**Q.1 a. What are the differences between C++ and JAVA?**
**Answer:**

**The following are the differences between C++ and JAVA:-**

| C++ | JAVA |
|---|---|
| 1. In C++ goto, sizeof and typedef statements are included. | 1. In JAVA, goto, sizeof and typedef statements are not included. |
| 2. It uses pointers. | 2. Java doesn't use pointers. |
| 3. In C++ we may have functions that are not attached to classes. | 3. In java, all functions including main, are attached to classes. |
| 4. In C++, garbage collection is done by the programmer only. | 4. In Java, garbage collection is done automatically. |
| 5. It does not produce bytecodes that for every java program that is executed. | 5. It produces bytecodes that for every java program that is executed, which is also platform independent. |

**b. Write a JAVA program to find the first m numbers of the Fibonacci series.**
**Answer:**

```
import java.io.*
class fibo
    {
            public static void main(String args [ ])
            {
                    try
                      {
                    DataInputStream dis = new DataInputStream(System.in);
                    try
                      {
                            System.out.print("Enter the value of m:");
                            String str = dis.readLine( );
                            int m = Integer.parseint(str);
                            Int f1 = 1, f2 = 1, f;
                            System.out.print("Fibonacci series is as follows:");
                            for(int i=1; i<=m; i++)
                              {
                                    if((i==1) || (i==2))
                                            System.out.print(f1+ " ");
                                    else
                                      {
                                            f = f1 + f2;
                                            f1 = f2;
                                            f2 = f;
                                            System.out.print(f+ " ");
                                      }
                                    }
```

```
                }
                Catch(NumberFormatException e)
                 {
                        System.out.println(“wrong input data:”);
                  }
        }
        Catch(Exception e)
           {
            }
          }
        }
```

**c. What is method overloading in JAVA? Give an example to explain.**

**Answer:**

Method overloading is used when a programmer intends to create several methods that perform closely related tasks. The following example explains this:-

```
Class Sample
{
  public int abs(int a)
    {
        if ( a < 0)
            a = -a;
        return a;
    }
public float abs(float b)
    {
        if ( b < 0)
            b = -b;
        return b;
    }
public double abs(double c)
    {
        if ( c < 0)
            c = -c;
        return c;
    }
```

Thus, we have three abs () methods here. At the time of compilation, the compiler resolves that which version of the abs() must be called depending on the parameter type passed.

**d. Distinguish between interfaces and abstract classes.**

**Answer:**

Differences between interfaces and abstract classes are as follows:-

1. An abstract class is an incomplete class that requires further specialization. An interface is just a specialization or prescription for the promised behavior.
2. A class can implement several interfaces at once whereas a class can extend only one parent class.
3. An abstract class is generally used where you want to initiate a hierarchy of more specialized classes. On the other hand, an interface is used where you say-“I need to be able to call methods with these signatures in your classes”.
4. Interfaces can be used to support callbacks also.

**g. With an example show how values can be passed to applets?**

**Answer:**

The PARAM tag is used to pass values to your applet. It is placed between <APPLET> and </APPLET> tag. The PARAM tag has two arguments—NAME and VALUE. The NAME argument specifies the name of the parameter and the VALUE defines its value.

**Q.2 a. What is type casting in JAVA? Explain its various types with suitable examples.**

**Answer:**

**The process of converting one data type to another data type is known as type casting. It is of two types:-**

1. **Implicit Casting:** It means simply assigning one entity to another without any transformation guidance to the compiler. It may not work for all application scenarios.

**For e.g.**
```
int t= 100;
long h = t;   //implicit casting
```

2. **Explicit Casting: It means very specifically informing the compiler about the transformation that is expected.**

**For e.g.**
```
long h = 100.00;
 t = (int) h;  //explicit casting
```

But in the following example, the compiler throws an exception:-
```
int t = 100;
long h = t; //implicit casting
int t=100;
long h = t;  //implicit casting
t = h;
```
This will give an error. This is so because ‘t’ is int type and size of ‘t’ is less size of ‘h’. So, we cannot put long into int.

The following conversions however are allowed in java:-

**byte (8-bits) to short, int, long, float, double.**
**short (16-bits) to int, long, float, double.**
**int (32-bits) to long, float, double.**
**long (64-bits) to float, double.**

**b. Write a simple program that finds the largest of two numbers where the two numbers are read from the keyboard.**

**Answer:**

```
          import java.io.*
              class large
                  {
                       public static void main(String args [ ])
                       {
                                 try
                                   {
                                 DataInputStream din = new DataInputStream(System.in);
                                 int a,b;
                                 System.out.print("Enter your first number: ");
                                 a = Integer.parseInt(din.readLine ( ));
                                 System.out.print("Enter your second number: ")
                                 b = Integer.parseInt(din.readLine ( ));
                       if ( a > b)
                         System.out.print("First number is greater than second number");
                       else
                       System.out.print("second number is greater than first number");
         }
         Catch( Exception e)
         {

         }
 }
}
```

