2.7. INNER CLASSES

Inner classes are the nested classes. We can simply represent that are defined inside the other classes.The below syntax defining the inner class is

```
Access_modifier class outerClass
{
        //code
Access_modifier class InnerClass
{
        //code
}
}
```

Properties of Inner class:

1 .The outer class can take over as many number of innerclass objects as it needs.
2.If the outer class and the equivalent inner class both are public then any new class can create an instance of this inner class.
3.The inner class objects do not get instantiated with outer class object
4.outer class can be able to call the private functions of inner class.
5.Inner class code has openway to all members of the outer class object that contains it.
6.If the inner class has a variable with same name then the variable of outer class's can be accessed in given syntax

Outerclassname.this.variable_name

Types of Inner class:

There are four kinds of inner classes

1.Static member classes
2 .Member classes
3 .Local classes
4 .Anonymous classes

1.Static member classes

- This inner class can be defined as the static member variable of other class.
- Static members of the outer class can access the static inner class.
- The non-static members of the outer class are not accessible to inner class.

Syntax:

```
Access_modifier class OuterClass
{
        //Code

        public static class InnerClass
        {
                //Code
        }
}
```

2.Member classes

This kind of inner class is non-static fields of outer class.

3.Local classes

This class is defined inside a Java code just similar to a local variable.
Local classes are not at all declared with an access specifier.
The life time of inner classes is constantlylimited to the block in which they are declared.
The local classes are totally hidden from the outside world.

Syntax:

```
Access_modifier class OuterClass
{
        //Code
        Access_modifierreturn_typemethodname(arguments) {
                classInnerclass
                {
                        //Code
                }
        //Code
}
```

4.Anonymous classes

Anonymous class is a local class with nospecified name.
Anonymous class is a one-shot class formed exactly where required.
The anonymous class is created in below situations

- When the class has very small body.
- Only one instance of the class is desired.
- Class is used directly after defining it.

The anonymous inner class can be able to extend the class,it can implement the interface or it can be stated in function argument.

Example:

```
class MyInnerClass implements Runnable
{
public void run()
{
System.outprmtin("Hello”);
}

class DemoClass
{
public static void main(String args[])
{
MyInnerClass my=new MyInnerClass();
Thread th=new Thread(my);
my.start();
}
}
}
```

Example2:

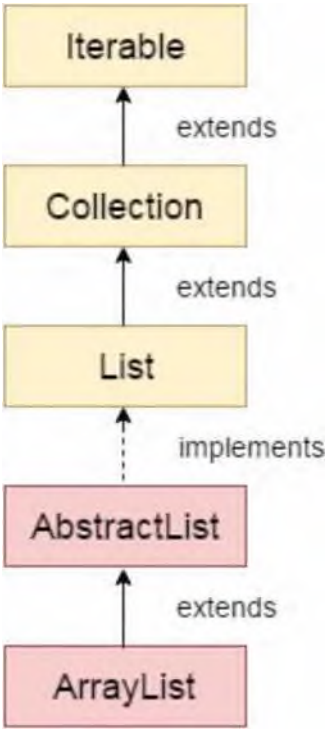
```
class Outer
{
int x=100;
void test()
{
Inner inner=new Inner();
Inner.display();
}

class Inner
{
void display()
{
System.outprintln("display x:”+x);
}
}
}

class IC
{
public static void main(String args[])
{
Outer outer=new Outer();
outer.test();
}
}
```

2.8. ARRAY LIST CLASS

Dynamic array is used in ArrayList class for storing the elements. It derivesAbstractList class and implements List interface.

Features of Java ArrayList class are:

- Java ArrayList class can hold duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non-synchronized.
- Java ArrayList allows random access since array works at the index basis.
- In Java ArrayList class, operation is slow because a lot of shifting wants to be occurred if any element is deleted from the array list.ArrayList class hierarchy is specified in below



Example:

```
import java.util.*;
class TestCollection1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();//Creating arraylist list.add("Ravi");//Adding object
in arraylist
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");
//Traversing list through Iterator
Iterator itr=list.iterator();
while(itr.hasNext()){
```

Two possible ways to iterate the elements of collection in java

There are two ways to visit collection elements:

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen visting ArrayList by Iterator. Let's see the example to go across ArrayList elements using for-each loop.

