**2015, Q1. What is the difference between Boolean & operator and && operator?**

**Ans.** Consider two Boolean operands A and B.

A & B means both the operands A as well as B are evaluated and then ‘&’ is applied to them. The logical Boolean operator & operate on booleanvalues in the same way that it operates on the bits of an integer.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A&B** |
| TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| FALSE | FALSE | FALSE |

Eg. 1: int a = 4;

int b =7;

System.out.println(a & b); //prints 4

//meaning in a 16 bit system :

//a = 0000 0000 0000 0100

//b = 0000 0000 0000 0111

//a & b = 0000 0000 0000 0100

Eg 2: boolean A=TRUE,B=FALSE ;

System.out.println(" A&B = " ,A & B);

A && B means A is evaluated first. If A is true then B is evaluated and then the operand is applied to them. If A is false, evaluation of B is skipped. This is known as shortcircuiting and may be considered an optimisation. This is especially useful in guarding against nullness.

It is secondary version of the Boolean AND.

Eg 1: if (b != 0 && a / b > 10)

Since the short-circuit form of AND (&&) is used, there is no risk of causing a run-time exception( dividing by zero) when b is zero. If this line of code were written using &, both sides would have to be evaluated, causing a run-time exception when bis zero.

In brief,

|  |  |
| --- | --- |
| **Operation** | **Meaning** |
| A & B | Logical AND |
| A && B | Logical AND with short-circuiting |

*Submitted by* *NEENU MARY MARGRET, Rollno: 21*

**2015, Q2. State whether each of the following is true or false.**

**(a)IP addresses from 224. 0. 0. 0 to 239. 255. 255. 255 are reserved for multicast.**

**(b)For security reasons many web browsers allow java applets to do file processing only on machine on which they execute.**

**(c)Datagram packet transmission over a network is reliable packets are guranted to arrive in sequence.**

**Ans.**

(a)True, the range of addresses between 224.0.0.0 and 224.0.0.255, inclusive,is reserved for the use of routing protocols and other low-level topology discovery or maintenance protocols, such as gateway discovery and group membership reporting.

(b)True,Applets that are not signed are restricted to the security sandbox, and run only if the user accepts the applet. Applets that are signed by a certificate from a recognized certificate authority can either run only in the sandbox, or can request permission to run outside the sandbox. In either case, the user must accept the applet's security certificate, otherwise the applet is blocked from running

(c)False,the individual packets which form a data stream may follow different paths between the source and the destination. As a result, the packets may arrive at the destination out of order. When this occurs, the packets will have to be reassembled to form the original message.

*Submitted by* *NEENU NELSON, Roll no: 22*

**2015, Q3. Which is the default layout manager for an applet?List the different layout managers.**

**Ans.** The default layout manager for an applet is Flow Layout. It is used to arrange the components in a line, one after another.

Fields of FlowLayout class:

|  |
| --- |
| 1. public static final int LEFT 2. public static final int RIGHT 3. public static final int CENTER 4. public static final int LEADING 5. public static final int TRAILING |

Constructors of FlowLayout class:

|  |
| --- |
| 1. FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap. 2. FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap. 3. FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap. |

Different layout managers:

1. BorderLayout
2. FlowLayout
3. GridLayout
4. CardLayout
5. GridBagLayout
6. BoxLayout
7. GroupLayout
8. ScrollPaneLayout
9. SpringLayout

*Submitted By NEIL MATHEW MALIACKAL, Rollno: 23*

**2015, Q4. How this() and super( ) clauses differ ?**

**Ans.**

The this and super are two special keywords in Java, which is used to represent current instance of a class and its super class. The main difference between this and super in Java is that this represents current instance of a class, while super represent current instance of the parent class. You can use this() to call no argument constructor of same class, while super() to call no argument or default constructor of parent class. By the way, call is not limited to only no argument constructor, you can call any constructor by passing appropriate parameters.

Another use of this and super in Java is for *accessing instance variables of a class and it's parent*. By the way, you can also access them even without prefixing super and this, if they are not ambiguous in current block e.g. if there is no [local variable](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) of same name exists in that block, but in case of ambiguity they provide explicit reference and are more readable. 

Similarities between this and super clause:

1) Both this and super are non static and can not be used in static context, which means you can not use this and super keyword inside [main method in Java](http://java67.blogspot.com/2012/08/what-is-main-method-in-java-why-main-is.html). Failing to do so will result in compiler error "non static variable this can not be referenced from static context". Same is true for using super keyword inside main method.

public static void main(String args[]) {

     // compiler error - non static variable can not be referenced from static context

     System.out.println(this.name); }

2) Both this and super can be used in constructor chaining to call another constructor e.g. this() and super() respectively calls **no argument constructor** of child and parent class.

