INTERNAL ASSESSMENT & TERM WORK (5BCA3) SOLUTION
MAHARAJA COLLEGE ARA

Q1. How do we create a Thread in Java? What are Thread priorities?

Ans: A better way to create a thread in Java is to implement Runnable interface. A thread can be created by extending Java Thread class also. Now the question arises why implementing Runnable interface is a better approach? Answer is, if the thread class you are creating is to be subclass of some other class, it can't extend from the Thread class. This is because Java does not allow a class to inherit from more than one class. In such a case one can use Runnable interface to implement threads.

Let us see an example of creating a thread by implementing Runnable interface.

```
class PrintString
{
   public static void main (String args [ ])
   {
      StringThread t = new StringThread ("Java",50);
      new Thread(t). start ( );
   }
}

class StringThread implements Runnable
{
   private String str;
   private int num;

   StringThread(String s, int n)
   {
      str  = new String (s);
      num =n;
   }

   public void run ( )
   {
      for (int i=1; i<=num; i++)
         System.out.print (str+" ");
   }
}
```

```
**************************************************OUTPUT**********
**************************
```

```
Java Java Java Java Java Java Java Java Java Java Java Java Java Java Java
Java
Java Java Java Java Java Java Java Java Java Java Java Java Java Java Java
Java
Java Java Java Java Java Java Java Java Java Java Java Java Java Java Java
Java
Java Java
```

Thread Priority: In Java, thread scheduler can use the threadpriorities in the form of integer value to each of its thread to determine the execution schedule of threads . Thread gets the ready-to-run state according to their priorities.
The thread scheduler provides the CPU time to thread of highest priority during ready-to-run state.
Priorities are integer values from 1 (lowest priority given by the constant Thread.MIN_PRIORITY) to 10 (highest priority given by the constant Thread.MAX_PRIORITY). The default priority is 5(Thread.NORM_PRIORITY).

Constant	Description
Thread.MIN_PRIORITY	The maximum priority of any thread (an int value of 10)
Thread.MAX_PRIORITY	The minimum priority of any thread (an int value of 1)
Thread.NORM_PRIORITY	The normal priority of any thread (an int value of 5)

The methods that are used to set the priority of thread shown as:

Method	Description
setPriority()	This is method is used to set the priority of thread.
getPriority()	This method is used to get the priority of thread.

When a Java thread is created, it inherits its priority from the thread that created it. At any given time, when multiple threads are ready to be executed, the runtime system chooses the runnable thread with the highest priority for

execution. In Java runtime system, preemptive scheduling algorithm is applied. If at the execution time a thread with a higher priority and all other threads are runnable then the runtime system chooses the newhigher priority thread for execution. On the other hand, if two threads of the same priority are waiting to be executed by the CPU then the round-robin algorithm is applied in which the scheduler chooses one of them to run according to their round of time-slice.
Thread Scheduler

In the implementation of threading scheduler usually applies one of the two following strategies:

- **Preemptive scheduling ? If the new thread has a higher priority then current running thread leaves the runnable state and higher priority thread enter to the runnable state.**

- **Time-Sliced (Round-Robin) Scheduling ? A running thread is allowed to be execute for the fixed time, after completion the time, current thread indicates to the another thread to enter it in the runnable state.**

You can also set a thread's priority at any time after its creation using the setPriority method. Lets see, how to set and get the priority of a thread.
Example:

```
class MyThread1 extends Thread{
  MyThread1(String s){
  super(s);
  start();
  }
  public void run(){
  for(int i=0;i<3;i++){
  Thread cur=Thread.currentThread();
  cur.setPriority(Thread.MIN_PRIORITY);
  int p=cur.getPriority();
  System.out.println("Thread Name  :"+Thread.currentThread().getName());
  System.out.println("Thread Priority  :"+cur);
  }
  }
}
  class MyThread2 extends Thread{
  MyThread2(String s){
  super(s);
  start();
```

```
 }

public void run(){
  for(int i=0;i<3;i++){
  Thread cur=Thread.currentThread();
  cur.setPriority(Thread.MAX_PRIORITY);
  int p=cur.getPriority();
  System.out.println("Thread Name  :"+Thread.currentThread().getName());
  System.out.println("Thread Priority  :"+cur);
  }
  }
}
public class ThreadPriority{
  public static void main(String args[]){
  MyThread1 m1=new MyThread1("My Thread 1");
  MyThread2 m2=new MyThread2("My Thread 2");
  }
}
```

OUTPUT:

Output of the Program:

```
C:\shashi java>javac
ThreadPriority.java

C:\shashi java>java
ThreadPriority
Thread Name :My Thread 1
Thread Name :My Thread 2
Thread Priority :Thread[My
Thread 2,10,main]
Thread Name :My Thread 2
Thread Priority :Thread[My
Thread 2,10,main]
Thread Name :My Thread 2
Thread Priority :Thread[My
Thread 2,10,main]
Thread Priority :Thread[My
Thread 1,1,main]
Thread Name :My Thread 1
Thread Priority :Thread[My
Thread 1,1,main]
Thread Name :My Thread 1
Thread Priority :Thread[My
Thread 1,1,main]
```

INTERNAL ASSESSMENT & TERM WORK (5BCA3) SOLUTION
MAHARAJA COLLEGE ARA

Q2. (i) What are the different between awt and swing?