Iterating Collection through for-each loop

```
import java.util.*;
class TestCollection2{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>();
al.add("Ravi");
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");
for(String obj:al)
   System.out.println(obj);
}
}
```

```
Ravi
Vijay
Ravi
Ajay
```

User-defined class objects in Java ArrayList

Let we see another example where we are storing Student class object in array list.

```
class Student{
int rollno;
String name;
int age;
Student(int rollno,String name,int age){
this.rollno=rollno;
this.name=name;
this.age=age;
}
}
```

```
import java.util.*;
public class TestCollection3{
public static void main(String args[]){
//Creating user-defined class objects

Student s1=new Student(101,"Sonoo",23);
Student s2=new Student(102,"Ravi",21);
Student s2=new Student(103,"Hanumat",25);
//creating arraylist
ArrayList<Student> al=new ArrayList<Student>();
al.add(s1);//adding Student class object
al.add(s2);
al.add(s3);
//Getting Iterator
Iterator itr=al.iterator();
//traversing elements of ArrayList object
```

```
while(itr.hasNext()){
   Student st=(Student)itr.next();
   System.out.println(st.rollno+" "+st.name+" "+st.age);
}
}
}
```

Output

```
101 Sonoo 23
102 Ravi 21
103 Hanumat 25
```

Example of addAll(Collection c) method

```
import java.util.*;
class TestCollection4{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>();
al.add("Ravi");
al.add("Vijay");
al.add("Ajay");
ArrayList<String> al2=new ArrayList<String>();
al2.add("Sonoo");
al2.add("Hanumat");
al.addAll(al2);//adding second list in first list
Iterator itr=al.iterator();
while(itr.hasNext()){
   System.out.println(itr.next());
}
}
}
```

Output

```
Ravi
Vijay
Ajay
Sonoo
Hanumat
```

The other methods of ArrayList are also used such as removeAll() methodretainAll() method.

```
   System.out.println(itr.next());

   }

}

}
```

Output:

```
Ravi
    Vijay
    Ravi
    Ajay
```

2.9.STRINGS

String is a collection of characters.
Easiest way to represent a collection of characters in java is by using a character array.

```
char chararray[]=new char[4];
chararray[0]='J';
chararray[1]='a';
chararray[2]='v';
chararray[3]='a';
```

In java,strings are class objects and implemented using two classes that is strings and stringbuffer.
A java string is an instantiated object of the string class.

```
String stringname;
Stringname=new String("string");
```

Example

```
String fname;
firstname=new String("CSE");
```

String Arrays

```
String items[]=new String[3];
```

String Methods

The string class defines a number of methods that allow us to accomplish a variety of string manipulation task.

String Methods	**Task Performed**
s2=s1.toLowercase;	Converts the string si to all lowercase
s2=s1.toUppercase;	To all uppercase
s2=s1.replace('x','y');	Replaces all appearances of x with y
s2=s1.trim();	Remove white spaces at the beginning and end of string
s1.equals(s2)	Return true if si is equal to s2
s1.length()	Gives the length of si
sl.charAt(n)	Gives n^{th} character of si
s1.CompareTo(s2)	Returns negative if si<s2,positive if si>s2 & zero if si is equal to s2.
s1.concat(s2)	Concatenate si and s2
sl.substring(n)	Gives substring starting from n^{th} character
s1.substring(n,m)	Gives substring starting from n^{th} character up to m^{th}.
sl.indexOf('x')	Gives the position of the first occurrence of 'x' in the string si.
s1.indexOf('x',n)	Gives the position of 'x' that occurs after n^{th} position in the string si.
String.value O f(variable)	Converts the parameter value to string representation.

Example Program

```
class Stringordering
{
static stringname [] ={"Madrs","Delhi","Calcutta","Bombay"};
```

```
public static void main(String args[]) {
int size=name.length;
String temp=null;
for(int i=0;i<size;i++)
{
for(j=i+1;j<size;j++)
{
if(name[j].compareTo(name[i])<0)
{
temp=name[i];
name[i]=name[j];
name[j]=temp;
}
}
}
for (int i=0;i<size;i++)
{
System.out.println(name[i]);
}
} }
```

Output:
Bombay
Calcutta
Delhi
Madras

String Buffer Class:

String buffer creates strings of flexible length that can be modified

Method	**Task**
s1.charAt(n,'x')	Modifies the nth character to x
s1.append(s2)	Appends the string s2 to si at the end
s1.insert(n,s2)	Inserts the string s2 at the position n of the string si
sl.setLength(n)	Sets the length of the string si to n.

