

B. E.
Eighth Semester Examination, May-2008
ADVANCE JAVA

Note : Attempt any five questions.

Q. 1. (a) How is platform independent feature of Java implemented? Explain the role of main() method in the execution of Java programs.

Ans. Platform independent :

Java's architecture facilitates the creation of platform-independent software, but also allows you to create software that is platform-specific. When you write a Java program, platform independence is an option. The degree of platform independence of any Java program depends on several factors. As a developer, some of these factors are beyond your control, but most are within your control. Primarily, the degree of platform independence of any Java program you write depends on how you write it. The most basic factor determining the a Java program's platform independence is the extent to which the Java Platform has been deployed on multiple platforms. Java programs will only run on computers and devices that host a Java Platform. Thus, before one of your Java programs will run on a particular computer owned by, say, your friend Alicia, two things must happen. First, the Java Platform must be ported to Alicia's particular type of hardware and operating system. Once the port has been done by some Java Platform vendor, that port must in some way get installed on Alicia's computer. So a critical factor determining the true extent of platform independence of Java programs--and one that is beyond the control of the average developer--is the availability of Java Platform implementations and their distribution. Fortunately for the Java developer, the deployment of the Java Platform has proceeded with great momentum, starting with Web-browsers, then moving on to desktop, workstation, network operating systems, and into many different kinds of consumer and embedded devices. It is increasingly likely, therefore, that your friend Alicia will have a Java Platform implementation on her computer or device.

Main() :

A Java package is a way to collect different parts of a large program together logically. The Java system (JDK or Java Development Kit) comes with several packages of its own, but you can create your own. In Java the qualifier public means that something is available across packages. It is also possible to restrict visibility of names like Temporary to a single package or even more, but here we make it public. The particular form of main is required by Java. While the following explanation of its parts is not needed now, and probably not especially understandable, you can return to it later when you know more. In Java, main is a static method. This means the method is part of its class and not part of objects. Robots are objects. They are defined by classes, which are like factories for the creation of objects. In terms of our metaphor for robot programming, a static method is like the instructions read to robots by the helicopter pilot, not the instructions known by the robots themselves. Strings in Java are sequence of characters, like the characters in this sentence. The matched brackets indicate that an array of Strings is required. An array is a certain kind of linear collection of things. The name args is just a name for this array. This name is the only part of the declaration of main that can vary. The declaration (String [] args) is in parentheses as part of the declaration of main

to indicate that when the class Temporary is executed by your computer's operating system, an array of

strings can be sent to the program to help with program initialization. Since they are not referenced again in the body of main (the part between braces {...}) they won't be used here. The parentheses, like the rest, are required. There is no space between [and]. Braces { and } are used to collect statements into a "block" in Java and in some other programming languages. Statements in Java end with semicolons. To execute the above program you first need to translate it into a machine readable form using the Java compiler. Then you need to use the Java run-time to execute the machine readable version. When you do this you give the run time the name of the class that contains your main method.

Q. 1. (b) Why exceptional errors occur during the execution of a program? Discuss the usage of "finally" block.

Ans. Exceptional error :

Exceptions in java are any abnormal, unexpected events or extraordinary conditions that may occur at runtime. They could be file not found exception, unable to get connection exception and so on. On such conditions java throws an exception object Java Exceptions are basically Java objects. No Project can never escape a java error exception. Java exception handling is used to handle error conditions in a program systematically by taking the necessary action. Exception handlers can be written to catch a specific exception such as Number Format exception, or an entire group of exceptions by using a generic exception handlers. Any exceptions not specifically handled within a Java program are caught by the Java run time environment

An exception is a subclass of the Exception/Error class, both of which are subclasses of the Throwable class. Java exceptions are raised with the throw keyword and handled within a catch block.

```
public class DivideException {
    public static void main(String[] args) {
        division(100,4);    //Line 1
        division(100,0);    // Line 2
        System.out.println("Exit main( )."); }
    public static void division(int totalSum, int totalNumber) {
        System.outprintln("Computing Division. ");
        int average = totalSum/totalNumber;
        System.outprintln("Average : "+ average); }}
```