Ans:

1. **Swing is also called as JFC's (Java Foundation classes) and AWT stands for Abstract windows toolkit.**
2. **AWT components are called Heavyweight component and Swings are called light weight component because swing components sits on the top of AWT components and do the work.**
3. **Swing components require javax.swing package where as AWT components require java.awt package .**
4. **swings components are made in purely java and they are platform independent whereas AWT components are platform dependent.**
5. **we can have different look and feel in Swing whereas this feature is not supported in AWT.**
6. **Swing has many advanced features like JTabel, Jtabbed pane which is not available in AWT. Also. Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer.**
7. **With AWT, you have 21 "peers" (one for each control and one for the dialog itself). A "peer" is a widget provided by the operating system, such as a button object or an entry field object.**
8. **With Swing, you would have only one peer, the operating system's window object. All of the buttons, entry fields, etc. are drawn by the Swing package on the drawing surface provided by the window object. This is the reason that Swing has more code. It has to draw the button or other control and implement its behavior instead of relying on the host operating system to perform those functions.**
9. **Several consequences result from this difference between AWT and Swing. AWT is a thin layer of code on top of the OS, whereas Swing is much larger. Swing also has very much richer functionality.**
10. **Using AWT, you have to implement a lot of things yourself, while Swing has them built in. For GUI-intensive work, AWT feels very primitive to work with compared to Swing. Because Swing implements GUI functionality itself rather than relying on the host OS, it can offer a richer environment on all platforms Java runs on.**

Q2. (ii) What are the different between applet and application?

Ans: : Applets as previously described, are the small programs while applications are larger programs. Applets don't have the main method while in an application execution starts with the main method. Applets can run in our browser's window or in an applet viewer. To run the applet in an applet viewer will be an advantage for debugging. Applets are designed for the client site programming purpose while the applications don't have such type of criteria.

Applet are the powerful tools because it covers half of the java language picture. Java applets are the best way of creating the programs in java. There are a less number of java programmers that have the hands on experience on java applications. This is not the deficiency of java applications but the global utilization of internet. It doesn't mean that the java applications don't have the place. Both (Applets and the java applications) have the same importance at their own places. Applications are also the platform independent as well as byte oriented just like the applets.

Applets are designed just for handling the client site problems. While the java applications are designed to work with the client as well as server. Applications are designed to exist in a secure area. While the applets are typically used.

Applications and applets have much of the similarity such as both have most of the same features and share the same resources. Applets are created by extending the java.applet.Applet class while the java applications start execution from the main method. Applications are not too small to embed into a html page so that the user can view the application in your browser. On the other hand applet have the accessibility criteria of the resources. The key feature is that while they have so many differences but both can perform the same purpose.

Java Application an example:

```
public class ClassA

{
   String Name;
  int AccNumber;
  float Bal;
  void display(){
  System.out.println("Name: " + Name);
  System.out.println("Account Number: " + AccNumber);
```

```
 System.out.println("Balance: " + Bal);
 }
 public static void main(String args[]) {
 ClassA a = new ClassA();
 a.Name = "Vinod";
 a.AccNumber = 467256282;
 a.Bal =635;
 for (int i = 0; i < 20; i++)
 System.out.print("--");
  System.out.println(" PARTICULARS");
  a.display();
  for (int i = 0; i < 20; i++)
 System.out.print("--");
  System.out.println("End of display");
 }
}
```

**********OUTPUT**********
****

```
-------------------------------------
PARTICULARS
Name: Shashi Bhusan
Account Number:
60010015000016647
Balance: 635.0
-------------------------------------End of
display
```

Step for invoked in developing and running a local applet:

1. **Start a MS-DOS command window.**

choose Start/All Programs/Accessories Command Prompt

2. **Now, change to the directory of your choice from within the command window.**
   For example:
   >cd c:shashi java

3. Create some Java source code with Notepad from within the command window.

For example, suppose we want a class named "mycls",

then we would start Notepad with the name of the Java file:

4. The source code, A.java, might look like this:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/**
<APPLET code="mycls.class"
width=600
height=500>
</Applet>*/

public class mycls extends Applet implements ActionListener
{
Button b1,b2;
Label l;
        public void init()
        {
        b1=new Button(" First Button");
        b2=new Button("Secon Button");
        l=new Label("                                    ");
        add(b1);
        add(b2);
        add(l);
        b1.addActionListener(this);
        b2.addActionListener(this);
        }

                public void actionPerformed(ActionEvent e)
                {
                if(e.getSource()==b1)
                l.setText("Frist button is clicked ");
                if(e.getSource()==b2)
```