As shown in this example, we are first forwarding call from no argument constructor of B, to a constructor which accepts one String argument, which further call to super(""), a call to super class, one argument constructor.

class *A*{     *A*(){ System.out.**println**("A'*s* *no* *argument* *constructor*");}

    A(String args){ System.out.**println**("A'*s* *one* *argument* *constructor*");}

}

 class *B* extends *A*{    *B*(){

**this**(""); // calling one arg constructor of class B

        System.out.**println**("B'*s* *no* *argument* *constructor*");

    }   B(String args){

**super**(""); // calling one argument constructor of class A

        System.out.**println**("B'*s* *one* *argument* *constructor*");

    }

}

 // Test Class and Output

public class *Test* {     public static void **main**(String *args*[]) {

       B b **=** **new** B();

}    }

 Output**:**

A's one argument constructor

B's one argument constructor

B's no argument constructor

3) If used them inside constructor than this and super must be first statement, otherwise compiler will complain. Which means you can not call this() and super() from same constructor.

Difference between this and super clause:

1) this is used in context of the class you are working on, while super is used to refer current instance of parent class.

2) Every constructor by default calls super(), which is a call to no argument constructor of parent class, but you can still call another constructor either explicitly by using this() or super().

*Submitted by NIGIL BENNO JOSEPH, Rollno: 24*  
  
**2015, Q5. What is JDBC-ODBC bridge driver?**

**Ans.** JDBC driver is a software component that enables java application to interact with the databse.There are four types of drivers:

1.JDBC-ODBC bridge driver.

2. Native API driver.(partially java driver)

3. Network Protocol driver(fully java driver).

4.Thin driver(fully java driver).

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

**Databas**e

**Vendor Database Library**

**Java**

**Application**

Fig:JDBC-ODBC Bridge Driver

Advantages:

* Easy to use.
* Can be easily connected to any database.

Disadvantages:

* Performance degraded because JDBC method call is converted into the ODBC function calls.
* The ODBC driver needs to be installed on the client machine.

*Submitted by NISHANT THOMAS, Rollno: 25*

**2015, Q6. Define encapsulation and information hiding in OOP.**

**Ans.**

Encapsulation :

Encapsulation is the mechanism that binds together code and the data it manipulates , and keeps both safe from outside inferences and misuse .It is a protective wrapper the prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper access to the code and data inside the wrapper is tightly controlled through a well-defined interface .

Example: consider a car. We don’t know the internal working of a car but within a car there are subsystems like steering, brake, clutch etc which are encapsulated.

Each class in java itself is an example of encapsulation. Each method or variable in a class may be private or public. The public interface of a class represents everything that external users of a class need to know. The private variables can only be accessed by code that is a member of class. Therefore, any other code that is not a member of the class cannot access a private method or variable. Thus encapsulation is said to be providing “access control” through which we can control which part of program can access the members of any class and thus prevent misuse.

Encapsulation gives maintainability, flexibility and extensibility to our code.

Information hiding:

Information hiding is the primary criteria of system modularization and should be concerned with hiding the critical decisions of OOP designing. Information hiding isolates the end users from the requirement of intimating knowledge of the design for the usage of a module.It is one of the most important principles of OOP inspired from real life which says that all information should not be accessible to all persons. Private information should only be accessible to its owner. That is Showing only those details to the outside world which are necessary for the outside world and hiding all other details from the outside world.

Example: An email server may have account information of millions of people but it will share only our account information with us.If we request to send anyone else accounts information, our request will be refused.

In object oriented programming approach we have objects with their attributes and behaviors that are hidden from other classes, so we can say that object oriented programming follows the principle of information hiding.Information Hiding is achieved in Object Oriented Programming using the following principles,

* All information related to an object is stored within the object
* It is hidden from the outside world
* It can only be manipulated by the object itself

The main advantage of information hiding is that it simplifies the object oriented model.

*Submitted by PAYAL MARIYA CHETTUPUZHAKKARAN, Rollno: 26*

**2015, Q7. How overloading is different from overriding. Illustrate with examples.**

**Ans.** A list of differences between method overloading and method overriding are given below:

|  |  |
| --- | --- |
| 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 class. | Method overriding occurs in two classes that have IS-A (inheritance) relationship. |
| In case of method overloading, parameter must be different. | In case of method overriding, parameter must be same. |
| Method overloading is the example of compile time polymorphism. | Method overriding is the example of run time polymorphism. |
| In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter. | Return type must be same or covariant in method overriding. |

Method Overloading example:

class OverloadingExample{

static int add(int a,int b){

return a+b;

}

static int add(int a,int b,int c){

return a+b+c;

} }

Method Overriding example:

class Animal{

void eat(){

System.out.println("eating...");

} }

class Dog extends Animal{

void eat(){

System.out.println("eating bread...");

