Chapter 2

Inheritance, Packages, Access Specifier

Topic	Details
Inheritance	➢ Types of Inheritance (Single, Multilevel, Hierarchical) ➢ Constructor in inheritance (super and this keyword) ➢ Super class & subclass ➢ Abstract method and classes ➢ Method overriding ➢ final keyword ➢ super keyword ➢ Defining and Implementing interfaces
Java Package and Access Specifier	➢ Importing classes ➢ User defined packages ➢ Modifiers & Access control (Default, public, private, protected, private protected) ➢ Understanding commonly used classes of java.lang package. ➢ Object class & String class ➢ Wrapper classes

Unit-2 - Inheritance(Part-1)

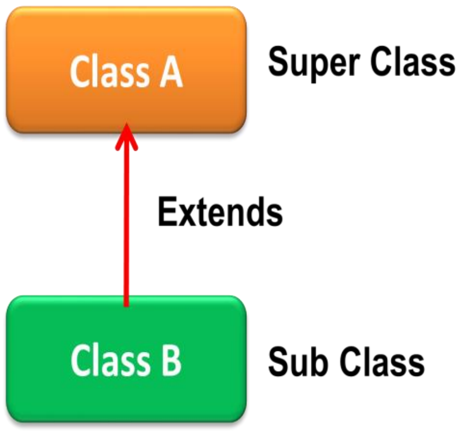
Inheritance

- 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 attributes of parent class, and you can add new methods and attributes also.
- Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.
- **Inheritance represents the parent-child relationship.**
- A class that is inherited is called **a super class/ Parent class**.
- The class that does the inheriting is called **a sub class/ Child class**.
- To inherit a class, you simply incorporate the definition of one class into another by using the **extend** keyword.

Example & Syntax:

Syntax:
```
class subcls-name extends supercls-name
{
    // body of class.
}
```

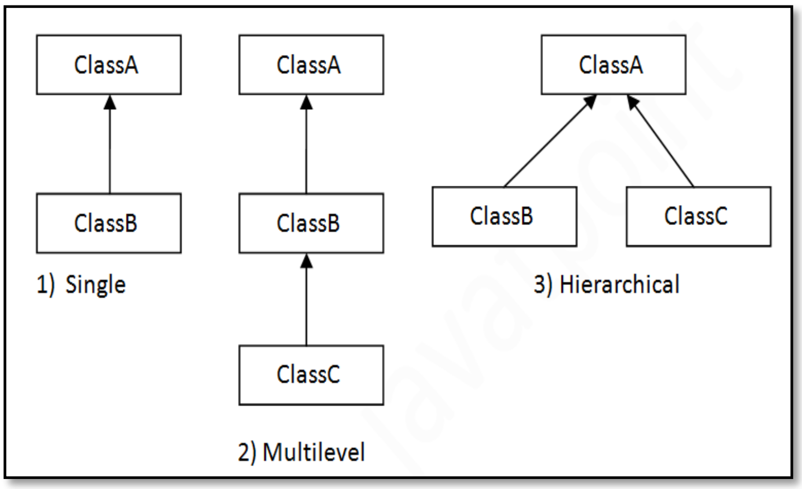
Example:
```
class A
{   //methods and attributes }
class B extends A
{
   //methods and attributes
}
```



Types of Inheritance

- There can be **three types of inheritance** in java:
  1. Single Inheritance
  2. Multilevel Inheritance
  3. Hierarchical Inheritance

❖ In java programming**, multiple and hybrid inheritance is not supported through class**.



1. Single Inheritance with Example

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

❖ In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class Animal
{
        void eat(){System.out.println("eating...");}
}
class Dog extends Animal
{
        void bark(){System.out.println("barking...");}
}
class Demo
{
        public static void main(String args[])
        {
                Dog d=new Dog();
                d.bark();  // method of Dog class
                d.eat();  // method of Animal class
        }
}
```

Output:

```
barking...
eating...
```

2. Multilevel Inheritance with Example

- When there is a chain of inheritance, it is known as multilevel inheritance.
- As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```
class Animal
{
        void eat(){System.out.println("eating...");}
}
class Dog extends Animal
{
        void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog
{
        void weep(){System.out.println("weeping...");}
}
class Demo
{
        public static void main(String args[])
        {
                BabyDog d=new BabyDog();
                d.weep();
                d.bark();
                d.eat();
        }
}
```