```
                l.setText("Second button is clicked");
                }
}
```

Don't forget to save the file!

5. If everything compiles fine

(we know this by the fact we had no error messages and the "mycls.class" file exists),

6. Then create html file with the help of this command write in command prompt:

> c:\shashi java javadoc mycls.java

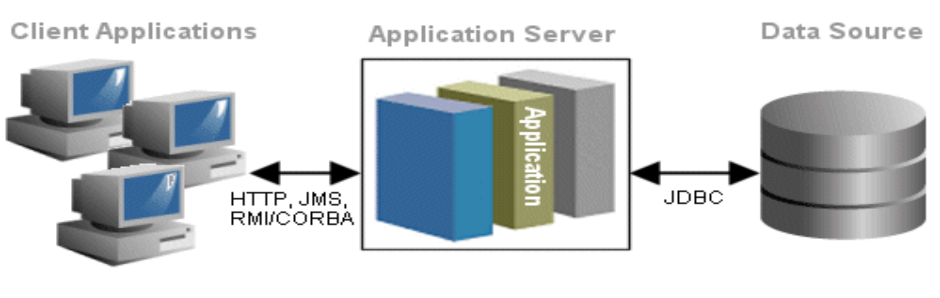
7. We are ready to run it. Write command in command prompt:

>c:\shashi java applatviewer mycls.html

OUTPUT

Q3. (i) Explain three tier architecture?

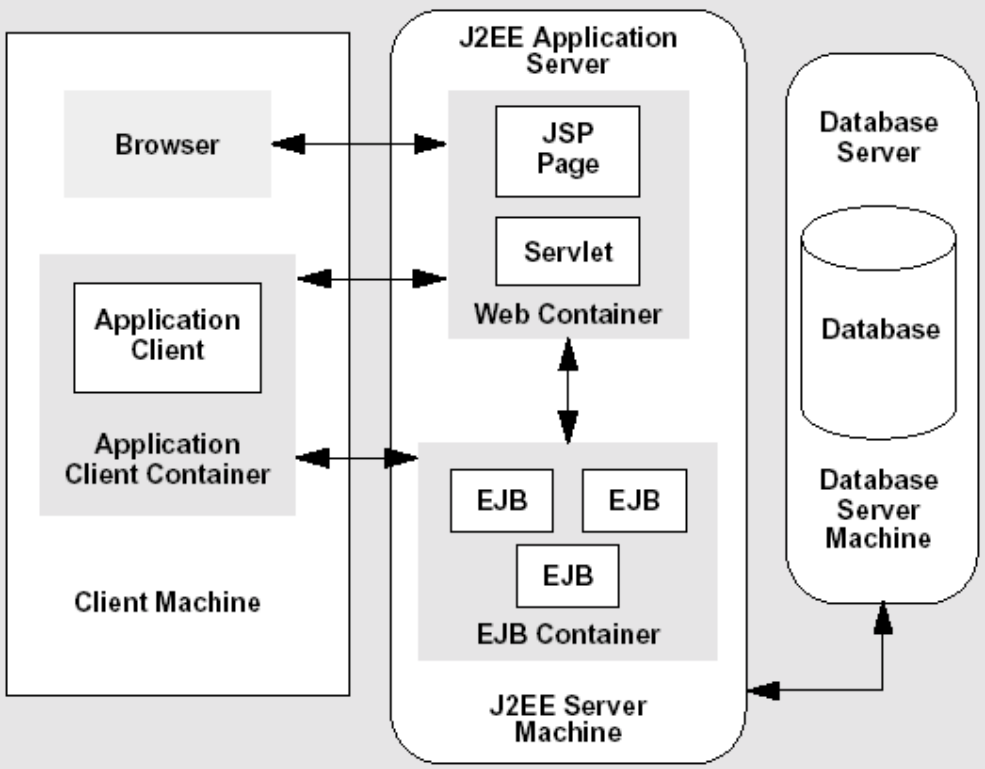
Ans: The three tier software architecture (a.k.a. three layer architectures) emerged to overcome the limitations of the two tier architecture. The third tier (middle tier server) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two tier architecture) by providing functions such as queuing, application execution, and database staging. The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user.



The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user.

A three tier distributed client/server architecture includes a user system interface top tier where user services (such as session, text input, dialog, and display management) reside.

The third tier provides database management functionality and is dedicated to data and file services that can be optimized without using any proprietary database management system languages. The data management component ensures that the data is consistent throughout the distributed environment through the use of features such as data locking, consistency, and replication. It should be noted that connectivity between tiers can be dynamically changed depending upon the user's request for data and services.



The middle tier provides process management services (such as process development, process enactment, process monitoring, and process resourcing) that are shared by multiple applications.

The middle tier server (also referred to as the application server) improves performance, flexibility, maintainability, reusability, and scalability by centralizing process logic. Centralized process logic makes administration and change management easier by localizing system functionality so that changes must only be written once and placed on the middle tier server to be available throughout the systems. With other architectural designs, a change to a function (service) would need to be written into every application.

In addition, the middle process management tier controls transactions and asynchronous queuing to ensure reliable completion of transactions. The middle tier manages distributed database integrity by the two phase commit process. It provides access to resources based on names instead of locations, and thereby improves scalability and flexibility as system components are added or moved.

Q3. (ii) Explain different types of JDBC drivers and their advantages?

Ans: A JDBC driver is a software component enabling a Java application to interact with a database.[1] JDBC drivers are analogous to ODBC drivers, ADO.NET data providers, and OLE DB providers.

To connect with individual databases, JDBC (the Java Database Connectivity API) requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

JDBC technology drivers fit into one of four categories.

1. **JDBC-ODBC Driver: The JDBC type 1 driver, also known as the JDBC-ODBC bridge, is a database driver implementation that employs the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls.**

The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the underlying operating system the JVM is running upon. Also, use of this driver leads to other installation dependencies; for example, ODBC must be installed on the computer having the driver and the database must support an ODBC driver. The use of this driver is discouraged if the alternative of a pure-Java driver is available. The other implication is that any application using a type 1 driver is non-portable given the binding between the driver and platform. This technology isn't suitable for a high-transaction environment. Type 1 drivers also don't support the complete Java command set and are limited by the functionality of the ODBC driver.

Sun provides a JDBC-ODBC Bridge driver: sun.jdbc.odbc.JdbcOdbcDriver. This driver is native code and not Java, and is closed source.

Functions

- **Translates a query by JDBC into a corresponding ODBC query, which is then handled by the ODBC driver.**

Advantages

Almost any database, for which ODBC driver is installed, can be accessed.

Disadvantages

- **Performance overhead since the calls have to go through the jdbc Overhead bridge to the ODBC driver, then to the native db connectivity interface (thus may be slower than other types of drivers).**
- **The ODBC driver needs to be installed on the client machine.**
- **Not suitable for applets, because the ODBC driver needs to be installed on the client.**