} }

*Submitted by PETER NELSON, Roll No: 27*

**2015, Q8. How Swing components are different from AWT components? Write advantages of Swing components.**

**Ans**:

|  |  |
| --- | --- |
| AWT | Swing |
| AWT stands for Abstract Window Toolkit. | Swing is a part of Java Foundation Class (JFC). |
| AWT components are heavy weight. | Swing components are light weight. |
| AWT components are platform dependent so there look and feel changes according to OS. | Swing components are platform independent so there look and feel remains constant. |
| AWT components are not very good in look and feel as compared to Swing components. See the button in below image, its look is not good as button created using Swing. | Swing components are better in look and feel as compared to AWT. See the button in below image, its look is better than button created using AWT. |

There are a few other advantages to Swing over AWT:

* Swing provides both additional components and added functionality to AWT-replacement components
* Swing components can change their appearance based on the current "look and feel" library that's being used. You can use the same look and feel as the platform you're on, or use a different look and feel
* Swing components follow the Model-View-Controller paradigm (MVC), and thus can provide a much more flexible UI.
* Swing provides "extras" for components, such as:
  + Icons on many components
  + Decorative borders for components
  + Tooltips for components
* Swing components are lightweight (less resource intensive than AWT)
* Swing provides built-in double buffering
* Swing provides paint debugging support for when you build your own components

*Submitted by PHILIP JOSEPH, Rollno: 28*

**2015, Q9. Write the uses of wait(),notify() and notifyAll() methods.**

**Ans.**

wait(),notify() and notifyAll() methods allow threads to communicate about the locked status of a resource.

wait() : It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify(). The wait() method releases the lock prior to waiting and reacquires the lock prior to returning from the wait() method. The wait() method is actually tightly integrated with the synchronization lock, using a feature not available directly from the synchronization mechanism.

General syntax for calling wait() method is like this:

|  |
| --- |
| synchronized( lockObject )  { while( ! condition )       {           lockObject.wait();       }        //take the action here;  } |

notify() : It wakes up one single thread that called wait() on the same object. It should be noted that calling notify() does not actually give up a lock on a resource. It tells a waiting thread that that thread can wake up. However, the lock is not actually given up until the notifier’s synchronized block has completed. So, if a notifier calls notify() on a resource but the notifier still needs to perform 10 seconds of actions on the resource within its synchronized block, the thread that had been waiting will need to wait at least another additional 10 seconds for the notifier to release the lock on the object, even though notify() had been called. Though if the notify() method is called when no other thread is waiting, notify() simply returns and the notification is lost.

General syntax for calling notify() method is like this:

|  |
| --- |
| synchronized(lockObject)  { //establish\_the\_condition;      lockObject.notify();        //any additional code if needed  } |

notifyAll() : It wakes up all the threads that called wait() on the same object. The highest priority thread will run first in most of the situation, though not guaranteed. Other things are same as notify() method above. All of the waiting threads wake up, but they still have to reacquire the object lock. So the threads do not run in parallel; they must each wait for the object lock to be freed. Thus, only one thread can run at a time, and only after the thread that called the notifyAll() method releases its lock.

General syntax for calling notify() method is like this:

|  |
| --- |
| synchronized(lockObject)  { establish\_the\_condition;    lockObject.notifyAll();  } |

In general, a thread that uses the wait() method confirms that a condition does not exist (typically by checking a variable) and then calls the wait() method. When another thread establishes the condition (typically by setting the same variable), it calls the notify() method. The wait-and-notify mechanism does not specify what the specific condition/ variable value is. It is on developer’s hand to specify the condition to be checked before calling wait() or notify().

**Eg:-Producer Consumer problem** using wait() and notify() methods.

class Producer implements Runnable

{

   private final List<Integer> taskQueue;

   private final int MAX\_CAPACITY;

   public Producer(List<Integer> sharedQueue, int size)

   {

      this.taskQueue = sharedQueue;

      this.MAX\_CAPACITY = size;

   }

   @Override

   public void run()

   {

      int counter = 0;

      while (true)

      {

         try

         {

            produce(counter++);

         }

         catch (InterruptedException ex)

         {

            ex.printStackTrace();

         }

      }

   }

   private void produce(int i) throws InterruptedException

   {

      synchronized (taskQueue)

      {

         while (taskQueue.size() == MAX\_CAPACITY)

         {

            System.out.println("Queue is full " + Thread.currentThread().getName() + " is waiting , size: " + taskQueue.size());

            taskQueue.wait();

         }

         Thread.sleep(1000);

         taskQueue.add(i);

         System.out.println("Produced: " + i);

         taskQueue.notifyAll();

      }

   }

}

class Consumer implements Runnable

{

   private final List<Integer> taskQueue;

   public Consumer(List<Integer> sharedQueue)

   {

      this.taskQueue = sharedQueue;

   }

   @Override

   public void run()

   {

      while (true)

      {

         try

         {

            consume();

         } catch (InterruptedException ex)

         {

            ex.printStackTrace();

         }

      }

   }

   private void consume() throws InterruptedException

   {

      synchronized (taskQueue)

      {

         while (taskQueue.isEmpty())