Finally block :

A finally block is always executed, regardless of the cause of exit from the try block, or whether any catch block was executed. Generally finally block is used for freeing resources, cleaning up, closing connections etc. If the finally block executes a control transfer statement such as a return or a break statement, then this control statement determines how the execution will proceed regardless of any return or control statement present in the try or catch.

```
try { <code> } catch "exception type1" <parameter1" { // 0 or more
<statements> } } finally {    // finally block
<statements> }
```


Q. 2. (a) Under what circumstances would a "Socket Exception" be thrown? Describe how a client connects to a server.

Ans. Socket exception :

This Socket Exception may be thrown during socket creation or setting options, and is the superclass of all other socket related exceptions. The Java API provides a nice set of libraries for initiating and communicating over TCP/IP sockets. The standard Unix API provides a very low-level interface, and requires a significant amount of error checking and handling. It is also very architecture-specific. Java, with its java.net package, has removed all of the portability issues, and most of the tedium of network I/O. However, there is room for improvement. To develop a Java-based server application that listens for client connections and spawns off threads to handle them, a number of steps must be taken. In the following paragraphs, we will go over each of these steps, and will touch on several useful techniques including the use of Sockets, Threads, and basic Java data I/O. The result will be a package that can be used to develop almost any client-server protocol by subclassing the classes in the package. These classes can then be integrated into an applet, or a standalone application. A typical client-server architecture consists of a server application running on one machine, and one or more clients -- often, but not always, operating on different machines across a network. The server is modeled as a perpetual process that waits for clients to connect (request a service), and then processes the client requests. Java threads are ideal to the implementation of a server application, because a new thread can be started for each connecting client. This prevents one client request from holding up others if and when it enters a wait state, or some time-consuming task.

Q. 2. (b) Discuss with a suitable example, how a client application would read a file from a server through a URL connection.

```
Ans.  import java. io. *;
      import java.net.*;
      public class ServletCom    {
      public static void main (String[] args)
      throws Exception {
      HttpURLConnection conn = null;
      BufferedReader br = null;
      DataOutputStream dos = null;
      DataInputStream inStream = null;
      InputStream is = null;
      OutputStream os = null;
      boolean ret = false;
      String StrMessage = "";
      String exsistingFileName = "C:\\account.xls";
      String lineEnd = "\r\n";
```

/

```
String twoHyphens = "--";
String boundary = "*****";
int bytesRead, bytesAvailable, bufferSize;
byte[] buffer;
int maxBufferSize = 1*1024*1024;
String responseFromServer = "";
String urlString = "http://localhost:8080/FileUpload/requestupload"; try {
//----- CLIENT REQUEST
FileInputStream fileInputStream = new FileInputStream( new
File (existingFileName) );
URL url = new URL(urlString);
conn = (URLConnection) url.openConnection();
conn.setDoInput(true);
conn.setDoOutput(true);
conn.setUseCaches(false);
conn.setRequestMethod("POST");
conn.setRequestProperty ("Connection", "Keep-Alive");
conn.setRequestProperty("Content-Type", "multipart/form
data;boundary="+boundary);
dos = new DataOutputStream( conn.getOutputStream() );
dos.writeBytes(twoHyphens + boundary + lineEnd);
dos.writeBytes("Content-Disposition: form-data; name=\"upload\";\"
+ " filename=\"" + existingFileName + "\"\" + lineEnd);
dos.writeBytes(lineEnd);
// create a buffer of maximum size
bytesAvailable = fileInputStream.available();
bufferSize = Math.min(bytesAvailable, maxBufferSize);
buffer = new byte[bufferSize];
// read file and write it into form...
```

```
bytesRead = fileInputStream.read(buffer, 0, bufferSize);
while (bytesRead > 0) {
dos.write(buffer, 0, bufferSize);
bytesAvailable = fileInputStream.available();
bufferSize = Math.min(bytesAvailable, maxBufferSize);
bytesRead = fileInputStream.read(buffer, 0, bufferSize); }
// send multipart form data necessary after file data...
dos.writeBytes(lineEnd);
dos.writeBytes(twoHyphens + boundary + twoHyphens + lineEnd);
// close streams
fileInputStream.close();
dos.flush();
dos.close(); }
catch (MalformedURLException ex) {
System.out.println("From ServletCom CLIENT REQUEST:"+ex);
catch (IOException ioe) {
System.out.println("From ServletCom CLIENT REQUEST:"+ioe);
//-----read the SERVER RESPONSE
try {
inStream = new DataInputStream ( conn.getInputStream() );
String str;
while (( str = inStream.readLine () ) != null) {
System.out.println("Server response is: "+str);
System.out.println(""); }
inStream.close(); }
catch (IOException ioex) {
System.out.println("From (ServerResponse): "+ioex); } } }.
```