2. **Native-API Driver: The JDBC type 2 driver, also known as the Native-API driver, is a database driver implementation that uses the client-side libraries**

of the database. The driver converts JDBC method calls into native calls of the database API.

Advantages

- **As there is no implementation of jdbc-odbc bridge, its considerably faster than a type 1 driver.**

Disadvantages

- **The vendor client library needs to be installed on the client machine.**
- **Not all databases have a client side library**
- **This driver is platform dependent**
- **This driver supports all java applications except Applets**

3. **Network-Protocol Driver(MiddleWare Driver): The JDBC type 3 driver, also known as the Pure Java Driver for Database Middleware, is a database driver implementation which makes use of a middle tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.**

This differs from the type 4 driver in that the protocol conversion logic resides not at the client, but in the middle-tier. Like type 4 drivers, the type 3 driver is written entirely in Java. The same driver can be used for multiple databases. It depends on the number of databases the middleware has been configured to support. The type 3 driver is platform-independent as the platform-related differences are taken care of by the middleware. Also, making use of the middleware provides additional advantages of security and firewall access.

Functions

- **Sends JDBC API calls to a middle-tier net server that translates the calls into the DBMS-specific network protocol. The translated calls are then sent to a particular DBMS.**
- **Follows a three tier communication approach.**
- **Can interface to multiple databases - Not vendor specific.**
- **The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into database commands for that database.**
- **Thus the client driver to middleware communication is database independent.**

Advantages

- **Since the communication between client and the middleware server is database independent, there is no need for the database vendor library on the client. The client need not be changed for a new database.**
- **The middleware server (which can be a full fledged J2EE Application server) can provide typical middleware services like caching (of connections, query results, etc.), load balancing, logging, and auditing.**
- **A single driver can handle any database, provided the middleware supports it.**
- **Eg:-IDA Server**

Disadvantages

- **Requires database-specific coding to be done in the middle tier.**
- **The middleware layer added may result in additional latency, but is typically overcome by using better middleware services.**

4. Native-Protocol Driver(Pure Java Driver)

The JDBC type 4 driver, also known as the Direct to Database Pure Java Driver, is a database driver implementation that converts JDBC calls directly into a vendor-specific database protocol.

Written completely in Java, type 4 drivers are thus platform independent. They install inside the Java Virtual Machine of the client. This provides better performance than the type 1 and type 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 3 drivers, it does not need associated software to work.

As the database protocol is vendor specific, the JDBC client requires separate drivers, usually vendor supplied, to connect to different types of databases. This type includes, for example, the widely used Oracle thin driver.

Advantages

- **Completely implemented in Java to achieve platform independence.**
- **These drivers don't translate the requests into an intermediary format (such as ODBC).**
- **The client application connects directly to the database server. No translation or middleware layers are used, improving performance.**
- **The JVM can manage all aspects of the application-to-database connection; this can facilitate debugging.**
- **Provides a way to manage copies of the database for each user.**

Disadvantages

- **Drivers are database dependent, as different database vendors use wildly different (and usually proprietary) network protocols.**

Q4. (i) Explain buffer reader and buffer writer classes?

Ans: Buffer Reader: Buffer Reader class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

The buffer size may be specified, or the default size may be used. The default is large enough for most purposes.

In general, each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream. It is therefore advisable to wrap a BufferedReader around any Reader whose read() operations may be costly, such as FileReaders and InputStreamReaders. For example,

BufferedReader in

= new BufferedReader(new FileReader("foo.in"));

will buffer the input from the specified file. Without buffering, each invocation of read() or readLine() could cause bytes to be read from the file, converted into characters, and then returned, which can be very inefficient.

Programs that use DataInputStreams for textual input can be localized by replacing each DataInputStream with an appropriate BufferedReader.

Programming example:

```
import java.io.*;

public class InputstramToBufferreaderExample
{
public static void main(String args[]) throws IOException
 {
 System.out.println("Enter the input stream:");
 InputStreamReader in= new InputStreamReader(System.in);
 BufferedReader bin= new BufferedReader(in);
 String text=bin.readLine();
 System.out.println("The values after BufferedReader:"+text);
 }}
```

The output of the program is given below:

```
C:\shashi>javac
InputstramToBufferreaderExample.java
C:\shashi>java
InputstramToBufferreaderExample
Enter the input stream:
Shahsi kumar
The values after
BufferedReader:shashi kumar
```

Buffer Writer class: Buffer Writer class write text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.

The buffer size may be specified, or the default size may be accepted. The default is large enough for most purposes.

A newLine() method is provided, which uses the platform's own notion of line separator as defined by the system property line.separator. Not all platforms use the newline character ('\n') to terminate lines. Calling this method to terminate each output line is therefore preferred to writing a newline character directly.

In general, a Writer sends its output immediately to the underlying character or byte stream. Unless prompt output is required, it is advisable to wrap a BufferedWriter around any Writer whose write() operations may be costly, such as FileWriters and OutputStreamWriters. For example,

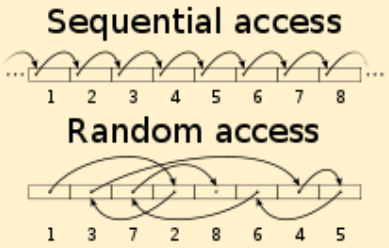
```
PrintWriter out
  = new PrintWriter(new BufferedWriter(new FileWriter("foo.out")));
```

will buffer the PrintWriter's output to the file. Without buffering, each invocation of a print() method would cause characters to be converted into bytes that would then be written immediately to the file, which can be very inefficient.

Q4. (ii) How we read data from sequential access files? Write a suitable example?

Ans> In computer science, sequential access means that a group of elements (e.g. data in a memory array or a disk file or on magnetic tape data storage) is accessed in a predetermined, ordered sequence. Sequential access is sometimes the only way of accessing the data, for example if it is on a tape. It may also be the access method of choice, for example if we simply want to process a sequence of data elements in order.[1]

In data structures, a data structure is said to have sequential access if one can only visit the values it contains in one particular order. The canonical example is the linked list. Indexing into a list that has sequential access requires O(k) time, where k is the index. As a result, many algorithms such asquicksort and binary search degenerate into bad algorithms that are even less efficient than their naïve alternatives; these algorithms are impractical without random access. On the other hand, some algorithms, typically those that don't index, require only sequential access, such as mergesort, and face no penalty.



Example:

```
import java.io.*;

class bytewrite

{

public static void main(String[] arg)

        {

byte na[]={'\n','\n','\t',' ','S','h','a','s','h','i','\n','\n','\t',' ','B','C','A'};

        FileOutputStream file=null;

        FileInputStream infile=null;

        int n;

                try

                {

                file=new FileOutputStream("name.dat");

                file.write(na);

                file.close();

                }
```

```
                    catch(Exception e)
                    {
                    System.out.print("\n\n\t Error !! "+e);
                    }

        try
        {       System.out.print("\n\n\t Read from file .....");
                infile=new FileInputStream("name.dat");
                while((n=infile.read())!=-1)
                    {
                    System.out.print((char)n);
                    }
        infile.close();
        }
        catch(Exception e)
        {
System.out.print("\n\n\t\t Error -----!!"+e);
        }}}
```

**********OUTPUT**********

```
Read from file.....