Manipulation of strings: Example

```
class Stringmanipulation
{
public static void main(String args[])
{
StringBuffer str=new StringBuffer("Object Language");
System.outprintin("Original String: "+str);
System.outprintln("Length of string"+str.length());
for(int i=0i<str.length();i++)
{
int p=i+1;
System.outprintln("character aat position"+p"is"+str.charAt());
}
String astringnew String(str.toString());
int pos=astring.indexOf("language");
```

```
str.insert(pos,"oriented");
System.outprintln("Modified String"+str);
str.setcharAt(6,'-');
System.outprintin("String now"+str);
str.append("improved security");
System.outprintln("appended string"+str);
}
}
```

Output
Original String: Object Language
Character at Position : 1 is o

Modified String : Object Oriented Language
String now:Object -oriented language
Appended String : Object-Oriented language improves security

2 MARK QUESTIONS AND ANSWERS

1. Define Inheritance. May/June 2012
Inheritance can be defined as the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.The resulting classes are known as derived classes, subclasses, or child classes. Older class is known as super class.

2. What are the conditions to be satisfied while declaring abstract classes
Java Abstract classes are used to declare common characteristics of subclasses. An abstract class cannot be instantiated. It can only be used as a superclass for other classes that extend the abstract class. Abstractclasses are declared with the abstract keyword. Abstract classes are used to provide a template or design for concrete subclasses down the inheritance tree.

3. You can create an abstract class that contains only abstract methods. On the other hand, you can create an interface that declares the same methods. So can you use abstract classes instead of interfaces?
Sometimes. But your class may be a descendent of another class and in this case the interface is your only option

4. Explain the concept of Polymorphism.
Polymorphism means when an entity behaves differently depending upon the context its being used. Moreover In other words Polymorphism is the capability of an action or method to do different things based on the object that it is acting upon. Means polymorphism allows you define one interface and have multiple implementations. That being one of the basic principles of object oriented programming.

5. Explain about Virtual methods.
A child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

6. What is Static Binding?
Connecting a method call(i.e. Function Call) to a method body(i.e. Function) is called binding.

When binding is performed before the program is run (by the compiler and linker, if there is one), it's called early binding or static Binding.

7. What is Dynamic binding?

The runtime system [JVM]during runtime determines the appropriate method call based on the class of the object. This feature is called as Polymorphism. All the methods in java are dynamically resolved. This cannot be determined by the Compiler.

8. What is an Abstract classes and Methods?

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

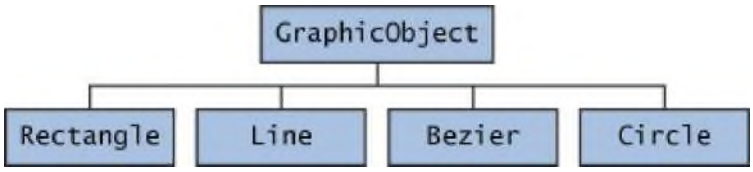
abstract void moveTo(double deltaX, double deltaY);

If a class includes abstract methods, the class itself *must* be declared abstract, as in:

```
public abstract class GraphicObject {
// declare fields
// declare non-abstract methods abstract void draw();
}
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, the subclass must also be declared abstract.

9. With example Explain Abstract Classes?



First, you declare an abstract class, GraphicObject, to provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method. GraphicObject also declares abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways.

10. When an Abstract Class Implements an Interface?

It was noted that a class that implements an interface must implement *all* of the interface's methods. It is possible, however, to define a class that does not implement all of the interface methods, provided that the class is declared to be abstract. For example, abstract class X implements Y {