Output:

```
weeping...
barking...
eating...
```

3. Hierarchical Inheritance with Example

❖ When two or more classes inherit a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
class Animal
{
        void eat(){System.out.println("eating...");}
}
class Dog extends Animal
{
        void bark(){System.out.println("barking...");}
}
class Cat extends Animal
{
        void meow(){System.out.println("meowing...");}
}
class Demo
{
        public static void main(String args[])
        {
                Cat c=new Cat();
                c.meow();
                c.eat();
        }
}
```

Output:

```
meowing...
eating...
```

- **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.

1 WORD QUESTION ANSWER

no.	Question	Answer
1.	________Is the process of creating new classes, called derived classes, from existing classes, which are often called base classes.	Inheritance
2.	________ Inheritance represents a form of inheritance when there is only one base class and one derived class.	Single
3.	________Inheritance represents a kind of inheritance when a derived class inherits properties of multiple classes.	Multiple
4.	________Inheritance represents a type of inheritance when a derived class is a base class for another class.	Multilevel
5.	When there is a need to create multiple derived classes that inherit properties of the same base class is known as ________ inheritance.	Hierarchical

Use of 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**.
2. **super** can be used to invoke immediate **parent class method**.
3. **super()** can be used to invoke immediate **parent class constructor**.

1. Accessing parent class instance variable

- ❖ We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

Example:

```
class Animal
{
        String color="white";
}
class Dog extends Animal
{       String color="black";
        void printColor()
        {       System.out.println(color);//prints color of Dog class
                System.out.println(super.color);//prints color of Animal class
        }
}
class Demo_Super
{       public static void main(String args[])
        {       Dog d=new Dog();
                d.printColor();
        }
}
```

Output:

```
black
white
```

2. Accessing parent class method

❖ The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

Example:

```
class Animal
{
        void eat( ){System.out.println("eating...");}
}
class Dog extends Animal
{       void eat(){System.out.println("eating bread...");}
        void bark(){System.out.println("barking...");}
        void work()
        {       super.eat();
                bark();
        }
}
class Demo
{       public static void main(String args[])
        {       Dog d=new Dog();
                d.work();
        }
}
```

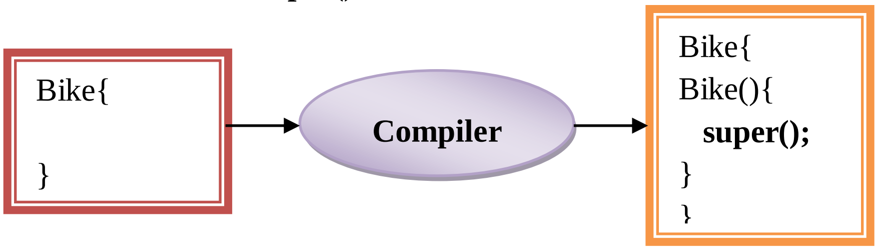
Output:

```
eating...
barking...
```

Constructor in Inheritance

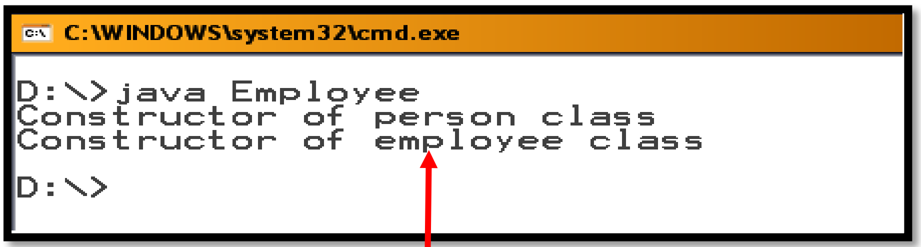
3. Accessing parent class constructor

- ❖ The super keyword can also be used to invoke the parent class constructor.
- ❖ **super() is added in each class constructor automatically by compiler if there is no super() or this().**
- ❖ **So there is no need to add super() to call base class default constructor**



Example:

```
class Person
{    Person()
  { System.out.println("Constructor of person class");}
}
class Employee extends Person
{     Employee()
      { System.out.println("Constructor of employee class"); }
       public static void main(String args[])
       {       Employee e1 = new Employee();
       }
}
```



```
C:\WINDOWS\system32\cmd.exe
D:\>java Employee
Constructor of person class
Constructor of employee class

D:\>
```

Note that the super class person constructor executes before the subclass Employee

1 WORD QUESTION ANSWER

NO.	QUESTION	ANSWER
1.	Which packages contains abstract keyword?	java.lang
2.	If super class and subclass have same variable name, which keyword should be used to use super class?	super

Method Overriding

- If **subclass (child class) has the same method as declared in the parent class**, it is known as method overriding in java.
- **Same name and Same Signature but in different class having parent child relationship.**
- Method overriding is used **for runtime polymorphism**

Example:

```
class Bank
{       int getRateOfInterest(){return 0;}  }
class SBI extends Bank
{  int getRateOfInterest(){return 8;}  }
class ICICI extends Bank
{  int getRateOfInterest(){return 7;}  }
class AXIS extends Bank
{  int getRateOfInterest(){return 9;}  }
class Test_Bank
{       public static void main(String args[])
        {
        SBI s=new SBI();
        ICICI i=new ICICI();
        AXIS a=new AXIS();
        System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
        System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
        System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
        }
}
```

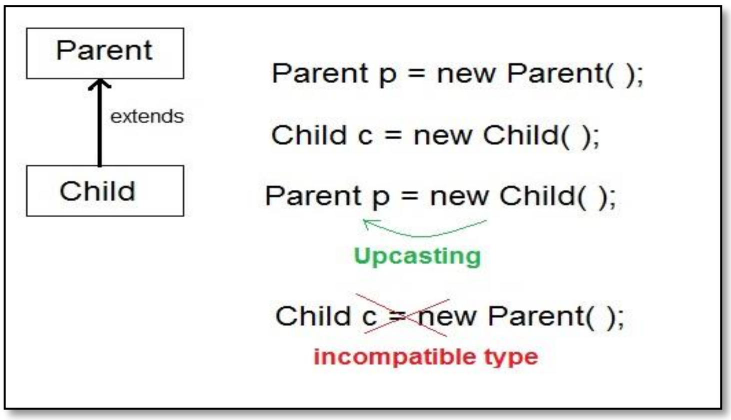
Output:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Difference between Method overloading and Method Overriding

Method Overloading	**Method Overriding**
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.
Method overloading is performed within a class.	Method overriding occurs in two classes that have IS-A relationship.
In case of method overloading parameter must be different.	In case of method overriding parameter must be same.
Compile time polymorphism	**Run time polymorphism**

Dynamic Method Dispatch in inheritance or Dynamic Binding



- ❖ Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at **run time, rather than compile time.**
- ❖ **When reference variable of Parent class refers to the object of Child class, it is known as upcasting.**

For example:

```
class A{}
class B extends A{}
A a=new B();//upcasting
```

```
class Game
 {        public void type( )
          { System.out.println("Indoor & outdoor"); }
}
class Cricket extends Game
{         public void type( )
          { System.out.println("outdoor game"); }
          public static void main(String[] args)
          {         Game gm = new Game();
                    Cricket ck = new Cricket();
                     gm.type();
                    ck.type();
                     gm=ck;//gm refers to Cricket object
                    gm.type(); //calls Cricket's version of type
          }
}
```

Output:

```
Indoor & outdoor
outdoor game
outdoor game
```

1 word Question Answer

NO.	QUESTION	ANSWER
1.	When specifies any class inside another class, then its known as _______class.	Nested
2.	When classes defined and used inside function block then it is known as __________ class.	Local
3.	Local class cannot have _________ data members	Static
4	What is nested class?	Class inside a class
5	Which feature of oop reduces the use of nested classes?	Inheritance
6	How many categories are nested classes divided into?	2
7	The nested class can be declared _____________________	Public, protected, private or package private
8	Use of nested class _____________ encapsulation.	Increases

Abstract & final class

1. • Class must declare with an abstract keyword
2. • It can have abstact and non abstract methods
3. • It can not be instantiated (Object)
4. • It can have final methods
5. • It can have constructors and static methods also

❖ An abstract class is a class that is declared abstract—it may or may not include abstract methods.
❖ Object of Abstract class cannot be created, but it can be sub classed.
❖ An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract Returntype methodName(argu list);
```

❖ If any class includes abstract methods, the class itself must be declared as abstract.

- If you are extending any abstract class having abstract method, you must either provide the implementation of the method or make this sub class also abstract.

Example:

```
abstract class Shape{  abstract void draw();  }
class Rectangle extends Shape
{   void draw( ) { System.out.println("drawing rectangle");}     }
class Circle extends Shape
{   void draw(){System.out.println("drawing circle");}           }
class Test_Abstract
{
    public static void main(String args[])
    {          Shape s=new Circle();
               s.draw();
               s= new Rectangle();
               s.draw();
     }
 }
```

Final Keyword

- The final keyword in java is used to restrict the user.
- final can be:
  1. variable
  2. method
  3. class
- **Java final variable:** if you make variable as final you cannot reinitialized it. Which used to declare a constant variable.
- **Java final method:** If you make any method as final, you cannot override it.
- **Java final class :**If you make any class as final, you cannot extend it.