Shashi

BCA
```

Q5 Explain Different types layout manager in java?

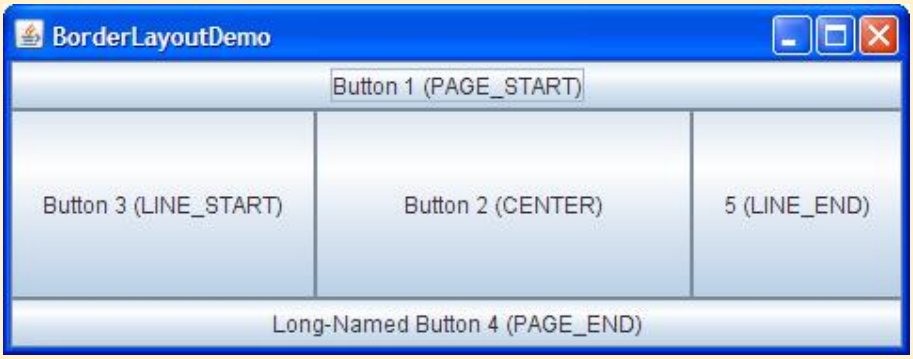
Ans: The Layout Manager inter face defines the responsibilities of something that wants to lay out Components within a Container. It is the Layout Manager's duty to determine the position and size of each component within the Container. You

will never directly call the methods of the Layout Manager inter face; for the most part, layout managers do their work behind the scenes. Once you have created a Layout Manager object and told the container to use it (by calling setLayout()), you're finished with it. The system calls the appropriate methods in the layout manager
when necessary.
Therefore, the LayoutManager inter face is most important when you are writing a new layout manager; we'll discuss it here because it's the scaffolding on which all layout managers are based. Like any interface, Layout Manager specifies the methods a layout manager must implement but says nothing about how the Layout Manager does its job. Therefore, we'll make a few observations before proceeding.

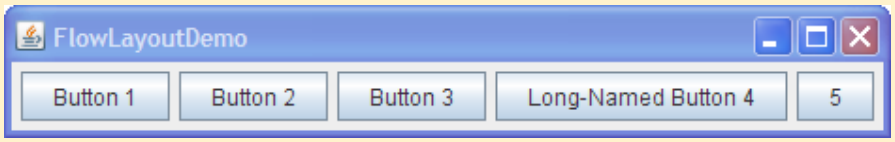
First, a layout manager is free to ignore some of its components; there is no requirement that a layout manager display everything. For example, a Container using a BorderLayout might include thirty or forty components. However, the BorderLayout will display at most five of them (the last component placed in each of its five named areas). Likewise, a CardLayout may manage many components but displays only one at a time.

A layout manager organizes the objects in a container
Different layouts are used to organize or to arrange objects

1. Border Layout: Every content pane is initialized to use a BorderLayout. (As Using Top-Level Containers explains, the content pane is the main container in all frames, applets, and dialogs.) ABorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area. Tool bars that are created using JToolBarmust be created within a BorderLayout container, if you want to be able to drag and drop the bars away from their starting positions

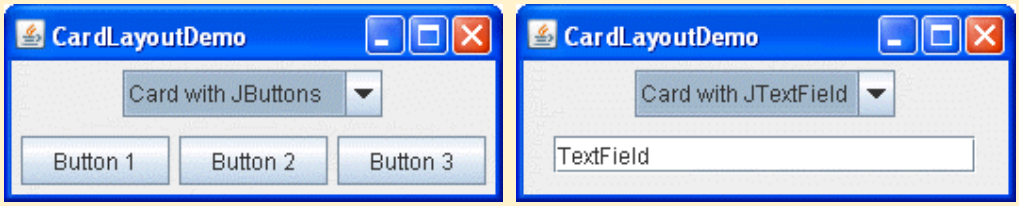


2. Flow Layout:



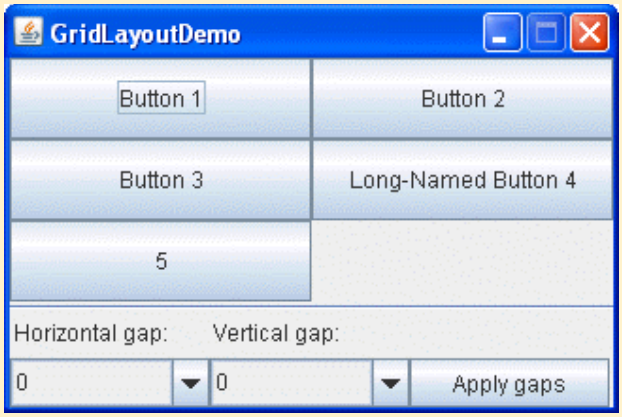
FlowLayout is the default layout manager for every Panel. It simply lays out components in a single row, starting a new row if its container is not sufficiently wide. Both panels in CardLayoutDemo, shown previously, use FlowLayout.

3. CardLayout



The CardLayout class lets you implement an area that contains different components at different times. A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays. An alternative to using CardLayout is using a tabbed pane, which provides similar functionality but with a pre-defined GUI.

5. GridLayout



GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.

Q6. Explain the life cycle of an applet?

Ans: Introduction: In this Section you will learn about the lifecycle of an applet and different methods of an applet. Applet runs in the browser and its lifecycle method are called by JVM when it is loaded and destroyed.

we will cover Applet Structure, Executing an Applet, Drawing shapes with Graphics Methods and Using Color and Font classes.

Applet Environment

An applet is a small program that is intended to be embedded inside another application such as a browser. The JApplet class must be the superclass of any applet that is to be embedded in a Web page or viewed by the Java Applet Viewer (appletviewer.exe). The JApplet class provides a standard interface between applets and their environment. JApplet hierarchy is as follows:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Panel
        java.applet.Applet
          javax.swing.JApplet
```

Applets Structure

- There is no main method.
- Two methods that are called automatically- init() and paint()
- The init method initializes variables and objects; if you don't have one you will inherit one from the JApplet class.
- Use paint to draw screen
- A lot of methods exist in JApplet class so the "extends" keyword inherits everything that the class has. In the above example JApplet is parent class and shellapplet is the subclass so use the keyword "extends" to create inheritance.
- You have to import JApplet and java.awt.Graphics (abstract windowing toolkit) to get Graphics to paint.
- All applets must inherit JApplet
- Use the "super" keyword in subclass to invoke method in the superclass.
- super.paint(g); this says uses the paint method from JApplet in my paint class and I'm not adding anything to it.