```
// implements all but one method of Y
}

class XX extends X {
// implements the remaining method in Y
}
```

In this case, class X must be abstract because it does not fully implement Y, but class XX does, in fact, implement Y.

11. What is Object Class?

The Object class defines the basic state and behavior that all objects must have, such as the ability to compare oneself to another object, to convert to a string, to wait on a condition variable, to notify other objects that a condition variable has changed, and to return the object's class.

12. What is an interface (Nov/Dec 2011)

An interface is a collection of method definitions (without implementations) and constant values. In Java, an interface is a reference data type and, as such, can be used in many of the same places where a type can be used (such as in method arguments and variable declarations)

13. What are the uses of Interfaces?(Nov/Dec2010)

- Capturing similarities between unrelated classes without forcing a class relationship.
- Declaring methods that one or more classes are expected to implement.
- Revealing an object's programming interface without revealing its class. (Objects such as these are called anonymous objects and can be useful when shipping a package of classes to other developers.)

14. Why Interfaces Do not Provide Multiple Inheritances?

- You cannot inherit variables from an interface.
- You cannot inherit method implementations from an interface.
- The interface hierarchy is independent of a class hierarchy--classes that implement the same interface may or may not be related through the class hierarchy. This is not true for multiple inheritances.

15. What is meant by Reflection API?

Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine. This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of the language.

16. What are the uses of Reflection?

Extensibility Features

An application may make use of external, user-defined classes by creating instances of extensibility objects using their fully-qualified names.

Class Browsers and Visual Development Environments

A class browser needs to be able to enumerate the members of classes. Visual development environments can benefit from making use of type information available in reflection to aid the developer in writing correct code.

Debuggers and Test Tools

Debuggers need to be able to examine private members on classes. Test harnesses can make use of reflection to systematically call a discoverable set APIs defined on a class, to insure a high level of code coverage in a test suite.

17. What are the Drawbacks of Reflection?

Reflection is powerful, but should not be used indiscriminately. If it is possible to perform an operation without using reflection, then it is preferable to avoid using it. The following concerns should be kept in mind when accessing code via reflection.

- **Performance Overhead**
- **Security Restrictions**
- **Exposure of Internals**

18. What is object cloning in Java? (May/June 2013)

Objects in Java are referred using reference types, and there is no direct way to copy the contents of an object into a new object. The assignment of one reference to another merely creates another reference to the same object. Therefore, a special clone() method exists for all reference types in order to provide a standard mechanism for an object to make a copy of itself. Here are the details you need to know about

cloning Java objects.

19. Define Inner classes. (Apr/May 2011)

- An inner class is a class that is defined inside another class
- Inner class methods can access the data from the scope in which they are defined including data that would otherwise be private.
- Inner classes can be hidden from other classes in the same package.
-

20. What are the properties of proxy class ?

- Proxy classes are created on the fly in the running program.
- Once they are created they are just like any other class in the V.M.
- All proxy classes extends the class proxy.
- A proxy class as only one instant field. The innovation handles which is defined in the proxy super class.

21. What is final modifier?(May/June 2013)

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.

final Classes- A final class cannot have subclasses.

final Variables- A final variable cannot be changed once it is initialized. final Methods- A final method cannot be overridden by subclasses.

PART B -13 MARK QUESTIONS

1. Explain the concept of inheritance and its types.
2. Explain the concept of overriding with examples.
3. What is dynamic binding? Explain with example.
4. Explain the uses of reflection with examples.
5. Define an interface. Explain with example.
6. Explain the methods under “object” class and "class” class.
7. What is object cloning? Explain deep copy and shallow copy with examples.
8. Explain static nested class and inner class with examples.
9. With an example explain proxies.
10. Develop a message abstract class which contains playMessage abstract method. Write a different sub-classes like TextMessage, VoiceMessage and FaxMessage classes for to implementing the playMessage method.
11. Develop a abstract Reservation class which has Reserve abstract method. Implement the sub-classes like ReserveTrain and ReserveBus classes and implement the same.
12. Develop an Interest interface which contains simpleInterest and compInterest methods and static final field of Rate 25%. Write a class to implement those methods.
13. Develop a Library interface which has drawbook(), returnbook() (with fine), checkstatus() and reservebook() methods. All the methods tagged with public.
14. Develop an Employee class which implements the Comparable and Cloneable interfaces. Implement the sorting of persons (based on name in alphabetical). Also implement the shallow copy (for name and age) and deep copy (for DateOfJoining).
15. Explain the different methods supported in Object class with example.

Part-C 15 MARK QUESTIONS

1. Develop a static Inner class called Pair which has MinMax method for finding min and max values from the array.
2. Explain the following with examples(NOV/DEC 2010)

i. The clone able interface(8)
ii. The property interface. (7)

3 .Develop an application using inheritance and interfaces
4 .Develop a book application and apply various string operations to find particular string.
5 .With the help of real time application explain object cloning in java.