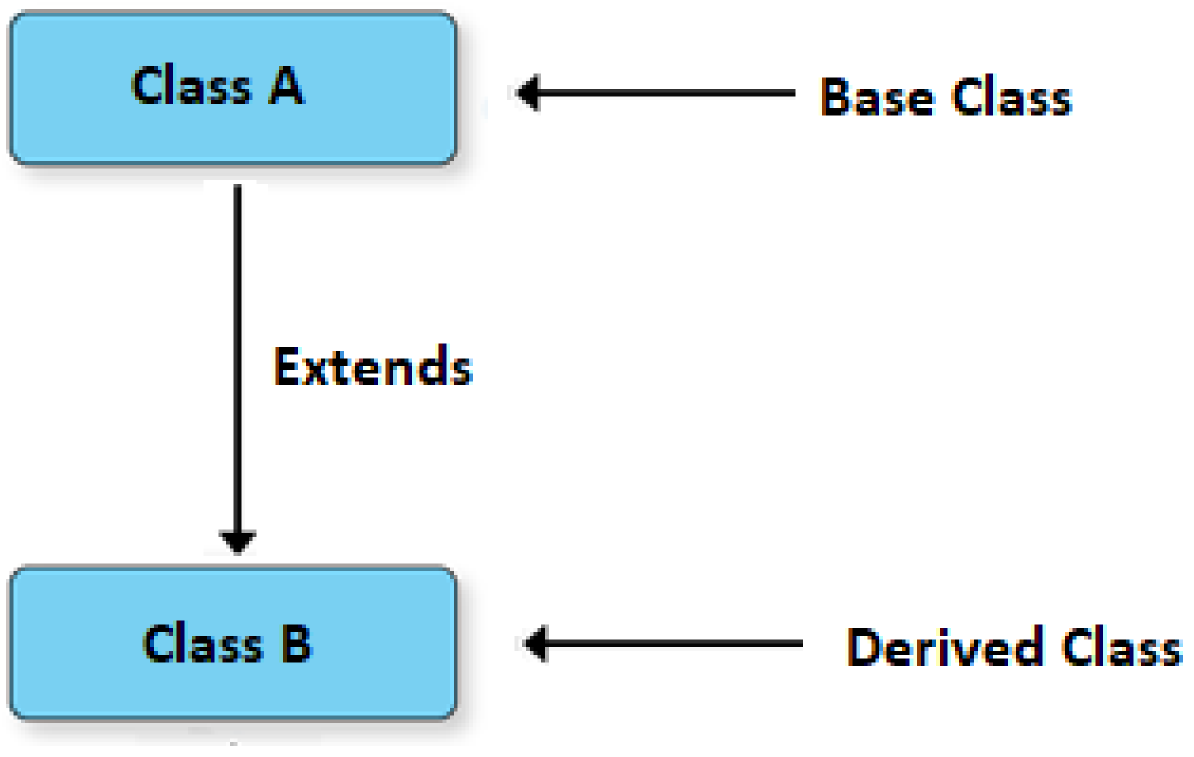
CJ Unit 3

Inheritance -

- The mechanism of deriving a new class from an old one is called inheritance.
- The old class is referred to as the base class and the new one is called the derived class or subclass.
- Inheritance provides the concept of reusability.
- The derived class inherits some or all of the traits from the base class.



Types of Inheritance -

1. Single Inheritance **A** → **B**

2. Multilevel Inheritance **A** → **B** → **C**
3. Multiple Inheritance **B, C** → **A**
4. Hierarchical Inheritance **A** → **B, C, D**
5. Hybrid Inheritance **A** → **B, C** → **D**

Super Keyword -

The keyword super is used to invoke the constructor method of the superclass.

Abstraction -

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Abstract Methods -

- A method without body (no implementation) is known as abstract method.
- A method must always be declared in an abstract class.
- **Example -**