Q. 3. (a) Explain various JDBC programming concepts.

Ans. JOBC programming concepts :

Transactions: Whenever a connection is created by using the JDBC, then by default it is in auto-commit mode. This means that SQL statement will be automatically committed immediately after it is executed and it is treated as a transaction. But imagine a situation where you want to execute a batch of statements, either they should commit at once or they should get failed together. For this we need to disable the auto-commit mode by using the method:

*** `Con.setAutoCommit(false)` :**

After setting the auto-commit as false, no SQL statement will be committed until we call the `con.commit()` method. If there arises any problem while committing then the set of statements will be rollback, without committing.

Logging: on the server--->logging--->JDBC.

By this we can enable JDBC logging and specify a log file name for the JDBC log.

Attributes of logging :

1. Enable JDBC Logging :

It determines whether the server has a JDBC log file.

2. JDBC log file name :

It is the name of the log file.

Isolation :

The isolation is needed when there are concurrent transactions. Concurrent transactions are transactions that occur at the same time. In isolation one transaction does not interfere with another. For setting the isolation level for a JDBC transaction, use the

`Connection.setTransaction(int level)` method.

By using the snapshot isolation level we can only see the snapshot of the data locked by other transactions when running from inside the transaction with snapshot isolation level.

Some of the transaction level are given below :

1. `Transaction_None`.
2. `Transaction_Read_Uncommitted`.
3. `Transaction_Read_Committed`.
4. `Transaction_Repeatable_Read`.
5. `Transaction_serializable`.

By setting the isolation levels you are having an impact on the performance of the transaction. You can get the existing isolation level with: `getTransactionIsolation()` method.

Concurrency : Database concurrency controls ensure that the transactions occur in an ordered fashion. Concurrency control deals with the issue involved with allowing multiple people simultaneous access to shared entities.

Q. 3. (b) Write a program to connect to a database query it and display the results.

Ans. The following steps describe the things with example program :

1. Arrange the components in the frame, using a grid bag layout.
2. Populate the author and publisher text boxes by running two queries that returns all author and publisher names from db.

We initialize the connection and statement objects in the constructor. We hang on to them for the life of the program. Just before the program exits we trap the 'window closing' event and these objects are closed.

```
class query DBframe( ) {
    conn = getconnection( );
    stat = conn. createstatement( );
    -----
    add (new) window adapter( )
    {
    public void window closing (window Event event) { try {
    stat.close( );
    conn.close( );}
    catch (SQL Exception e) {
    while (e! = null) {
    e.printStackTrace( );
    e = e.getNextException( ); } } }
    /** Executes the selected query private void execute Query( ) {
    Resultset rs = null;
    try {
    string author= (string) author.get selected item( );
    -----
    -----
    -----
```

