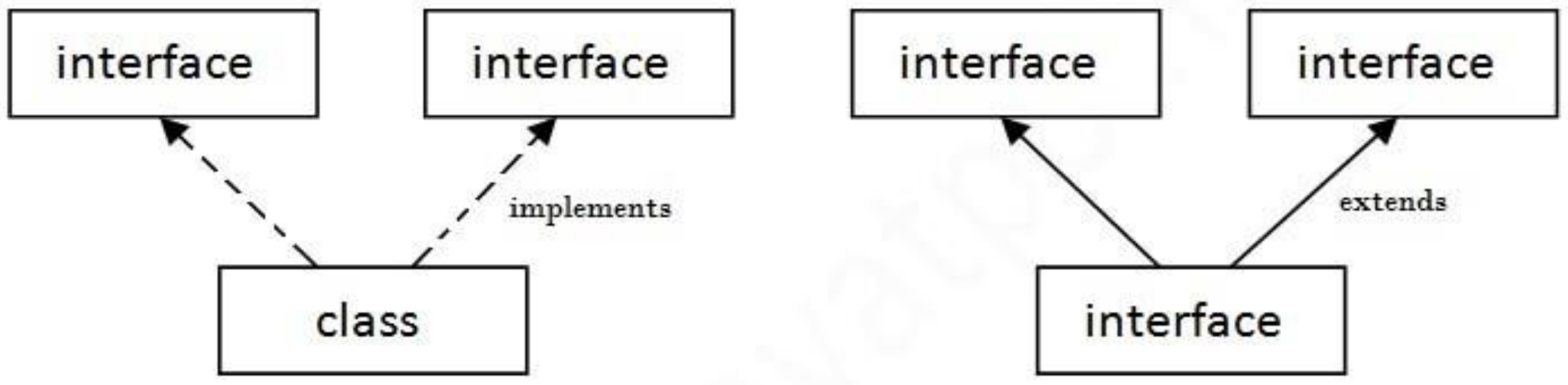


- TestInterface1.java

Interface

Multiple inheritance in Java by interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

- TestInterface3.java TestInterface4.java TestInterface5.java

Interface

- Java 8 Default Method in Interface
- Since Java 8, we can have method body in interface. But we need to make it default method.
- *TestInterfaceDefault.java*
- Java 8 Static Method in Interface
- Since Java 8, we can have static method in interface.
- *TestInterfaceStatic.java*
- marker or tagged interface
- An interface which has no member is known as a marker or tagged interface
- If a class includes an interface but does not fully implement the methods required by that interface, then that class must be declared as abstract.

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.

Simple Example of java Package

- The package keyword is used to create a package in java.

```
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

- How to compile java package

- If you are not using any IDE, you need to follow the syntax given below:

```
javac -d directory javafilename
```

```
javac -d . Simple.java
```

Simple Example of java Package

```
javac -d directory javafilename
```

```
javac -d . Simple.java
```

- The -d switch specifies the destination where to put the generated class file.
- You can use any directory name like d:/abc (in case of windows) etc.
- If you want to keep the package within the same directory, you can use . (dot).
- **How to run java package program**
- You need to use fully qualified name e.g. mypack.Simple etc to run the class.
- For Example:

To Compile: javac -d . Simple.java
To Run: java mypack.Simple

Package

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.

Package

1) Using packagename.$\boxed{*}$

- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The import keyword is used to make the classes and interface of another package accessible to the current package.
- A.java B.java

Package

2) Using packagename.classname

- If you import package.classname then only declared class of this package will be accessible.
- A.java   D.java

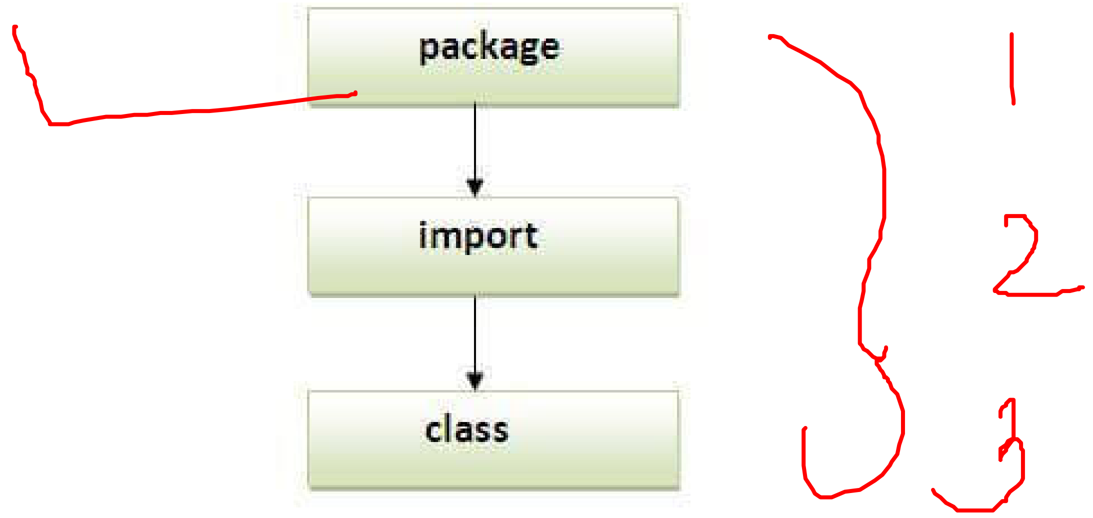
Package

3) Using fully qualified name

- If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import.
- But you need to use fully qualified name every time when you are accessing the class or interface.
- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.
- A.java D.java

Package

- If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.
- Sequence of the program must be package then import then class.



Subpackages

- Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.
- Let's take an example, Sun Microsystem has defined a package named java that contains many classes like System, String, Reader, Writer, Socket etc.
- These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on.
- So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.
- SimpleDemo.java

Subpackages

- Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.
- Let's take an example, Sun Microsystem has defined a package named java that contains many classes like System, String, Reader, Writer, Socket etc.
- These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on.
- So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.
- SimpleDemo.java