Java final method

Example:

```
class Bike
{    final void run() {System.out.println("running");}    }
class Honda extends Bike
{         void run()  X // compile time error
          { System.out.println("running safely with 100kmph"); }
           public static void main(String args[])
          {
```

```
                         Honda honda= new Honda();
                        honda.run();
              }
 }
```

Java final class

Example:

```
final class Bike
{  void run(){System.out.println("running ”); } }
class Honda extends Bike   X // compile time error
{             void run() {System.out.println("running safely ");}
              public static void main(String args[])
              {         Honda honda= new Honda();
                        honda.run();
               }
}
```

1 word Question Answer

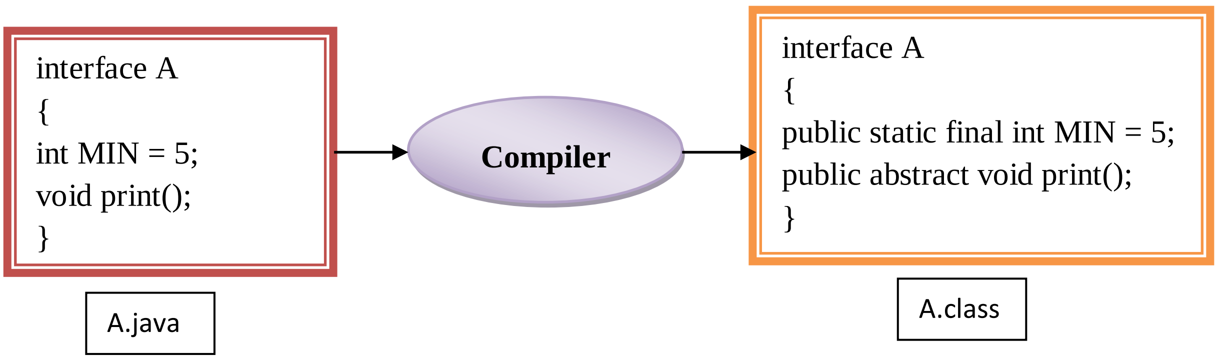
NO.	QUESTION	ANSWER
1.	Final keyword in java is used with	A. Class B. Class fields C. Class methods
2.	A class that can not be sub classed is called a____	Final class
3.	What is the use of final class?	High level of security in your application
4	Which class cannot be extended?	Final class
5	Can we declare constructors as final? (true/ false)	False

Interface

- ❖ An interface in java is a **blueprint of a class**.
- ❖ Interface contains **static constants and abstract methods only**.
- ❖ The interface in java is a mechanism to achieve **fully abstraction**.
- ❖ There can be only abstract methods in the java interface not method body. It is used to achieve **fully abstraction and multiple inheritance in Java**.
- ❖ It cannot be instantiated just like abstract class.

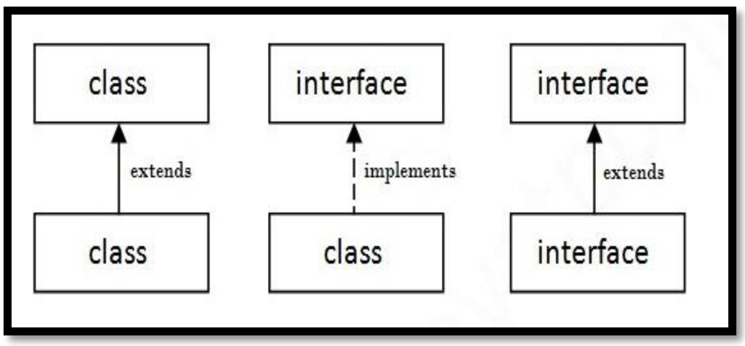
- ❖ An interface can contain any number of methods.
- ❖ An interface is written in a file with a (.java) extension, with the name of the interface matching the name of the file.
- ❖ The bytecode of an interface appears in a (.class) file.
- ❖ A class extends another class, an interface extends another interface but a class implements an interface.