**Q.3 a. What is the order of calling constructors in JAVA?**

**Answer:**

When an object is created it calls the default constructor. When we create an object of the subclass, it automatically calls the super class constructor prior to the subclass constructor. For e.g.

```
          Class First
            {
                  First( )
                    {
                          System.out.println("First Constructor");
                    }
            }
Class second extends First
  {
          Second ( )
            {
                  System.out.println("Second constructor");
            }
}
```

Now when we create an object of Second class as follows:-

```
Second s = new Second ( );
```

It will invoke the constructor of the class First and then the constructor of the class Second and displays the following result:-

```
First Constructor.
Second Constructor.
```

**b. Explain different access specifiers in JAVA?**

**Answer:**

Access specifiers in JAVA are given in a tabular form below:-

| Access | Public | Protected | Default(Friendly) | Private | Private Protected |
|---|---|---|---|---|---|
| From the same class | Yes | Yes | Yes | Yes | Yes |
| From any class in same package | Yes | Yes | Yes | No | No |
| From any class outside the package | Yes | No | No | No | No |
| From ma subclass in the same package | Yes | Yes | Yes | No | Yes |
| From a subclass outside package | Yes | Yes | No | No | Yes |

**c. Explain the following with an example:-**

i. **Final data member.**
ii. **Final method.**
iii. **Final class.**
iv. **Final object.**

**Answer:**

**i. Final data member**

When we declare a data member as **final** then it means that its value cannot be changed. For e.g.

```
final float pi = 3.14f;
final int i=10;
final char status='y';
```

**ii. Final method**

When we declare any method as **final** in java then it means that it cannot be overridden in its subclasses. For e.g.

```
final void decrement ( )
{
  //…….
}
```

**iii. Final class**

A **final** class cannot be inherited at any cost. The function of a final class is just reverse of an abstract class. For e.g.

```
final class classname
 {
   //……
 }
```

**iv. Final object**

The **final** object is a constant object and it must be initialized to an object at the point of declaration. The **final** object cannot be changed to point to another object.

**Q.4 a. Why do we use import statement?**

**Answer:**

**Import statement in JAVA**

Java package can be used in a program by using import statement. Its syntax is as follows:-

```
import PackageName.*;          //importing an entire package PackageName
import PackageName.ClassName;  //importing a class ClassName from package
                                                    //PackageName
```

For example,

```
import java.awt.*;
import java.awt.Button;
```

Here, a dot separates an element. Thus, a hierarchy has been followed while creating these in-built packages. The first import statement imports all the classes or interfaces of the awt package. The second statement imports Button class from the awt package. After importing the above package, we can create an object of the Button class as :-

```
Button b = new Button("yes");
```

**b. Write a java program to show how interfaces can be extended in Java?**

**Answer:**

Like classes, interfaces can also be extended. It means that an interface can be subinterfaced from other interfaces. The new subinterface will inherit all the members of the super interface. The syntax is as follows:-

```
interface SubInterfaceName extends SuperInterfaceName
  {
        //……..
 }
```

Let us give an example now.

```
interface first
{
        void show1( );
}
interface second extends first
  {
      void show2 ( );
}
class third implements second
  {
     public void show1 ( )        //implementing first interface method declaration.
          {        System.out.println("First interface method.");
          }

   public void show2 ( )          //implementing second interface method declaration.
          {        System.out.println("Second interface method.");
          }
    }
Class ExtendInterfaceDemo
  {
     public static void main(String args[ ])
       {
         third t = new third ( );
          t.show1 ( );
          t.show2 ( );
       }
  }
```

OUTPUT:

```
        First interface method.
        Second interface method.
```

**Q.5 a. List some common java exceptions and explain.**

**Answer:**

**Common JAVA exceptions are as follows:-**

1. **The ArithmeticException:** This exception is thrown when an exceptional arithmetic condition such as division by 0, encounters.
2. **The ArrayOutOfBoundsException:** This exception is thrown when an attempt is made to access an array element beyond the arry index.
3. **The NullPointerException:** This exception is thrown when you try to use null where an object is required. For example, if you try to call a method on an object which is just declared, not created, it throws this exception.
4. **The NumberFormatException:** This exception is thrown when you attempt to perform an invalid conversion of a string to a number format.
5. **The StringIndexOutOfBoundsException:** This exception is thrown when you try to attempt to an index outside the bounds of a string.

6. **The InterruptedException:** This exception is thrown when one thread interrupts another thread.

