SOLVED PAST PAPER INTERNET PROGRAMMING 2013

SHORT QUESTIONS:

Q2. What is difference between instance variable and class variable?
A: Static(**Class**) **variables** and **instance variables** both are member **variables** because they are both associated with a specific **class**, but the **difference between** them is **Class variables** only have one copy that is shared by all the different objects of a **class**, whereas every object has it's own personal copy of an **instance.**
An instance variable is a variable which has one copy per object / instance. That means every object will have one copy of it.

A class variable is a variable which has one copy per class. The class variables will not have a copy in the object.
Example :

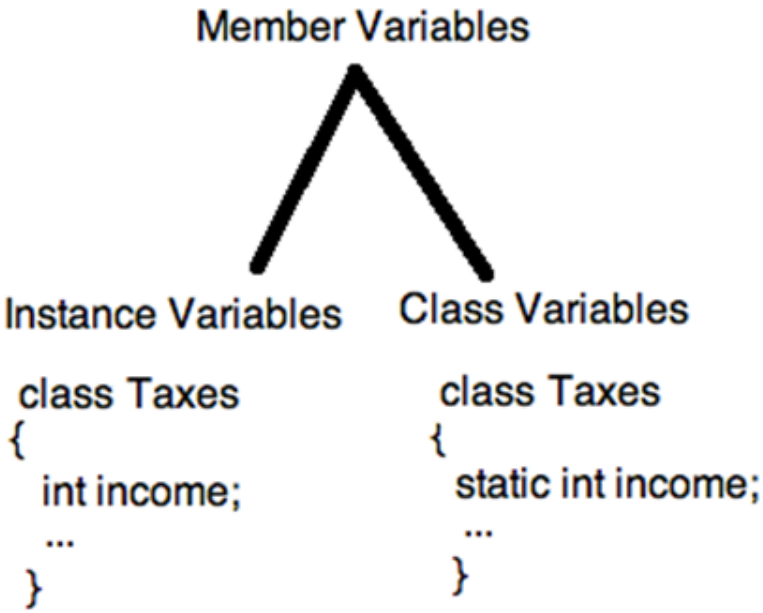
```
class Employee
{
  int empNo;
  string empName,department;
  double salary;
  static int officePhone;
}
```

OR
Class Variables In Java:
1) Class variables, also called as static variables, are declared with the keyword *static*.
?

```
class StaticVariables
{
    static int i;        //Static Variable

    static String s;     //Static Variable
}
```

Instance Variables in Java:
1) Instance variables, also called as non-static variables are declared without *static* keyword.
?

```
class InstanceVariables
{
    int i;        //Instance Variable

    String s;     //Instance Variable
}
```



Q3. What is the relationship between an event-listener interface and an event-adapter class?
A: An example of **an event** is the **Java** bean **java**.awt.Component, which raises **events when** the mouse moves over it. The **listener interface**,**java**.awt.**event**.MouseMotionListener, implements the following two methods: ... The first of these methods is always present on a **Java** bean that has bound properties. An event listener registers with an event source to receive notifications about the events of a particular type. Various event listener interfaces are defined in the java.awt.event package is given below:

Interface	Description
ActionListener	Defines the actionPerformed() method to receive and process action events.
MouseListener	Defines five methods to receive mouse events, such as when a mouse is clicked, pressed, released, enters, or exits a component
MouseMotionListener	Defines two methods to receive events, such as when a mouse is dragged or moved.
AdjustmentListner	Defines the adjustmentValueChanged() method to receive and process the adjustment events.
TextListener	Defines the textValueChanged() method to receive and process an event when the text value changes.
WindowListener	Defines seven window methods to receive events.
ItemListener	Defines the itemStateChanged() method when an item has been selected or deselected by the user.

<u>An event-adapter class:</u>
Adapter is a pattern that provides default (often empty) implementation of interface or abstract class. For example **MouseAdapter** provides empty implementation of **MouseListener** interface. It is useful because very often you do not really use all methods declared by interface, so implementing the interface directly is very verbose.
Controller is a part of MVC - Model-View-Controller pattern. No direct relation with Adapter.

Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes

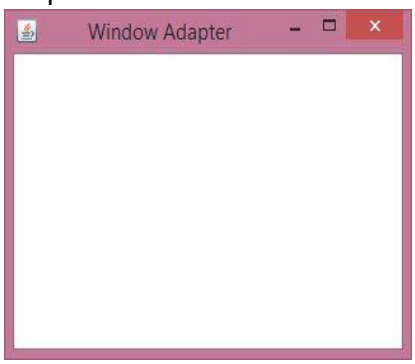
Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
Frame f;
AdapterExample(){
f=new Frame("Window Adapter");
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent e) {
        f.dispose();
}
});
```

```
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
new AdapterExample();
}
}
```

Output:



Q4: What is servlet and explain the life cycle?

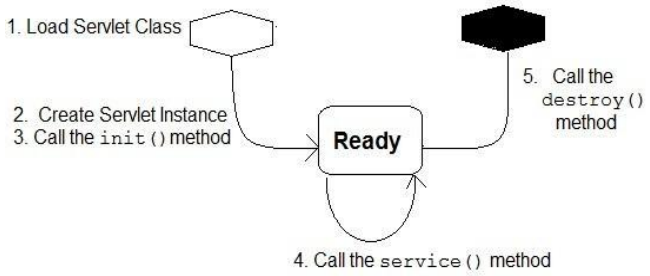
A: A Java servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers.

A **servlet** is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API.

The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

Servlet Life Cycle:



1. **Loading Servlet Class :** A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation :**After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. **Call to the init() method :** init() method is called by the Web Container on servlet instance to initialize the servlet.

   Signature of init() method :

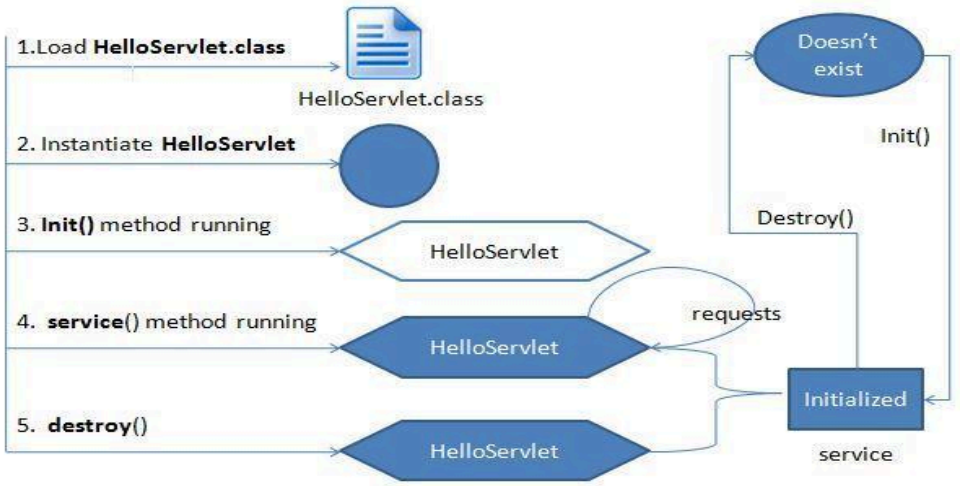
```
public void init(ServletConfig config) throws ServletException
```

4. **Call to the service() method :** The containers call the service() method each time the request for servlet is received. The service() method will then call the doGet() or doPost() methos based ont eh type of the HTTP request, as explained in previous lessons.

   Signature of service() method :

```
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException
```

5. **Call to destroy() method:** The Web Container call the destroy() method before removing servlet instance, giving it a chance for cleanup activity.



Q5. What is Java Beans?

A: A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

Simple example of java bean class:

```
//Employee.java
package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
```

}

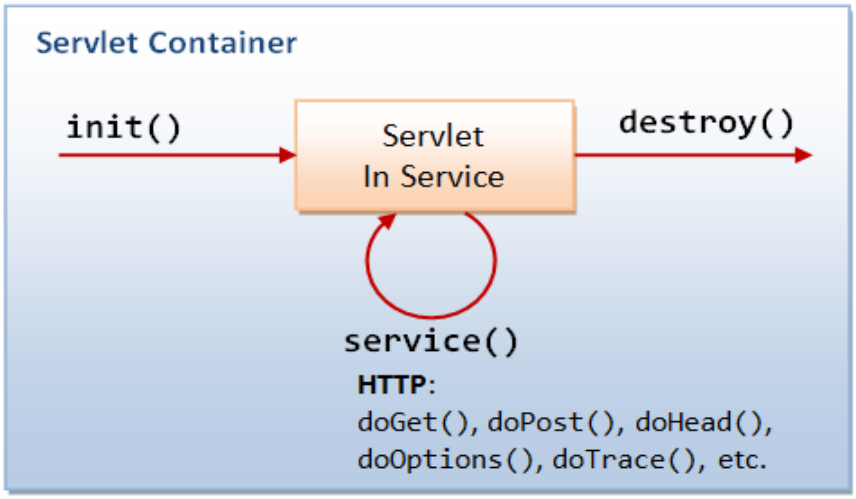
Q6: Why use JSP when we can do the same thing with servlets?
A: While JSP may be great for serving up dynamic Web content and separating content from presentation, some may still wonder why servlets should be cast aside for JSP. The utility of servlets is not in question. They are excellent for server-side processing, and, with their significant installed base, are here to stay. In fact, architecturally speaking, you can view JSP as a high-level abstraction of servlets that is implemented as an extension of the Servlet 2.1 API. Still, you shouldn't use servlets indiscriminately; they may not be appropriate for everyone. For instance, while page designers can easily write a JSP page using conventional HTML or XML tools, servlets are more suited for back-end developers because they are often written using an IDE -- a process that generally requires a higher level of programming expertise.
When deploying servlets, even developers have to be careful and ensure that there is no tight coupling between presentation and content. You can usually do this by adding a third-party HTML wrapper package like htmlKona to the mix. But even this approach, though providing some flexibility with simple screen changes, still does not shield you from a change in the presentation format itself. For example, if your presentation changed from HTML to DHTML, you would still need to ensure that wrapper packages were compliant with the new format. In a worst-case scenario, if a wrapper package is not available, you may end up hardcoding the presentation within the dynamic content. So, what is the solution? One approach would be to use both JSP and servlet technologies for building application systems.

OR

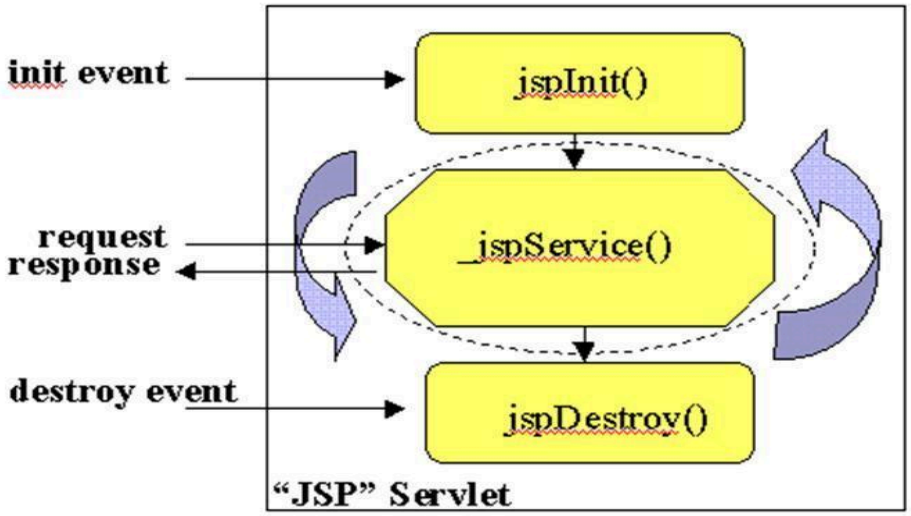
The basic difference is that, JSP is for web page and Servlet is for Java components. Servlet is HTML in Java where as JSP is Java in HTML.

- Java EE Complete **Reference**
- A servlet is like any other java class. You put HTML into print statements like you use System.out or how javascript uses document.write.
- A JSP technically gets converted to a servlet but it looks more like PHP files where you embed the java into HTML.

Servlet Life Cycle



JSP Life Cycle



Difference Between Servlet and JSP

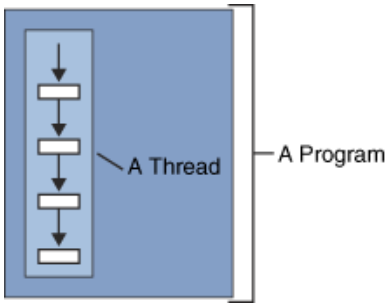
In this article we will list some of the differences between Servlets and JSP.

SERVLET	JSP
A servlet is a server-side program and written purely on Java.	JSP is an interface on top of Servlets. In another way, we can say that JSPs are extension of servlets to minimize the effort of developers to write User Interfaces using Java programming.
Servlets run faster than JSP	JSP runs slower because it has the transition phase for converting from JSP page to a Servlet file. Once it is converted to a Servlet then it will start the compilation
Executes inside a Web server, such as Tomcat	A JSP program is compiled into a Java servlet before execution. Once it is compiled into a servlet, it's life cycle will be same as of servlet. But, JSP has it's own API for the lifecycle.
Receives HTTP requests from users and provides HTTP responses	Easier to write than servlets as it is similar to HTML.
We can not build any custom tags	One of the key advantage is we can build custom tags using JSP API (there is a separate package available for writing the custom tags) which can be