Note: The java compiler adds **public** and **abstract** keywords before the interface method and **public, static** and **final** keywords before data members



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

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



- ❖ The **interface keyword** is used to declare an interface.
- ❖ Here is a simple example to declare an interface:
- ❖ **Syntax:**

```
interface NameOfInterface
{
   //Any number of final, static fields
  //Any number of abstract method declarations
 }
```

Example of Multiple inheritance using interface

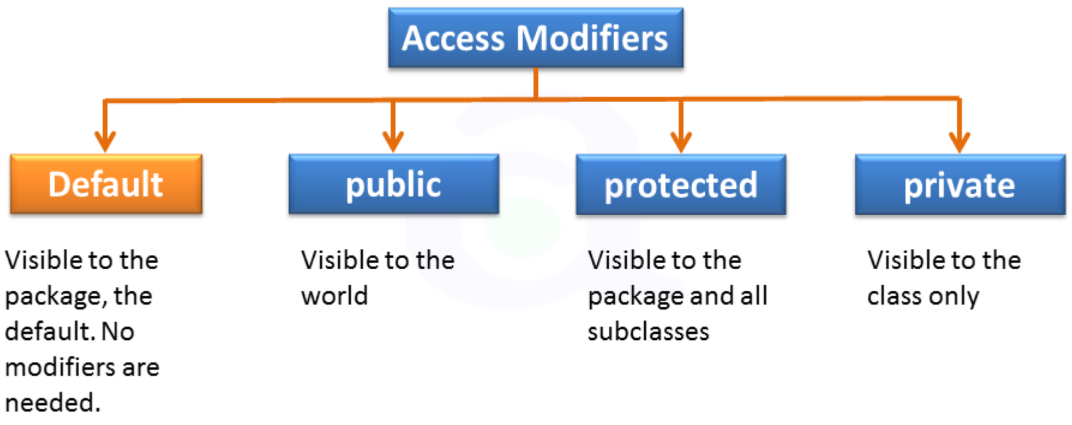
```
interface Printable
{           void print( );  }
interface Showable
{           void show();  }
class Demo implements Printable,Showable
{
  public void print(){System.out.println("Hello");}
  public void show(){System.out.println("Welcome");}
 public static void main(String args[])
  {
            Demo obj = new Demo();
            obj.print();
            obj.show();
  }
}
```

1 WORD QUESTION ANSWER

NO.	QUESTION	ANSWER
1.	Which keyword is used to define interfaces in java?	Interface
2.	Which can be used to fully abstract a class from its implementation?	Interfaces
3.	Which access specifies can be used for an interface?	Public
4.	Which keyword is used by a class to use an interface defined previously?	Implements

Unit-2 - Packages (Part-2)

Access Specifier/Modifiers

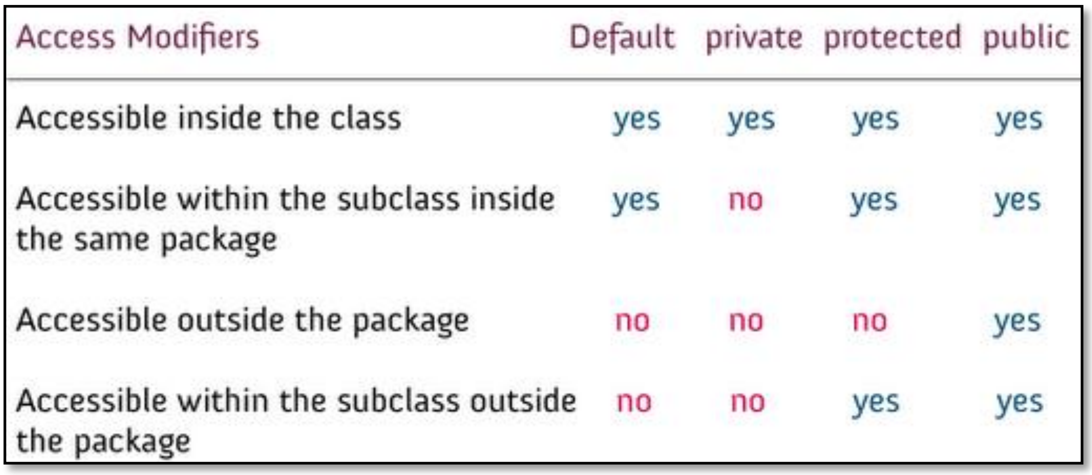

Access Modifiers
Default
public
protected
private
Visible to the package, the default. No modifiers are needed.
Visible to the world
Visible to the package and all subclasses
Visible to the class only

- ❖ Java supplies a rich set of access specifies. Some aspects of access control are related to inheritance or packages. Let's begin by examining access control as it applies to a single class.
- ❖ Java's access Specifier are:

1. **Public**
   - When a member of a class is modified by the public specifier, then that member **can be accessed by any other code.**
2. **Private**
   - When a member of a class is specified as private, then that member can only be accessed by other members of its class.
3. **Protected**
   - If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then declare that element protected.
4. **Default**
   - When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package.
   - Now, we can understand why main (), has always been preceded by the public specifier. It is called by the code outside of the program. (by java run-time system).
   - When no access specifier is used, then by default the member of a class is public within its own package, **but cannot be accessed outside of its package.**

```
public int i;
private double j;
private int myMethod (int a, char b);
```

Summary:



Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes

❖ T

1 WORD QUESTION ANSWER

NO	QUESTION	ANSWER
1.	Data members and member functions of a class in java program are by default	Private
2.	The classes in c++ are used to manipulate ____________ .	Data and functions
3.	Variables of class is known as __________ and function known as __________.	Data member, member function
4.	The _________members are accessible only inside the class or their own, class's member.	Private
5.	The _________ members accessible inside the class and also outside the class also.	Public

Normal Import

❖ Consider the java following import statements:
1. **import package.ClassA;**
2. **import package.*;**
3. **Fully qualified name;**

❖ Here (1) gives you a license to use ClassA inside the whole program without the package reference. That is you can use like

```
ClassA obj = new ClassA(); or ClassA.getStatic()
```

- ❖ Here statement (2) allows you to **use all the java classes** that belong the **imported package** in the above manner. That is without the package reference.
- ❖ Here statement (3) allows you to don't use import statement, you can still use the classes of that package. But you should invoke it with package reference where ever you use.

```
package.ClassAobj = new package.ClassA(); – looks very ugly isn't it?
```

Static Import

- ❖ Static import in java allows to import static members of class and use them, as they are declared in the same class. Static import is introduced in JDK 5.

Advantages of Static Import in Java

- Less coding is required if you have access any static member of a class often.

Drawback of Static Import in Java

- If you overuse the static import feature, it makes the program unreadable and unmanageable.

Simple Example of static import

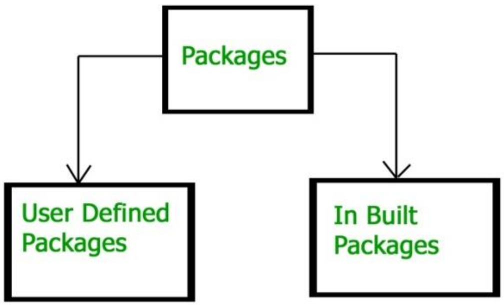
```
import static java.lang.System.*;
class Test
{
        public static void main(String args[])
        {       out.println("Hello");//Now no need of System.out
                out.println("Java");

        }
}
```

1 WORD QUESTION ANSWER

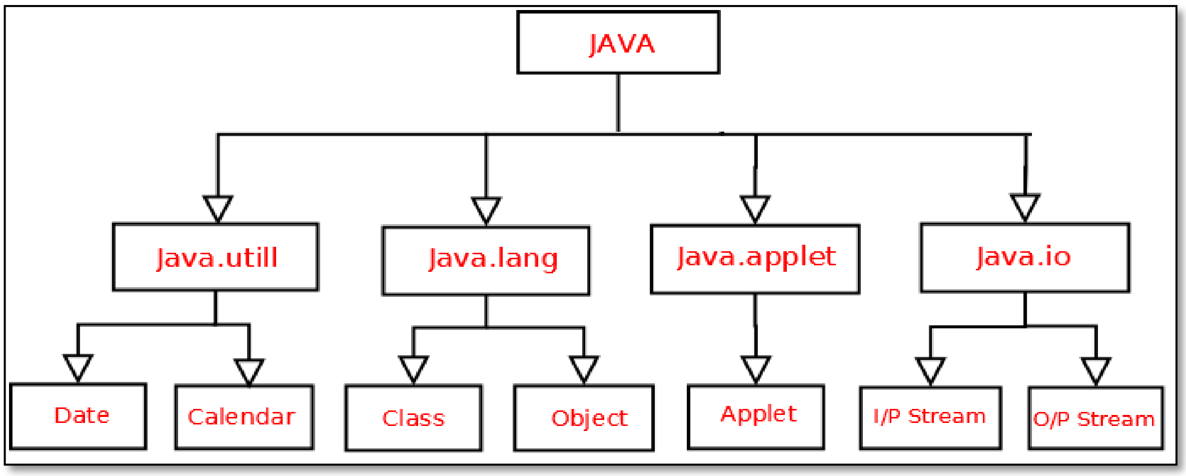
SR NO.	Question	Answer
1.	Static members are declared in ________	Same class
2.	Static import allow user to____________	Access static member of class

Types of Java packages



- ❖ A **java package** is a group of **similar types of classes, interfaces and sub-packages**.
- ❖ Types of packages :
  1. **built-in package – System defined packages**
  2. **user-defined package**

Java API Packages (System Defined Packages)



Java System Packages and Their Classes

java.lang	Language support classes.
java.util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
java.io	Input/output support classes.
java.applet	Classes for creating and implementing applets.
java.awt	Set of classes for implementing graphical user interface.