},

Q. 4. (a) Differentiate between skeleton and stub.

Ans. Skeleton and stub :

A stub is a proxy for a remote object that runs on the client computer. A skeleton is a proxy for a remote object that runs on the server. Stubs forward a client's remote method invocations (and their associated arguments) to skeletons, which forward them on to the appropriate server objects. Skeletons return the results of server method invocations to clients via stubs. The stub is a piece of code that implements an interface to a remote object or service in the address space of a client of that service. The job of the stub is to open up a communication channel to the server, convert all the arguments to be sent to the server into a form that can be transmitted across the wire, and dispatch those converted arguments. The stub code then waits for the response from the server, converting any return values from their wire representation to the internal form used in the process, and handing those results back to the program or thread that made the call.

The skeleton code provides similar functionality on the side of the server. The skeleton code receives the information transmitted by a stub, converts the information that has been transmitted over the network into a form that can be understood by the server program, and makes the appropriate up-call to that server program. The server program will return any result values to the stub code, which will translate those results into a form that can be transmitted over the wire, and send them back to the calling client. The code for the stub and the skeleton is produced by a compiler that takes as input, a definition of the interface between the client and the service written in some programming language-neutral declarative language (which, seemingly no matter what its form is, called IDL). These compilers produce source code (sometimes in various languages) that can be compiled by the native compilers for the machines on which the client or service is going to run. The stub and skeleton can then be linked (either statically or at run time) into the client and the service. A client, in such a system, uses a single stub to communicate with any service that implements a particular interface, and a service uses a single skeleton to talk to any client that is calling it through a particular interface. This works because the stubs and skeletons produced by the IDL compiler all correspond with a single wire protocol that is defined as part of the overall RPC system. This gives such systems their language- and processor-independence. It doesn't matter what the environment of the client is, or the environment of the server. The system defines what is on the wire, and each system, both client and server, understands that protocol in a way that is appropriate to the language and environment running on the client or the server.

Q. 4. (b) What is distributed computing? Discuss it with reference to Remote Method Invocation.

Ans. Distributed Computing :

Distributed computing deals with hardware and software systems containing more than one processing element or storage element, concurrent processes, or multiple programs, running under a loosely or tightly controlled regime. In distributed computing a program is split up into parts that run simultaneously on multiple computers communicating over a network. Distributed computing is a form of parallel computing, but parallel computing is most commonly used to describe program parts running simultaneously on multiple processors in the same computer. Both types of processing require dividing a program into parts that can run simultaneously, but distributed programs often must deal with heterogeneous environments, network links of varying latency.

and unpredictable failures in the network or the computers.

Java Remote Method Invocation (RMI) allows you to write distributed objects using Java. This paper describes the benefits of RMI, and how you can connect it to existing and legacy systems as well as to components written in Java. RMI provides a simple and direct means for distributed computation with Java objects. These objects can be new Java objects, or can be simple Java wrappers around an existing API. Java embraces the "Write Once, Run Anywhere" model. RMI extends the Java model to be run everywhere. "Because RMI is centered around Java, it brings the power of Java safety and portability to distributed computing. You can move behavior, such as agents and business logic, to the part of your network where it makes the most sense. When you expand your use of Java in your systems, RMI allows you to take all the advantages with you. RMI connects to existing and legacy systems using the standard Java native method interface JNI. RMI can also connect to existing relational database using the standard JDBC package. The RMI/JNI and RMI/JDBC combinations let you use RMI to communicate today with existing servers in non-Java languages, and to expand your use of Java to those servers when it makes sense for you to do so. RMI lets you take full advantage of Java when you do expand your use.