         {

            System.out.println("Queue is empty " + Thread.currentThread().getName() + " is waiting , size: " + taskQueue.size());

            taskQueue.wait();

         }

         Thread.sleep(1000);

         int i = (Integer) taskQueue.remove(0);

         System.out.println("Consumed: " + i);

         taskQueue.notifyAll();

      }

   }

}

public class ProducerConsumer

{

   public static void main(String[] args)

   {

      List<Integer> taskQueue = new ArrayList<Integer>();

      int MAX\_CAPACITY = 5;

      Thread tProducer = new Thread(new Producer(taskQueue, MAX\_CAPACITY), "Producer");

      Thread tConsumer = new Thread(new Consumer(taskQueue), "Consumer");

      tProducer.start();

      tConsumer.start();

   }

}

*Submitted by POORNA C B, Roll no: 29*

**2015, Q10. Write a Java program to read the contents of a text file and display it on the screen.**

**Ans:**

import java.io.BufferedReader;

import java.io.FileReader;

public class readtext

{

    public static void main (String args[])

{

String line;

            FileReader filereader = new FileReader("Sample.txt");

            BufferedReader bufferedReader = new BufferedReader(filereader);

            while ((line = bufferedReader.readLine())!= null)

{

                System.out.println(line);

            }

            filereader.close();

    }

}

*Submitted by PRANAV RAJ , Rollno: 30.*

**2015, Q11. (a) How unsigned right shift ( >>> ) is different from right shift ( >> ) operator? (b). What is a constructor? What do you mean by overloading a constructor? Explain with suitable example.**

**Ans.**

**(a)** The Right Shift (>>) :

The right shift operator, >> shifts all of the bits in a value to the right a specified number of times. Its general form is shown below:

Value >> num

Here, num specifies the number of positions to right-shift the value in Value. That is, the >> moves all of the bits in the specified value to the right the number of bit positons specified by num.The following code fragment shifts the value 32 to the right by two positions, resulting in ‘a’ being set to 8:

int a = 32 ; a = a >> 2 ; // a now contains 8

When a value has bits that are “shifted off” , those bits are lost. For example, the next code fragment shifts the value 35 to the right two postions, which causes the two low-order bits to be lost, resulting in ‘a’ being set to 8.

int a = 35; a = a >> 2 ; // a still contains 8

Each time you shift a value to the right, it divides that value by two – and discards any remainder. You can take advantage of this fro high-performance integer division by 2.

When you are shifting right, the top (leftmost) bits exposed by the right shift are filled in with the previous contents of the top bit. This is called sign extension and serves to preserve the sign of negative numbers when you shift them right. For example, -8 >> 1 is -4 , which in binary, is

11111000 -8

>>1

11111100 -4

It is interesting to note that if you shift -1 right, the result always remains -1 , since sign extension keeps bringing in more ones in the high-order bits.

The Unisigned Right Shift( >>>) :

The right shift automatically fills the higher order bit with its previous contents each time a shift occurs. This preserves the sign of the value. However, sometimes this is undesirable. For example, if you are shifting something that does not represent a numeric value, you may not want sign extension to take place. This situation is common when you are with pixel-based values and graphics. In these cases you will generally want to shift a zero into the high-order bit no matter what its initial value was. This is known as an unsigned shift. To accomplish this, you will use Java’s unsigned shift-right operator , >>> , which always shifts zeros into the high-order bit.

The following code fragment demonstrates the >>> . Here, a is set to -1 , which sets all 32 bits to 1 in binary. This value is then shifted right 24bits, filling the top 24bits with zeros, ignoring normal sign extension. This sets a to 255.

int a = 1 ;

a = a >>> 24 ;

Here is the same operation in binary form to further illustrate what is happening:

11111111 11111111 11111111 11111111 -1 in binary as an int

>>>24

00000000 00000000 00000000 11111111 255 in binary as an int

**(b)** Java allows objects to initialise themselves when they are created. This automatic initialization is performed through the use of a constructor. A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes. Constructors looklittle strange because they have no return type, not even void. This is because rthe implicit return type of a class’ constructor is the class type itself. It is the constructor’s job to initialize the internal state of an object so that the code creating an instance will have a fully initialised, usable object immediately.

Let us see an example for a constructor :

class Box{

double width ;

double height;

double length;

Box(){

system.out.println(“Constructing Box”) ;

width = 10 ;

height = 10 ;

length = 10 ;

}

double volume() {

return width \* height \* length

}

}

class BoxDemo {

public static void main(String args[]) {

//declare, allocate and initialise Box objects

Box mybox1 = new Box() ;

Box mybox2 = new Box() ;

double vol ;

//get volume of first box

vol = mybox1.volume() ;

system.out.println(“Volume is “ + vol) ;

//get volume of second box

vol = mybox2.volume() ;

system.out.println(“Volume is “ + vol) ;

}

}

Overloading Constructors :

In addition to overloading norma methods, you can also overload constructor methods. In fact, for most real-world classes that you create, overloaded constructors will be the norm, not the exception.