Creating & Using User Defined Package &Subpackage

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

Example:

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

```
Compile: javac –d . Simple_Pack.java
( Here dot represents current directory )
Run : java mypack.Simple_Pack
```

Access package from another package

```
package pack;
public class A
{    public void msg()
        {         System.out.println("Hello");   }
}
====================
package mypack;
import pack.*;
class B
{    public static void main(String args[ ])
        {     A obj = new A( );
                obj.msg();
        }
}
```

Java.lang Package

- ❖ Provides classes that are fundamental to the design of the Java programming language.
- ❖ Following are few important classes of java.lang package
  1. Math
  2. Wrapper Classes
  3. String
  4. String Buffer

Wrapper class in java.lang package

- ❖ The wrapper classes in Java are used to convert primitive types (int, char, float, etc) into corresponding objects.
- ❖ Each of the 8 primitive types has corresponding wrapper classes.

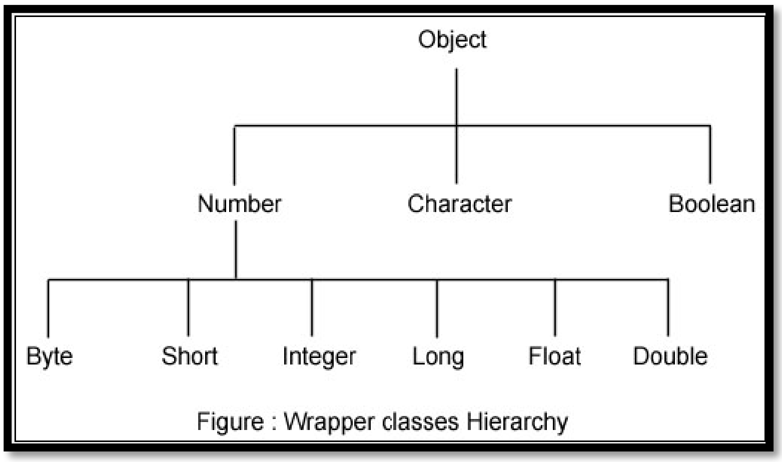


Figure : Wrapper classes Hierarchy

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

Wrapper Conversations

1. Convert Object number to primitive number

Method	Action
Integer val = new Integer(10); int i = val.intValue( )	Integer object to int

Float val = new Float(10.5); Float f = val.floatValue( )	Float object to float
Long val = new Long(100000); long l = val.longValue( )	Long object to long
Double val = new Double(10); double d = val.doubleValue( )	Double object to double

2. **Converting numbers to String**
   - Using **toString( ) of Object class** any Number object can converted into String object.

   For ex:

   ```
   Integer  val = new Integer(10);
   String str = val.toString( ); //now str = 10
   ```

   - Using **vlaueOf( ) method** of Wrapper class we can convert any object to String object.

   For ex:

   ```
   String s1= String.valueOf('a'); // s1="a"
   String s2=String.valueOf(true); // s2="true"
   ```

3. **Convert numeric string to primitive data types**
   - The wrapper classes also provide methods which can be used to convert a String to any of the primitive data types, except character.
   - All these methods are static. These methods have the **format parse*()** where * refers to any of the primitive data types except char.

   For ex:

   ```
   String str = “34”;
   int x = Integer.parseInt(str); // x=34
   String str = “34.45”;
   double y = Double.parseDouble(str); // y =34.45
   ```

String class in java.lang package

- The java.lang.String class provides many useful methods to perform operations on sequence of char values.
- **Constructors:**

String( )	Constructs a new empty String.	String ct=new String()

String(String)	Constructs a new String that is a copy of the specified String.	String ct=new String(“Rajkot”)

String(char[ ])	Constructs a new String whose initial value is the specified array of characters.	char[ ] c={‘r’, ’a’, ’j’, ’k’, ’o’, ‘t’};
		String ct=new String(c);