Server-defined policy :

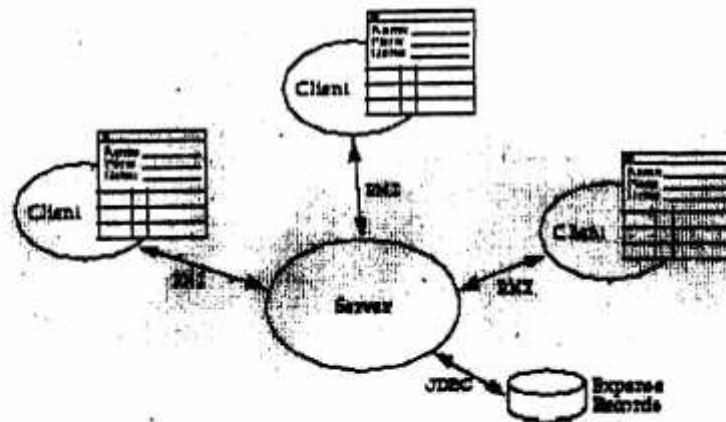


Figure 1 shows the general picture of such a dynamically configurable expense reporting system. A client displays a GUI (graphical user interface) to a user, who fills in the fields of the expense report. Clients communicate with the server using RMI. The server stores the expense reports in a database using JDBC, the Java relational database package. So far this may look like any multi-tier system, but there is an important difference-RMI can download behavior.

Q. 5. (a) Differentiate between Heavy weight and Light weight components.

Ans. Heavy weight components and light weight components :

A heavyweight component is the one which related to its own native screen resource. Where as a lightweight component is the one which "borrows" the screen resource of an ancestor. There are quite a few vital differences among lightweight and heavyweight components, because all the AWT components are heavyweight and nearly all the Swing component are lightweight, these dissimilarities turn evident when you start putting together Swing components along with AWT components. A lightweight component may have translucent pixels, where as a heavyweight ones are generally opaque. A lightweight component may seem to come into view as non-rectangular normally because of their aptitude to set translucent areas, where as a heavyweight is restricted to be rectangular.

Q. 5. (b) Discuss various swing components with implementation :

- (i) Lists
- (ii) Trees
- (iii) Styled Text Components.

Ans. (i) List :

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.DefaultListSelectionModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

public class SimpleList2 extends JPanel {
```

/

```
String label[] = { "Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine",  
"Ten", "Eleven" };  
  
JList list;  
  
public SimpleList2() (  
    setLayout(new BorderLayout());  
  
    list = new JList(label);  
  
    JButton button = new JButton("Print");  
  
    JScrollPane pane = new JScrollPane(list);  
  
    DefaultListSelectionModel m=new DefaultListSelectionModel( );  
  
    m.setSelectionMode(ListSelectionModel.SINGLE_SELECTION  
    m.setLeadAnchorNotificationEnabled(false);  
  
    list.setSelectionModel(m);  
  
    list.addListSelectionListener(new ListSelectionListener() (  
        public void valueChanged(ListSelectionEvent e) {  
            system.out.println(e.toString()); } });  
  
    button.addActionListener(new PrintListener( ) );  
  
    add (pane, BorderLayout.NORTH);  
  
    add (button, BorderLayout.SOUTH); }  
  
    public static void main(String s[]) {  
        JFrame frame = new JFrame("List Example");  
  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        frame.setContentPane(new SimpleList2() );  
  
        frame.pack();  
  
        frame.setVisible(true); }  
  
    // An inner class to respond to clicks on the Print button
```



```
class PrintListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        int selected[] = list.getSelectedIndices();  
        System.out.println("Selected Elements: ");  
        for (int i = 0; i < selected.length; i++) {  
            String element = (String) list.getModel().getElementAt(selected[i]);  
            System.out.println(" " + element);  
        }  
    }  
}
```

Trees :