Example :

class Box{

double width ;

}

double height ;

double length ;

//This is the constructor for Box.

Box(double w , double h, double l){

width = w ;

height = h ;

length = l ;

}

//compute and return volume

double volume(){

return width \* height \* length ;

}

}

As you can see, the Box() constructor requires 3 parameters. This means that all declarations of Box objects must pass three arguments to the Box() constructor. eg: the following statement is currently invalid

Box ob = new Box() ;

Since Box() requires 3 arguments, it’s an error to call it without them. This raises some important questions. What if you simply wanted a box and did not care ( or know) what its initial dimensions were? Or, what if you want to be able to initialize a cube by specifying only one value that would be used for all 3 dimensions? As the Box class is currently written, these other options are not available to you.

Fortunately, the solutiom to these problmes is quite easy : simply overload the Box constructor so that it handles the situations just described. Here is a program that contains an improved version of Box that does just that:

/\*Here, Box defines 3 constructors to initialise the dimensions of a box in various ways

\*/

class Box{

double width ;

double height;

double length ;

//constructor used when all dimensions specified

Box(double w , double h , double l){

width = w ;

height = h ;

length = l;

}

//constructor used when no dimensions specified

Box() {

width = -1 ; // use -1 to indicate

height = -1 // an uninitialized

length = -1 //box

}

//constructor used when cube is created

Box(double l){

width = height = length = l ;

}

double volume(){ //compute and return volume

return width\*height\*length ;

}

}

class OverloadCons {

public static void main(String args[]) {

//create boxes using the various constructors

Box mybox1 = new Box(10,20,15) ;

Box mybox2 = new Box() ;

Box mybox3 = new Box(7) ;

double vol ;

//get volume of first box

vol = mybox1.volume() ;

System.out.println(“Volume of mybox1 is “ + vol) ;

//get volume of second box

vol = mybox2.volume() ;

System.out.println(“Volume of mybox2 is “ + vol) ;

//get volume of third box

vol = mybox3.volume() ;

System.out.println(“Volume of mybox3 is “ + vol) ;

}

}

The output produced by this program is shown here:

Volume of mybox1 is 3000.0

Volume of mybox2 is -1.0

Volume of mybox3 is 343.0

*Submitted by PRANOV SARATH, Rollno: 31*

**2015 Q12:-** **Write a java program to find biggest element among the set of elements stored in a 2 dimensional array.**

**Ans.**

import java.util.Scanner;

public class largest{

public static void main(String[] args){

int a[][] = new int[10][10];

Scanner scanner = new Scanner(System.in);

System.out.println("Number of coloums in 2D array");

int m = scanner.nextInt();

System.out.println("Number of rows in 2D array");

int n = scanner.nextInt();

System.out.println("Enter the elements");

for (int i=0;i<n;i++){

for(int j=0;j<m;j++){

a[i][j] = scanner.nextInt();

}

}

System.out.println("The 2D array is :");

int largest = a[0][0];

for(int i=0;i<n;i++){

System.out.printf("\n");

for(int j=0;j<m;j++){

System.out.printf("%d\t",a[i][j]);

if(a[i][j] > largest){

largest = a[i][j];

}

}

}

System.out.println("The largest of 2D array is : "+largest);

}

}

*Submitted by PRIYATH SAJI, Rollno: 32*

**2015, Q13. What is a thread? Explain with an example how to define, instantiate, and run multiple threads.**

**Ans.**

Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking. Every thread in Java is created and controlled by the **java.lang.Thread class .**

Thread Creation:-

There are two ways to create thread in java;

* Implement the Runnable interface (java.lang.Runnable).
* By Extending the Thread class (java.lang.Thread).

## Create a Thread by Implementing a Runnable Interface :-

**The Runnable Interface Signature**

public interface Runnable {

void run();

One way to create a thread in java is to implement the Runnable Interface and then instantiate an object of the class. We need to override the run() method into our class which is the only method that needs to be implemented. The run() method contains the logic of the thread.

**The procedure for creating threads based on the Runnable interface is as follows:**

1. A class implements the Runnable interface, providing the run() method that will be executed by the thread. An object of this class is a Runnable object.

public void run();

2. As a second step, you will instantiate a **Thread** object using the following constructor-

Thread(Runnable threadObj, String threadName);

Where, threadObj is an instance of a class that implements the **Runnable** interface and **threadName** is the name given to the new thread.

3.The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned.

void start();

4. The thread ends when the run() method ends, either by normal completion or by throwing an uncaught exception.

Example for Runnable Interface :-

class RunnableDemo implements Runnable {

private Thread t;

private String threadName;

RunnableDemo( String name) {

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// Let the thread sleep for a while.