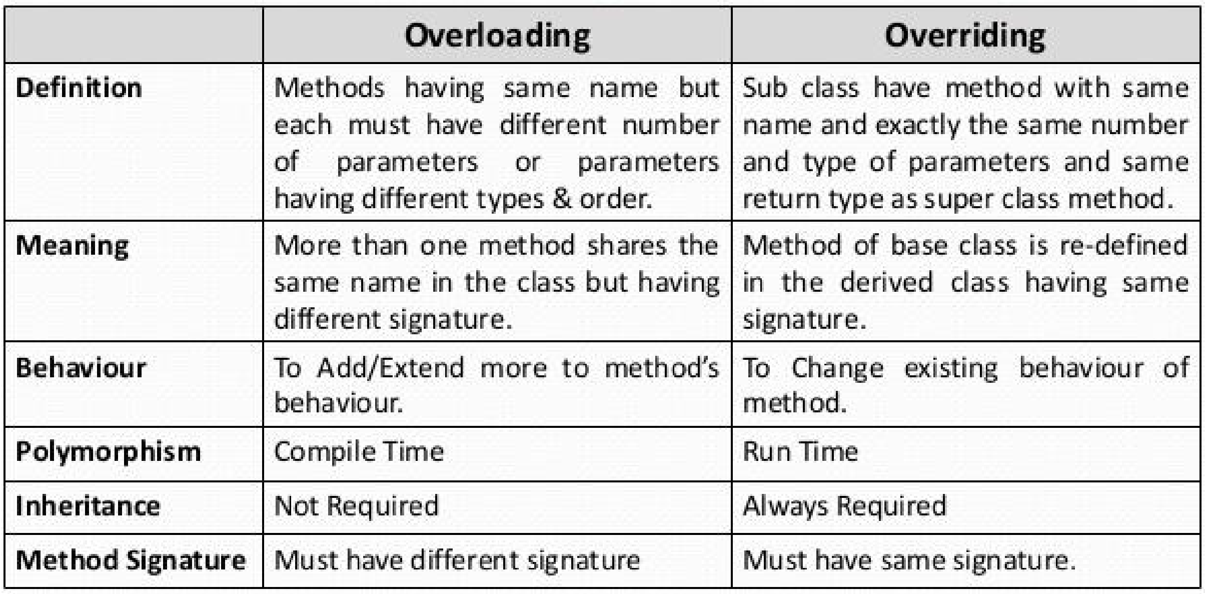
public abstract int

myMethod (int n1, int n2);

Method Overriding -

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
- Also Known as Run Time Polymorphism.
- Rules for Java Method Overriding -
    - The method must have the same name as in the parent class.
    - The method must have the same parameter as in the parent class.

Distinguish Between Method overloading and Overriding -



	Overloading	Overriding
Definition	Methods having same name but each must have different number of parameters or parameters having different types & order.	Sub class have method with same name and exactly the same number and type of parameters and same return type as super class method.
Meaning	More than one method shares the same name in the class but having different signature.	Method of base class is re-defined in the derived class having same signature.
Behaviour	To Add/Extend more to method's behaviour.	To Change existing behaviour of method.
Polymorphism	Compile Time	Run Time
Inheritance	Not Required	Always Required
Method Signature	Must have different signature	Must have same signature.

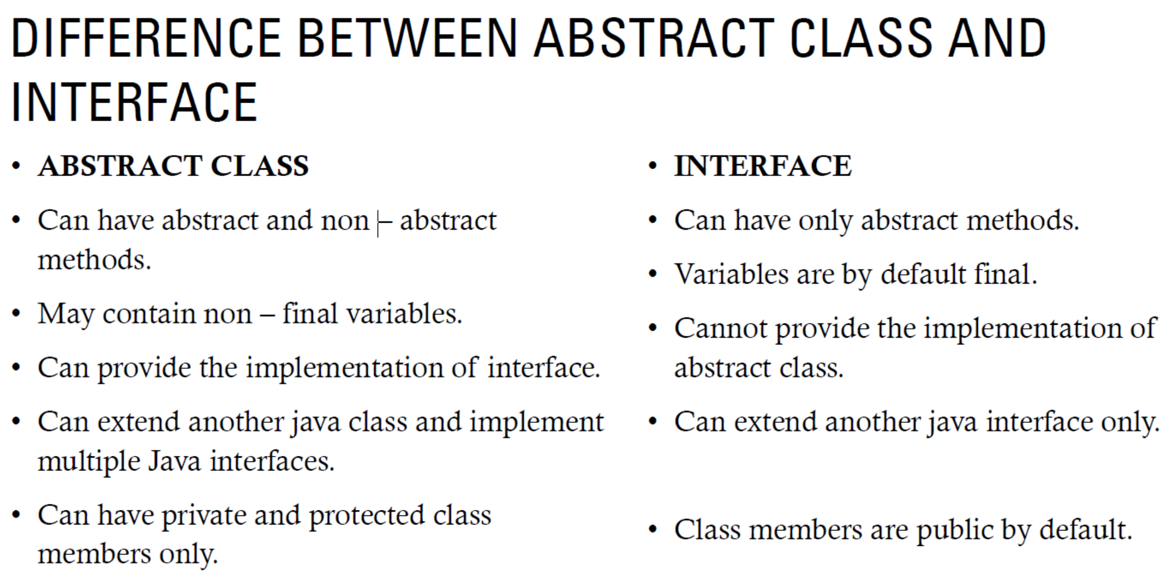
Interface -

- An **interface in Java** is a blueprint of a class. 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 abstraction and multiple inheritance in Java.
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.
- 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 three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.



DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

ABSTRACT CLASS	INTERFACE
Can have abstract and non – abstract methods.	Can have only abstract methods.
May contain non – final variables.	Variables are by default final.
Can provide the implementation of interface.	Cannot provide the implementation of abstract class.
Can extend another java class and implement multiple Java interfaces.	Can extend another java interface only.
Can have private and protected class members only.	Class members are public by default.

Java Lambda Expressions -

- Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.
- The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.
- Java lambda expression is treated as a function, so compiler does not create .class file.
- Syntax - (argument list) → {body}
- EXAMPLE - (int x) → {System.out.println(2*x)}

Types of Lambda Expressions -

Zero Parameter -

- () --> {System.out.println("Zero Parameter lambda");}

One Parameter -

- (p) --> {System.out.println(“OneParameter:”+p);}

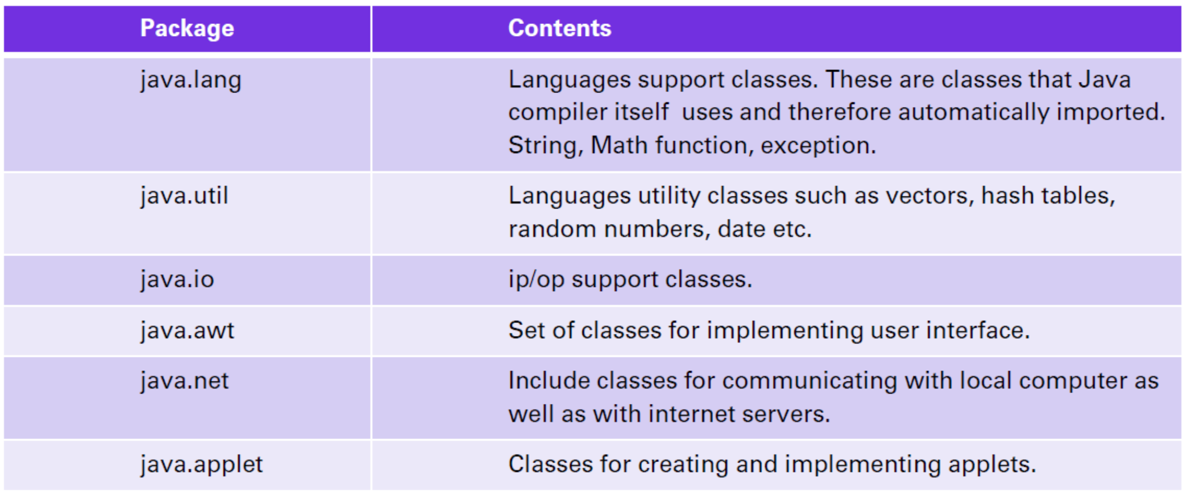
Multiple Parameter -

- (p1,p2) --> {System.out.println(“Multiple parameters: ”+p1+”,”+p2);}

Packages -

- Packages are Java’s way of grouping a variety of classes and /or interfaces together.
- **Benefits -**
    - Reuse of Classes & Interfaces
    - Provides a way to “hide” classes thus preventing other programs or
    - Packages from accessing classes that are meant for internal use only.
    - Provides a way for separating “design ” and “coding ”.
    - Classes in two different packages can have the same name.

Java API Packages -



Package	Contents
java.lang	Languages support classes. These are classes that Java compiler itself uses and therefore automatically imported. String, Math function, exception.
java.util	Languages utility classes such as vectors, hash tables, random numbers, date etc.
java.io	ip/op support classes.
java.awt	Set of classes for implementing user interface.
java.net	Include classes for communicating with local computer as well as with internet servers.
java.applet	Classes for creating and implementing applets.

Creating Packages -

1. Declare the package at the beginning of a file using the form
2. package packagename;
3. Define the class that is to be put in the package and declare it public.
4. Create a subdirectory (with same name as package)under the directory where the main source
   files are stored.
5. Store the listing as the classname.java file in the subdirectory created.
6. Compile the file. This creates .class file in the subdirectory.

```
// Example

package p1;
public class B
{
//body of B
}
```

Accessing a Package -

```
// Syntax
import package1[.package][.package3]. classname

// Example:
import
firstpackage.secondpackage.myclass
OR
import
packagename
```

Hiding Classes -

- When we import a package using asterisk(*) all public classes are imported. However, we may prefer to “not import” certain classes.
- That is we want to hide these classes from accessing from outside of the package. Such classes should be declared “not public”.

```
Package p1;
Public class X
{
//body of class X
}
class Y
{
//body of class Y
}
```