```
import java.awt.BorderLayout;  
import java.awt.Container;  
import javax.swing.JFrame;  
import javax.swing.JScrollPane;  
import javax.swing.JTree;  
  
public class TreeSample {  
    public static void main(String args[]) {  
        JFrame f = new JFrame("JTree Sample");  
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Container content = f.getContentPane();  
        JTree tree = new JTree();  
        JScrollPane scrollPane = new JScrollPane(tree);  
        content.add(scrollPane, BorderLayout.CENTER);  
        f.setSize(300, 200);  
        f.setVisible(true);  
    }  
}
```

(iii) **Styled text components :**

```
import javax.swing.*;

import javax.swing.text.*;

import java.awt.*;

public class StylesExample1 {

    public static void main(String[] args) {

        try { UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLook
        .AndFeel");    } catch (Exception evt) {}

        JFrame f = new JFrame("Styles Example 1");

        // Create the StyleContext, the document and the pane

        StyleContext sc = new StyleContext();

        final DefaultStyledDocument doc = new
        DefaultStyledDocument(sc);

        JTextPane pane = new JTextPane(doc);

        // Create and add the style

        final Style heading2Style = sc.addStyle("Heading2", null);

        heading2Style.addAttribute(StyleConstants.Foreground, Color.red);

        heading2Style.addAttribute(StyleConstants.FontSize, new Integer(16));

        heading2Style.addAttribute(StyleConstants.FontFamily, "serif");

        heading2Style.addAttribute(StyleConstants.Bold, new Boolean(true));

        try { SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                try {

                    // Add the text to the document
```

```
doc.insertString(0, text, null);
// Finally, apply the style to the heading
doc.setParagraphAttributes(0, 1, heading2Style, false);
} catch (BadLocationException e) {} } } ;
} catch (Exception e) {
System.out.println("Exception when constructing document: " + e);
System.exit(1); }

f.getContentPane().add (new JScrollPane(pane));
f.setSize(400, 300);
f.setVisible(true); public static final String text = "Attributes, style s and Style Contexts\n"
+ "The simple PlainDocument class that you saw in the previous"
+ "chapter is only capable of holding text. The more complex text" + "
components use a more sophisticated model that implements the" + "Styled
Document interface. StyledDocument is a sub-interface of "
+ "Document that contains methods for manipulating attributes that"
+ "control the way in which the text in the document is displayed. "
+ "The Swing text package contains a concrete implementation of "
+ "StyledDocument called DefaultStyledDocument that is used as the"
+ "default model for JTextPane and is also the base class from which"
+ "more specific models, such as the HTMLDocument class that handles "
+ "input in HTML format, can be created. In order to make use of "
+ "DefaultStyledDocument and JTextPane, you need to understand how"
+ "Swing represents and uses attributes.\n";}
```


Q. 6. (a) Write a program to create three text fields in which the numbers are entered using Action Listener and multiplication of such numbers is printed on screen?

Ans.

```
import java.applet. Applet;

import java.awt.*; import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class multiplier extends Applet{

    Label First = new label( );

    int columnNumber, rowNumber;

    int idx = 0;

    String labels[ ];

    TextField tf1 = new TextField( );

    TextField tf2 = new TextField( );

    private TextArea txtarea = new TextArea( );

    public void init( ){

        Panel p == new Panel( );

        p.setLayout(new BorderLayout());

        p.add(tf1, BorderLayout. NORTH);

        p.add(tf2, BorderLayout.SOUTH);

        Panel p2 = new Panel( );

        p2.setLayout(new GridLayout(9,9));

        p.add (p2, BorderLayoutCENTER);

        add(p2);

        for(rowNumber=1; rowNumber <=9; row Number ++ ) {

            // System.outprint(rowNumber + "\t ");

            /
```

```
f for( column Number2; columnNumber<=:9; columnNumber + "\t ");  
  
// System.outprint(columnNumber*rowNumber + "\t ");  
  
Labels [idx++]. setText("") +( column Number*row Number");  
  