Thread.sleep(50);

}

}catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start () {

System.out.println("Starting " + threadName );

if (t == null) {

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

RunnableDemo R1 = new RunnableDemo( "Thread-1");

R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

}

}

This will produce the following result :

### Output

Creating Thread-1

Starting Thread-1

Creating Thread-2

Starting Thread-2

Running Thread-1

Thread: Thread-1, 4

Running Thread-2

Thread: Thread-2, 4

Thread: Thread-1, 3

Thread: Thread-2, 3

Thread: Thread-1, 2

Thread: Thread-2, 2

Thread: Thread-1, 1

Thread: Thread-2, 1

Thread Thread-1 exiting.

Thread Thread-2 exiting.

## Create a Thread by Extending a Thread Class :-

The second way to create a thread is to create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

### Step 1

You will need to override **run( )** method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of run() method – public void run()

### Step 2

Once Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method –

void start();

Example for creating a thread by extending the thread class :-

class ThreadDemo extends Thread {

private Thread t;

private String threadName;

ThreadDemo( String name) {

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// Let the thread sleep for a while.

Thread.sleep(50);

}

}catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start () {

System.out.println("Starting " + threadName );

if (t == null) {

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

ThreadDemo T1 = new ThreadDemo( "Thread-1");

T1.start();

ThreadDemo T2 = new ThreadDemo( "Thread-2");

T2.start();

}

}

This will produce the following result −

### Output

Creating Thread-1

Starting Thread-1

Creating Thread-2

Starting Thread-2

Running Thread-1

Thread: Thread-1, 4

Running Thread-2

Thread: Thread-2, 4

Thread: Thread-1, 3

Thread: Thread-2, 3

Thread: Thread-1, 2

Thread: Thread-2, 2

Thread: Thread-1, 1

Thread: Thread-2, 1

Thread Thread-1 exiting.

Thread Thread-2 exiting.

When creating threads, there are two reasons why implementing the Runnable interface may be preferable to extending the Thread class:

* Extending the Thread class means that the subclass cannot extend any other class, whereas a class implementing the Runnable interface  
  has this option.
* A class might only be interested in being runnable, and therefore, inheriting the full *overhead of the Thread class would be excessive.*

*Submitted by R.P SHEKHA MARIYAM , Roll No: 33*

**2015, Q14. Create a class called complex for performing arithmetic on complex numbers.**

**Complex numbers have the form a+bi where a is real part and b is imaginary part and I = sqrt(-1).**

**Write a program to test your class. Use floating point variables to represent the private data of the class. Provide constructor that enable an object to b initialized when it is declared. Provide no argument constructor with default values in case no initializers are provided. Provide public methods for addition, subtraction, multiplication of complex numbers. Pass objects of complex as parameters of the method.**

**Ans.**

import java.util.Scanner;

class Complex{

private float real;

private float imag;

Complex(float r1,float im1)

{ real=r1;

imag=im1;

}

Complex()

{

real=0;

imag=0;

}

public static void add(Complex obj,Complex obj2)

{

float SumReal=obj.real+obj2.real;

float SumImag=obj.imag+obj2.imag;

System.out.println("Sum = "+SumReal+"+"+SumImag+"i");

}

public static void subtract(Complex obj,Complex obj2)

{

float SubReal=obj.real-obj2.real;

float SubImag=obj.imag-obj2.imag;

if(SubImag>=0)

{

System.out.println("Difference = "+SubReal+"+"+SubImag+"i");

}

else{

System.out.println("Difference = "+SubReal+SubImag+"i");

}

}

public static void multiplication(Complex obj,Complex obj2)

{

float MultReal=(obj.real\*obj2.real)+(obj.imag\*obj2.imag)\*-1;

float MultImag=(obj.real\*obj2.imag)+(obj.imag\*obj2.real);

System.out.println("Product ="+MultReal+"+"+MultImag+"i");

}

}

class Test9 extends Complex{

public static void main(String[] args){

Scanner userInput=new Scanner(System.in);

float r1,r2,im1,im2;

System.out.println("Enter the real and imaginary part of first complex number");

r1=userInput.nextFloat();

im1=userInput.nextFloat();

Complex complexObject1=new Complex(r1,im1);

System.out.println("Enter the real and imaginary part of the second complex number");

r2=userInput.nextFloat();

im2=userInput.nextFloat();

Complex complexObject2=new Complex(r2,im2);

Complex complexObject=new Complex();

System.out.println("Choice 1.Add 2.Substract 3.Multiply");

int choice=userInput.nextInt();

switch(choice)

{

case 1: add(complexObject1,complexObject2);

break;

case 2: subtract(complexObject1,complexObject2);

break;

case 3: multiplication(complexObject1,complexObject2);

break;

}

userInput.close();

}

}

*Submitted by RAHUL R, Rollno: 34*

**2015, Q15: Write an applet so that wherever you click in the applet, display "Hello" message there.**

**Ans:**

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

/\*

<applet code="MouseEvents" width=400 height=200>

</applet>

\*/

public class MouseEvents extends Applet

{ int x=0;

int y=0;

public void init()