Here are the lifecycle methods of an Applet:

init(): This method is called to initialized an applet

start(): This method is called after the initialization of the applet.

stop(): This method can be called multiple times in the life cycle of an Applet.

destroy(): This method is called only once in the life cycle of the applet when applet is destroyed.

init () method: The life cycle of an applet is begin on that time when the applet is first loaded into the browser and called the init() method. The init() method is called only one time in the life cycle on an applet. The init() method is basically called to read the PARAM tag in the html file. The init () method retrieve the passed parameter through the PARAM tag of html file using get Parameter() method All the initialization such as initialization of variables and the objects like image, sound file are loaded in the init () method .After the initialization of the init() method user can interact with the Applet and mostly applet contains the init() method.
Start () method: The start method of an applet is called after the initialization method init(). This method may be called multiples time when the Applet needs to be started or restarted. For Example if the user wants to return to the Applet, in this situation the start Method() of an Applet will be called by the web browser and the user will be back on the applet. In the start method user can interact within the applet.

Stop () method: The stop() method can be called multiple times in the life cycle of applet like the start () method. Or should be called at least one time. There is only miner difference between the start() method and stop () method. For example the stop() method is called by the web browser on that time When the user leaves one applet to go another applet and the start() method is called on that time when the user wants to go back into the first program or Applet.
destroy() method: The destroy() method is called only one time in the life cycle of Applet like init() method. This method is called only on that time when the browser needs to Shut down.

Running an Applet: We need to use html code to run applets. The minimum html required to run applet with a browser (java host) is as follows:

```
<HTML>
<HEAD>
<TITLE>TitleName</TITLE>
</HEAD>
<BODY>
<APPLET>
   CODE = Classname.class
   CODEBASE = . directory of class file
   WIDTH = 50 width of window in pixels
   HEIGHT = 50 height of window in pixels
</APPLET>
</BODY>
</HTML>
```

Note that in the applet tag you include the. class bytecode file and not the .java.

Example:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/**
<APPLET code="mycls.class"
width=600
height=500>
</Applet>*/

public class mycls extends Applet implements ActionListener
{
Button b1,b2;
Label l;
        public void init()
        {
        b1=new Button(" First Button");
        b2=new Button("Secon Button");
```

```
l=new Label("                                    ");
add(b1);
add(b2);
add(l);
b1.addActionListener(this);
b2.addActionListener(this);
}

        public void actionPerformed(ActionEvent e)
        {
        if(e.getSource()==b1)
        l.setText("Frist button is clicked ");
        if(e.getSource()==b2)
        l.setText("Second button is clicked");
        }
}
```

**********OUTPUT**********

Q7. Write shorts notes

(i). JAVA Beans

Ans: JavaBeans are reusable software components for Java. Practically, they are classes written in the Java programming language conforming to a particular convention. They are used to encapsulate many objects into a single object (the bean), so that they can be passed around as a single bean object instead of as multiple individual objects. A JavaBean is a Java Object that isserializable, has a 0-argument constructor, and allows access to properties using getter and setter methods.

Advantage of Java Beans:

- **A Bean obtains all of the benefits of Java's "write once, run anywhere" paradigm.**
- **The properties, events, and methods of a Bean that are exposed to another application can be controlled.**
- **Auxiliary software can be provided to help configure a Bean.**
- **The configuration settings of a Bean can be saved in a persistent storage and can be restored at a later time.**
- **A Bean may register to receive events from other objects and can generate events that are sent to it.**

Disadvantage of Java Bean:

- **A class with a nullary constructor is subject to being instantiated in an invalid state. If such a class is instantiated manually by a developer (rather than automatically by some kind of framework), the developer might not realize that the class has been improperly instantiated. The compiler can't detect such a problem, and even if it's documented, there's no guarantee that the developer will see the documentation.**
- **Having to create a getter for every property and a setter for many, most, or all of them, creates an immense amount of boilerplate code.**

Example:

```
package beans;

/**
 * Class <code>PersonBean</code>.
 */
public class PersonBean implements java.io.Serializable {

    private String name;

    private boolean deceased;

    /** No-arg constructor (takes no arguments). */
    public PersonBean() {
    }

    /**
     * Property <code>name</code> (note capitalization) readable/writable.
```

```
     */
    public String getName() {
        return this.name;
    }

    /**
     * Setter for property <code>name</code>.
     * @param name
     */
    public void setName(final String name) {
        this.name = name;
    }

    /**
     * Getter for property "deceased"
     * Different syntax for a boolean field (is vs. get)
     */
    public boolean isDeceased() {
        return this.deceased;
    }

    /**
     * Setter for property <code>deceased</code>.
     * @param deceased
     */
    public void setDeceased(final boolean deceased) {
        this.deceased = deceased;
    }
}
```

(ii). Random Access files

Ans: Instances of this class support both reading and writing to a random access file. A random access file behaves like a large array of bytes stored in the file system. There is a kind of cursor, or index into the implied array, called the file pointer; input operations read bytes starting at the file pointer and advance the file pointer past the bytes read. If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. Output operations that write past the current end of the implied array cause the array to be extended. The file pointer can be read by the getFilePointer method and set by the seek method.

It is generally true of all the reading routines in this class that if end-of-file is reached before the desired number of bytes has been read,

an EOFException (which is a kind of IOException) is thrown. If any byte cannot be read for any reason other than end-of-file, an IOException other than EOFException is thrown. In particular, an IOException may be thrown if the stream has been closed.

Example:

```
import java.io.*;

  public class ReadAccessFile{
  public static void main(String[] args) throws IOException{
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  System.out.print("Enter File name : ");
  String str = in.readLine();
  File file = new File(str);
  if(!file.exists())
  {
  System.out.println("File does not exist.");
  System.exit(0);
  }
  try{
    RandomAccessFile rand = new RandomAccessFile(file,"r");
  int i=(int)rand.length();
  System.out.println("Length: " + i);
 rand.seek(0);  //Seek to start point of file
  for(int ct = 0; ct < i; ct++){
  byte b = rand.readByte(); //read byte from the file
  System.out.print((char)b); //convert byte into char
  }
  rand.close();
  }
  catch(IOException e)
  {
  System.out.println(e.getMessage());
  }
  }
}
```

Output of the Program:

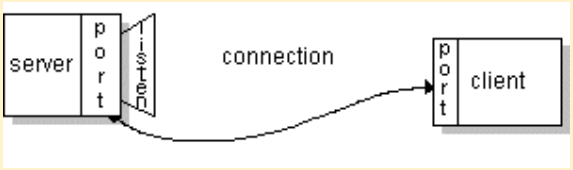
```
C:\shashi>java
ReadAccessFile
Enter File name
: Filterfile.txt
Length: 30
??t h i s i s a f i
l e
C:\shashi>
```

(iii).Socket

Ans: Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

- **Socket is a combination of ip address and port number**

- **Client sends request to the server by using server's ip address and the corresponding port number**
- **Java has java.net.Socket class to communicate through socket**
- **A socket allows to perform 4 fundamental socket operations:**
  1. Connecting to remote systems
  2. Sending data to the nodes in the network
  3. Receiving data from the server
  4. Closing connection to the server
- **Every socket is associated with only one remote host.**
- **Data is sent and received through byte oriented streams. The getOutputStream() is utilized to place the request and getInputStream() is utilized to receive response.**
- **The following are the methods to return the information about the socket:**
  public InetAddress getInetAddress() – to obtain the ip address
  public int getPort() – to obtain the port

Explain the characteristics of Java socket class.

The following are certain characteristics that socket share:

1. The socket is available as long as the network process maintains an open link to the socket
2. A socket can be named in order to communicate with other sockets in a network
3. Communication is performed by the sockets in network when the server receives the connections and requests from them.
4. Messages can also be shared between a client and the server
5. Pairs of sockets can be created. However, this is possible in the AF_UNIX address family

(iv). Thread Synchronization

Ans: In Java, the threads are executed independently to each other. These types of threads are called as asynchronous threads. But there are two problems may be occur with asynchronous threads.

- **Two or more threads share the same resource (variable or method) while only one of them can access the resource at one time.**
- **If the producer and the consumer are sharing the same kind of data in a program then either producer may produce the data faster or consumer may retrieve an order of data and process it without its existing.**

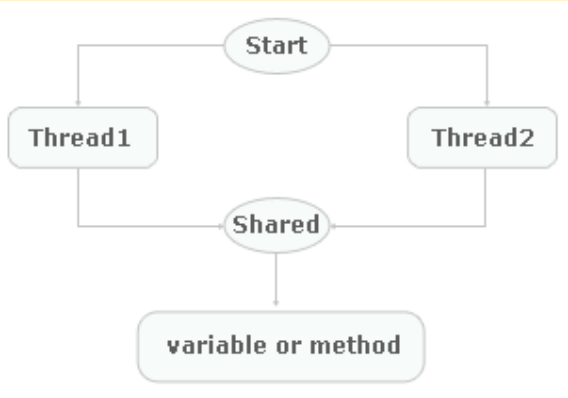
INTERNAL ASSESSMENT & TERM WORK (5BCA3) SOLUTION
MAHARAJA COLLEGE ARA

Suppose, we have created two methods as increment( ) and decrement( ). which increases or decreases value of the variable "count" by 1 respectively shown as:

```
public void
increment( ) {
 count++;
 }

 public void
decrement( ) {
 count--;
 }
public int value() {
      return count;
  }
```

When the two threads are executed to access these methods (one for increment( ),another fordecrement( )) then both will share the variable "count". in that case, we can't be sure that what value will be returned of variable "count".
We can see this problem in the diagram shown below:



To avoid this problem, Java uses monitor also known as? semaphore? to prevent data from being corrupted by multiple threads by a keyword synchronized to synchronize them and intercommunicate to each other. It is basically a mechanism which allows two or more threads to share all the available resources in a sequential manner. Java's synchronized is used to ensure that only one thread is in a critical region. critical region is a lock area where only one thread is run (or lock) at a time. Once the thread is in its critical section, no other thread can enter to that critical region. In that case, another thread will has to wait until the current thread leaves its critical section.

BY: ABHAY KUMAR MISHRA, DEPARTMENT OF B.C.A. ,MAHARAJA COLLEGE ARA

INTERNAL ASSESSMENT & TERM WORK (5BCA3) SOLUTION
MAHARAJA COLLEGE ARA

BY: ABHAY KUMAR MISHRA, DEPARTMENT OF B.C.A. ,MAHARAJA COLLEGE ARA