// System. out. print("\n");}}}
```

Q. 6. (b) Discuss Readers and writes for image. What is Image Manipulation?

Ans. Prior to version 1.4, JDK had very limited capabilities for reading and writing image file. It was possible to read GIF and JPEG image, but there was no official support for writing image at all. This situation is now much improved. JDK 1.4 introduces the `javax.image io` package that contains out of box support for reading and writing several common file formats as well as framework that enables third parties to add readers and writers for other formats. Specifically, JDK contains readers for GIF, JPEG and PNG formats and writers for JPEG and PNG. The basic of the library are extremely straight forward. To load an image, use the static read method of image IO class :

```
file f = -----  
  
Buffered image = Image IO.read (f)
```

The image IO class picks an appropriate reader, based on the file type. It may consult the file extension and the "magic number" at the beginning of the file for that purpose. If no suitable reader can be found or the reader can't decode the file contents then the read method returns null.

Writing an image to a file is just as simple :

```
file f = ----- ;  
  
string format = -----;  
  
Image IO.write (Image, format, f);
```

Here the format string is a string identifying the image format, such as "JPEG" or "PNG". The appropriate image IO class picks an appropriate writer and save the file.

Q. 7. (a) What are Java Beans? How they facilitate component oriented software construction?

Ans. Java beans :

The Java Beans™ architecture is based on a component model which enables developers to create software units called components. Components are self-contained, reusable software units that can be visually assembled into composite components, applets, applications, and servlets using visual application builder tools. JavaBean components are known as beans. A set of APIs describes a component model for a particular language. The JavaBeans API specification describes the core detailed elaboration for the Java Beans component architecture. Beans are dynamic in that they can be changed or customized. Through the design mode of

a builder tool you can use the Properties window of the bean to customize the bean and then save (persist) your beans using visual manipulation. You can select a bean from the toolbox, drop it into a form, modify its appearance and behavior, define its interaction with other beans, and combine it and other beans into an applet, application, or a new bean. The following list briefly describes key bean concepts :

- * Builder tools discover a bean's features (that is, its properties, methods, and events) by a process known as introspection. Beans support introspection in two ways:

- By adhering to specific rules, known as design patterns, when naming bean features. The Interoceptor class examines beans for these design patterns to discover bean features. The Interoceptor class relies on the core reflection API. The trail The Reflection API is an excellent place to learn about reflection.

- By explicitly providing property, method, and event information with a related bean information class. A bean information class implements the Baneful interface. A Beaninfo class explicitly lists those bean features that are to be exposed to application builder tools.

- * Properties are the appearance and behavior characteristics of a bean that can be changed at design time. Builder tools introspect on a bean to discover its properties and expose those properties for manipulation.

- * Beans expose properties so they can be customized at design time. Customization is supported in two ways: by using property editors, or by using more sophisticated bean customizers.

- * Beans use events to communicate with other beans. A bean that is to receive events (a listener bean) registers with the bean that fires the event (a source bean). Builder tools can examine a bean and determine which events that bean can fire (send) and which it can handle (receive). Persistence enables beans to save and restore their state. After changing a bean's properties, you can save the state of the bean and restore that bean at a later time with the property changes intact. The JavaBeans architecture uses Java Object Serialization to support persistence.

A bean's methods are no different from Java methods, and can be called from other beans or a scripting environment. By default all public methods are exported.

Beans vary in functionality and purpose. You have probably met some of the following beans in your programming practice:

- * GUI (graphical user interface)
- * Non-visual beans, such as a spelling checker Animation applet
- * Spreadsheet application.

Q. 7. (b) Discuss the Bean writing process in detail.

Ans. Bean writing process :

The simplest kind of bean is really nothing more than a Java class that follows some fairly strict naming

conventions for its methods. Following example gives the view of ban.