{

addMouseListener(new mymouselistener());

}

public void start()

{ }

public void paint(Graphics g)

{

g.drawString("Hello",x,y);

}

public class mymouselistener extends MouseAdapter

{

public void mouseClicked(MouseEvent e)

{

x = e.getX();

y = e.getY();

repaint();

}

}

}

*Submitted by RINKLE ROY, Rollno: 35*

**2015, Q16. Write a program to read in text file and print the entire content in capital**

**letters**

**Ans.**

import java.io.\*;

public class TextFile

{

public static void main (String[] args) throws IOException

{

File file1 = new File("intext.txt");

File file2 = new File("outtext.txt");

BufferedReader in = (new BufferedReader(new FileReader(file1)));

PrintWriter out = (new PrintWriter(new FileWriter(file2)));

int ch;

while ((ch = in.read()) != -1)

{

if (Character.isLowerCase(ch))

{

ch = Character.toUpperCase(ch);

}

out.write(ch);

}

in.close();

out.close();

}

}

*Submitted by RISA JOY, Rollno: 36*

**2015, Q17. Write a program to establish socket connection between a client and a server. Let the client sends two numbers to the server. The server adds the numbers and sends back the sum to the client**

**Ans.**

The program for client side is as follows

Client.java :

package Client.java;

import java.net.\*;

import java.io.\*;

import java.util.Scanner;

public class GreetingClient

{

public static void main(String [] args)

{

Scanner userInput = new Scanner(System.in);

System.out.println("Enter Server Address: ");

String serverName;

serverName = userInput.next();

System.out.println("Enter Port Number: ");

String port;

port = userInput.next();

try

{

System.out.println("Connecting to " + serverName + " on port " + port);

Socket client = new Socket(serverName, Integer.parseInt(port));

System.out.println("Just connected to " + client.getRemoteSocketAddress());

OutputStream outToServer = client.getOutputStream();

DataOutputStream out = new DataOutputStream(outToServer);

System.out.println("Enter a first number: ");

Integer x= userInput.nextInt();

System.out.println("Enter a second number: ")

Integer y= userInput.nextInt();

out.writeInt(x);

out.writeInt(y);

InputStream inFromServer = client.getInputStream();

DataInputStream in = new DataInputStream(inFromServer);

System.out.println("Server responds: " +in.readInt());

client.close();

}catch(IOException e)

{

e.printStackTrace();

}

The program for server side is as follows:

Server.java :

package Client.java;

import java.net.\*;

import java.io.\*;

public class GreetingServer extends Thread

{

private ServerSocket serverSocket;

public GreetingServer(int port) throws IOException

{

serverSocket = new ServerSocket(port);

serverSocket.setSoTimeout(100000);

}

public void run()

{

while(true)

{

try

{

System.out.println("Waiting for client on port " +

serverSocket.getLocalPort() + "...");

Socket server = serverSocket.accept();

System.out.println("Just connected to " + server.getRemoteSocketAddress());

DataInputStream in= new DataInputStream(server.getInputStream());

Integer x=in.readInt();

Integer y=in.readInt();

Integer sum = ( x + y );

DataOutputStream out = new DataOutputStream(server.getOutputStream());

System.out.println(sum);

out.writeInt(sum);

server.close();

}catch(SocketTimeoutException s)

{ System.out.println("Socket timed out!");

break;

}catch(IOException e)

{ e.printStackTrace();

break;

}

}

}

public static void main(String [] args)

{

Scanner userInput = new Scanner(System.in);

System.out.println("Please specify a port number (1~65535): ");

String port;

port = userInput.next();

try

{

Thread t = new GreetingServer(Integer.parseInt(port));

t.start();

}catch(IOException e)

{ e.printStackTrace();

}

}

}

*Submitted by* ROHAN GEORGE, Roll no: 37

**2015, Q18. Define RMI. Explain stub-skeleton model for RMI in java.**

**Ans.**

The RMI(Remote Method Invocation)

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A remote object is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

### stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

### skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.



#### Creating a Simple RMI application involves following steps

* Define a remote interface.
* Implementing remote interface.
* create and start remote application
* create and start client application

**Define a remote interface**

A remote interface specifies the methods that can be invoked remotely by a client. Clients program communicate to remote interfaces, not to classes implementing it. To be a remote interface, a interface must extend the Remote interface of java.rmi package.

import java.rmi.\*;

public interface AddServerInterface extends Remote

{

public int sum(int a,int b);

}

#### Implementation of remote interface

For implementation of remote interface, a class must either extend UnicastRemoteObject or use exportObject() method of UnicastRemoteObject class.

import java.rmi.\*;

import java.rmi.server.\*;

public class Adder extends UnicastRemoteObject implemnts AddServerInterface{

Adder() throws RemoteException{

super();

}

public int sum(int a,int b){

return a+b; }

}

#### Create AddServer and host rmi service

You need to create a server application and host rmi service Adder in it. This is done using rebind() method of java.rmi.Naming class. rebind() method take two arguments, first represent the name of the object reference and second argument is reference to instance of Adder.

import java.rmi.\*;

import java.rmi.registry.\*;

public class AddServer{

public static void main(String args[]){

try{

AddsServerInterface addService=new Adder();

Naming.rebind(“AddService”,addService);

//addservice object is hosted with name Addservice.