**b. Give an example of a program in java to show how you can create your own exception classes?**

**Answer:**

To create your own exceptions we extend the Exception class. The extended class contains constructors, data members and methods like any other class. The throw and throws keywords are used when implementing user-defined exceptions.

```
// own exceptions creation program
Class MyException extends Exception
  {
      MyException ( )
       {
         }

      MyException (String str)
       {
                super(str);
         }
}

Class UserException
  {
      public static void main(String args[ ])
      {
          try
                {
                        show( );
                }
           catch(MyException e)
                {
                        System.out.println(e.getMessage( ));
                }
         try
           {
                display ( );
           }
         catch (MyException e)
          {
                System.out.println("e.getMessage( ));
          }
      }
static void show ( ) throws MyException
 {
```

```
                System.out.println(“Throwing my own exception from show()”);
                throw new MyException ( );
}

static void display ( )throws MyException
 {
        System.out.println(“throwing my own exception…”);
        throw new MyException(“Originated in display ( )”);
}
}
```

**Q.6 a. What do the following methods do:-**
**i) wait( ).**
**ii) notify ( ).**
**iii) notifyAll ( ).**

**Answer:**

**i) wait ( )**
The wait ( ) method makes a thread in waiting state.
**ii) notify ( )**
The notify () wakes up the first thread that called wait() on the same object.
**iii) notifyAll ( )**
This method wakes up all threads that are waiting on the same object.

**b. Differentiate between Applications and Applets.**

**Answer:**
Let us tabulate the differences between the two:-

| Applications | Applets |
|---|---|
| 1. Application programs use main( ) method for initiating the execution of the code. | 1. Applets donot use any main ( ) method. |
| 2. These programs can run independently. | 2. Applets donot run independently. They run embedded into a web page on a web browser using HTML tags. |
| 3. These can read from or write to the files on a local computer. | 3. These cannot read from or write to the files on a local computer. |
| 4. These can run some other program from the local computer. | 4. These cannot run some other program from the local computer. |
| 5. Java language can use libraries of other languages like C and C++ using native method. | 5. Applets are restricted from using libraries from other languages. |

**Q.7 a. Write a JAVA program to create three simple sliders—plain slider, slider with tick marks and a third one with both tick marks and sliders.**

**Answer:**

```
//slider program in java
Import javax.swing.*;
```

```
Public class JSliders extends JFrame{
  Public static void main(String[] args){
     New JSliders( );
}
Public JSliders( ){
   Super("Using JSlider");
  //comment out next line for java look-and-feel
  WindowUtilities.setNativeLookAndFeel( );
  addWindowListner(new ExitListner());
  Container content = getContentPane();
  Content.setBackground(Color.white);

JSlider slider1 = new JSlider();
Slider1.setBorder(BorderFactory.createTitledBorder("JSlider without Tick Marks"));
Content.add(slider1, BorderLayout.North);

JSlider slider2 = new JSlider();
slider2.setBorder(BorderFactory.createTitledBorder("JSlider with Tick Marks"));
slider2.setMajorTickSpacing(20);
slider2.setMajorTickSpacing(5);
slider2.setPaintTicks(true);
content.add(slider2, Borderlayout.CENTER);

JSlider slider3 = new JSlider();
Slider3.setBorder(BorderFactory.createTitledBorder("JSlider with Tick Marks"));
Slider3.setMajorTickSpacing(20);
Slider3.setMajorTickSpacing(5);
Slider3.setPaintTicks(true);
Slider3.setPaintLabels(true);
content.add(slider3, Borderlayout.SOUTH);

pack( );
setVisible(true);
}
}
```

**b. Show with an example program how inheritance is supported by beans.**

**Answer:**

```
//Inheritance by beans
public class Subject1BeanInfo extends SimpleBeanInfo {
  private final static Class myClass = Subject1.class;
     public PropertyDescriptor[] getPropertyDescriptions ()
        {
           try{
                 PropertyDescriptor maximumStudents = new
PropertyDescriptor("maximumStudents", myClass);
```

```
        PropertyDescriptor numberOfStudents = new
        PropertyDescriptor(“numberOfStudents”, myClass);
        PropertyDescriptor foreground = new PropertyDescriptor(“foreground”,
myClass);

        PropertyDescriptor background = new PropertyDescriptor(“background”,
myClass);
        PropertyDescriptor name= new PropertyDescriptor(“name”, myClass);

        foreground.setHidden(true);
        background.setHidden(true);
        name.setHidden(true);
        PropertyDescriptor[] properties = {maximumStudents,
numberOfStudents,foreground, background,name};
        return properties;
        }

        catch(IntrospectionException e) {
           e.printStackTrace();
        }
        return null;
        }
    }
```

**Text Books**

1. Cay Horstmann-Computing Concepts with Java 2 Essentials, John Wiley, 3rd Edition

2. E. Balagurusamy- Programming with Java: A Primer, 3rd Edition, 2006, Tata McGraw-Hill