String(char[ ], int, int)	Constructs a new String whose initial value is the specified sub array of characters.	char[] c={‘r’, ’a’, ’j’, ’k’, ’o’, ‘t’}; String ct=new String(c,0,2); //raj
String(byte[])	Constructs a new String whose initial value is the specified array of bytes.	byte[] b={65,66,67,68,69}; String str=new String(b); //ABCDE
String(byte[], int, int)	Constructs a new String whose initial value is the specified sub array of bytes.	byte[] b={65,66,67,68,69}; String str=new String(b,2,3); //CD

(NOTE : ALL METHODS FOR 1 MARKS)

Method	**Description**
char charAt(int index)	Returns char value for the particular index
int length()	Returns string length
String substring(int beginIndex)	Returns substring for given begin index.
String substring(int beginIndex, int endIndex)	Returns substring for given begin index and end index.
boolean contains(Char Sequences)	Returns true or false after matching the sequence of char value.
boolean equals(Object another)	Checks the equality of string with the given object.
Boolean isEmpty()	Checks if string is empty.
String concat(String str)	Concatenates the specified string.
String replace(char old, char new)	Replaces all occurrences of the specified char value.
static String equalsIgnoreCase(String another)	Compares another string. It doesn't check case.
int indexOf(int ch)	Returns the specified char value index.
String toLowerCase()	Returns a string in lowercase.
String toUpperCase()	Returns a string in uppercase.
String trim()	Removes beginning and ending spaces of this string.

int compareTo(String)	Compares this String to another specified String if match then return zero(0) otherwise not zero(0).
int compareToIgnoreCase(String)	Compares two strings lexicographically, ignoring case differences.

Example:

```
class Test_Equals
{
        public static void main(String[ ] args)
        {       String s1 = new String("ATMIYA");
                String s2 = new String("ATMIYA");
                String s3 = s2;
                if(s1==s2)
                        System.out.println("S1 & S2 are identical objects");
                if(s1.equals(s2))
                        System.out.println("S1 & S2 are identical in values");
                if(s3==s2)
                        System.out.println("S3 & S2 are identical objects");
                if(s3.equals(s2))
                        System.out.println("S3 & S2 are identical in values");
        }
}
```

Output:

```
S1 & S2 are identical in values
S3 & S2 are identical objects
S3 & S2 are identical in values
```

StringBuffer class in java.lang package

- Java StringBuffer class is used to **create mutable (modifiable)** string.
- String represents **fixed length and immutable** character sequence.
- StringBuffer allows **to grow and also writable character** sequence.
- **Ex:** StringBuffer str=new StringBuffer(“ATMIYA”);
- **Constructors:**

StringBuffer( )	Reserves room for 16 characters
StringBuffer(int size)	Explicitly sets the size of buffer
StringBuffer(String str)	Sets the initial content of the string Buffer and allocates room

	for 16 more characters

(NOTE : ALL METHODS FOR 1 MARKS)

Method	Description
append(String s)	Is used to append the specified string with this string.
insert(int offset, String s)	Is used to insert the specified string with this string at the specified position.
replace(intstartIndex, intendIndex, String str)	Is used to replace the string from specified startindex and endindex.
delete(int startIndex, int endIndex)	Is used to delete the string from specified startindex and endindex.
reverse()	Is used to reverse the string.
capacity()	Is used to return the current capacity.
charAt(int index)	Is used to return the character at the specified position.
length()	Is used to return the length of the string i.e. Total number of characters.
substring(int beginIndex)	Is used to return the substring from the specified begin index.
int indexOf(String str)	returns the index within this string of the first occurrence of the specified substring.

Universal class of JAVA (Object Class)

- The **Object class** is the parent class of all the classes in **java by default.**
- The Object class, in the **java.lang package**, sits at the top of the class hierarchy tree.
- Every class you use or write inherits the instance methods of Object. You need not use any of these methods, but, if you choose to do so, you may need to override them with code that is specific to your class.
- The methods inherited from Object that are discussed in this section are:
- **The methods in the object class are**

FOR 1 MARKS

clone( )	Creates and returns a copy of this object.
equals (Object obj)	Indicates whether some other object is "equal to" this one.

finalize ( )	Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
getClass( )	Returns the runtime class of this Object.
hashCode( )	Returns a hash code value for the object.
notify( )	Wakes up a single thread that is waiting on this object's monitor.
notifyAll( )	Wakes up all threads that are waiting on this object's monitor.
toString( )	Returns a string representation of the object.

wait( )
wait(long timeout)
wait(long timeout, int nanos)
	Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.