}catch(Exception e){

System.out.println(e);

}

}

}

#### Create client application

Client application contains a java program that invokes the lookup() method of the Naming class. This method accepts one argument, the rmi URL and returns a reference to an object of type AddServerInterface. All remote method invocation is done on this object.

import java.rmi.\*;

public class Client{

public static void main(String args[]){

try{

AddServerInterface st=(AddServerInterface)Naming.lookup(“rmi://”+args[0]+”/AddService”);

System.out.println(st.sum(25,8);

}catch(Exception e){

System.out.println(e);

}

}

}

*Submitted by RONY THANKACHAN, Rollno: 38*

**2015, Q19. Write a program read in name, roll number and GPA of N students in a class. Extend the program so that the read data is stored in a database table.**

**Ans.**

import java.sql.\*;

public class JDBCExample

{

// JDBC driver name and database URL

static final String JDBC\_DRIVER = "com.mysql.jdbc.Driver";

static final String DB\_URL = "jdbc:mysql://localhost/STUDENTS";

// Database credentials

static final String USER = "username";

static final String PASS = "password";

public static void main(String[] args)

{

Connection conn = null;

Statement stmt = null;

try{

// Register JDBC driver

Class.forName("com.mysql.jdbc.Driver");

// Open a connection

System.out.println("Connecting to a selected database...");

conn = DriverManager.getConnection(DB\_URL, USER, PASS);

System.out.println("Connected database successfully...");

// Execute a query

System.out.println("Inserting records into the table...");

stmt = conn.createStatement();

String sql = "INSERT INTO Student " +

"VALUES (100, 'Anu', 'Ali', 8)";

stmt.executeUpdate(sql);

sql = "INSERT INTO Student " +

"VALUES (101, 'Minnu', 'jose', 9)";

stmt.executeUpdate(sql);

sql = "INSERT INTO Student " +

"VALUES (102, 'Zaid', 'Khan', 7)";

stmt.executeUpdate(sql);

sql = "INSERT INTO Student " +

"VALUES(103, 'Sumit', 'raj', 8)";

stmt.executeUpdate(sql);

System.out.println("Inserted records into the table...");

}

catch(SQLException se)

{ //Handle errors for JDBC

se.printStackTrace();

}

catch(Exception e)

{ //Handle errors for Class.forName

e.printStackTrace();

}

finally{

//finally block used to close resources

try{

if(stmt!=null)

conn.close();

}

catch(SQLException se){

}// do nothing

try{

if(conn!=null)

conn.close();

}

catch(SQLException se){

se.printStackTrace();

}//end finally try

}//end try

}//end main

}//end JDBC

*Submitted by ROSE MARIA GEORGE, Rollno: 39*

**2015, Q20. Define a Servlet. Explain Servlet life cycle.**

**Ans.**

Servlets are small programs that execute on the server side of a web connection. Just as applets dynamically extent the functionality of a web browser servlets dynamically extend the functionality of a web server.

Servlets offers several advantages .

First performance is significantly better. Servlets execute within the address space of a web server. It is not necessary to create a separate process to handle each client request.

Second, servlets are platform-independent because they are written in java.

Third, the java security manager on the server enforces a set of restrictions to protect the resources on a server machine.

Finally the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMl mechanisms.

The Life Cycle of a Servlet

Three methods are central to the life cycle of a servlet. These are init( ), service( ), and destroy( ). They are implemented by every servlet and are invoked at specific times by the server. Let us consider a typical user scenario to understand when these methods are called. First, assume that a user enters a Uniform Resource Locator (URL) to a web browser, The browser then generates an HTTP request for this URL. This request is then sent to an appropriate server.

Second, this HTTP request is received by the web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address Space of the server.

Third, the server invokes the init( ) method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.

Fourth, the server invokes the service( ) method of the servlet. This method is called to process the HTTP request You will see that It is possible for the servlet to read data that has been provided in the HTTP request. It may also formulate an HTTP response for the cheat.

The servlet remains in the server's address space and is available to process any other HTTP requests received from clients The service( ) method is called for each HTTP requests. Finally, the server may decide to unload the servlet from its memory. The algorithm by which this determination is made are specific to each server The server calls the destroy() method to relinquish any resources such as file handles that are allocated for the servlet. Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

*Submitted by ROSHAN JOY, Rollno: 40*