```
import java.awt.*;
import java.io.*;
import java.awt.image.*;
import java.awt.swing.*;

public class Image Viewer Bean extends JComponent
{
    public Image Viewer Bean() {
        setBorder(BorderFactory.createEtchedBorder());
    }

    public void setFilename(String filename)
    { try {
        File file = new File(filename);
        setIcon(new ImageIcon(ImageIO.read(file)));
    } catch (IOException e)
    {
        file = null;
        setIcon(null); } }

    public String getFileName() {
        if (file == null) return null;
        else return file.getPath();
    }

    public Dimension getPreferredSize() {
        return new Dimension(X_PREFERRED_SIZE, Y_PREFERRED_SIZE);
    }
}
```

```
}  
  
Private file file=null;  
  
private static final int XPREFSIZE = 200;  
  
private static final int YPREFSIZE = 200;}
```

Q. 8. Write short notes on (any three) :

- (i) **Class Loader**
- (ii) **Byte Code Verification**
- (iii) **Digital Signature**
- (iv) **Encryption.**

Ans. (i) Class loader :

The class loader concept, one of the cornerstones of the Java virtual machine, describes the behavior of converting a named class into the bits responsible for implementing that class. Because class loaders exist, the Java run time does not need to know anything about files and file systems when running Java programs. Classes are introduced into the Java environment when they are referenced by name in a class that is already running. There is a bit of magic that goes on to get the first class running (which is why you have to declare the main() method as static, taking a string array as an argument), but once that class is running, future attempts at loading classes are done by the class loader. At its simplest, a class loader creates a flat name space of class bodies that are referenced by a string name. The method definition is: `Class r = loadClass(String className, boolean resolve)`.

(ii) Byte code verification :

Byte code verification is a crucial security component for Java applets, on the Web and on embedded devices such as smart cards. This paper reviews the various bytecode verification algorithms that have been proposed, recasts them in a common framework of dataflow analysis, and surveys the use of proof assistants to specify bytecode verification and prove its correctness. One way to guarantee these conditions is to check them dynamically, while executing the bytecode. This is called the "defensive JVM approach" in the literature. However, checking these conditions at run-time is expensive and slows down execution significantly. The purpose of bytecode verification is to check these conditions once and for all, by static analysis of the bytecode at loading-time. Bytecode that passes verification can then be executed faster, omitting the dynamic checks for the conditions above. It must be emphasized that bytecode verification by itself does not guarantee secure execution of the code: many crucial properties of the code still need to be checked dynamically, for instance via array bounds checks and null pointer checks in the virtual machine, and access control checks in the API. The purpose of bytecode verification is to shift the verifications listed above from run-time to loading-time.

(iii) Digital signature :

A digital signature or digital signature scheme is a type of asymmetric cryptography used to simulate the security properties of a handwritten signature on paper. Digital signature schemes consist of at least three algorithms: a key generation algorithm, a signature algorithm, and a verification algorithm. A signature provides authentication of a "message". Messages may be anything, from electronic mail to a contract, or even a message sent in a more complicated cryptographic protocol. Digital signatures are often used to implement electronic signatures, a broader term that refers to any electronic data that carries the intent of a signature, but not all electronic signatures use digital signatures. In some countries, including the United States, and in the European Union, electronic signatures have legal significance. However, laws concerning electronic signatures do not always make clear their applicability towards cryptographic digital signatures, leaving their legal importance somewhat unspecified.

(iv) Encryption :

In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (in cryptography, referred to as ciphertext). In many contexts, the word encryption also implicitly refers to the reverse process, decryption (e.g. "software for encryption" can typically also perform decryption), to make the encrypted information readable again (i.e., to make it unencrypted). Encryption has long been used by militaries and governments to facilitate secret communication. Encryption is now used in protecting information within many kinds of civilian systems, such as computers, networks (e.g. the Internet e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. Encryption is also used in digital rights management to prevent unauthorized use or reproduction of copyrighted material and in software also to protect against reverse engineering. Encryption, by itself, can protect the confidentiality of messages, but other techniques are still needed to protect the integrity and authenticity of a message; for example, verification of a message authentication code (MAC) or a digital signature.