SERVLET	JSP
	available as the re-usable components with lot of flexibility
Servlet has the life cycle methods init(), service() and destroy()	JSP has the life cycle methods of jspInit(), jspService() and jspDestroy()
Written in Java, with a few additional APIs specific to this kind of processing. Since it is written in Java, it follows all the Object Oriented programming techniques.	JSPs can make use of the Javabeans inside the web pages
In MVC architecture Servlet acts as controller.	In MVC architecture JSP acts as view.

Q7. What is Threads?

A: A thread is similar to the sequential programs described previously. A single thread also has a beginning, a sequence, and an end. At any given time during the runtime of the thread, there is a single point of execution. However, a thread itself is not a program; a thread cannot run on its own. Rather, it runs within a program. The following figure shows this relationship.



A thread is a single sequential flow of control within a program.

A thread, in the context of Java, is the path followed when executing a program. All Java programs have at least one thread, known as the main thread, which is created by the JVM at the program's start, when the main() method is invoked with the main thread. In Java, creating a thread is accomplished by implementing an interface and extending a class. Every Java thread is created and controlled by the java.lang.Thread class.

When a thread is created, it is assigned a priority. The thread with higher priority is executed first, followed by lower-priority threads. The JVM stops executing threads under either of the following conditions:

- The exit method has been invoked and authorized by the security manager
- All of the daemon threads of the program have died
- An Error or Exception has been thrown
- A stop() method gets called by another thread

A thread is a program's path of execution. Most programs written today run as a single thread, causing problems when multiple events or actions need to occur at the same time. Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time. The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time. Threads allow us to do this.

Java Threads Examples:
Threading is a facility to allow multiple tasks to run concurrently within a single process. Threads are independent, concurrent execution through a program, and each thread has its own stack. Extending Thread Class is required to 'override run()' method.

Q8. What is difference between Get and Post Method in HTML Form?
A: Post Method:
The method attribute specifies how to send form-data (the form-data is sent to the page specified in the action attribute). The form-data can be sent as URL variables (with method="get") or as HTTP post transaction (with method="post"). Notes on GET: Appends form-data into the URL in name/value pairs.
Get Method:
The form-data can be sent as URL variables (with method="get") or as HTTP post transaction (with method="post"). Notes on GET: Appends form-data into the URL in name/value pairs. The length of a URL is limited (about 3000 characters)
In HTML, one can specify two different submission methods for a form. The method is specified inside a FORM element, using the METHOD attribute. The difference between METHOD="GET" (the default) and METHOD="POST" is primarily defined in terms of form data encoding. The official recommendations say that "GET" should be used if and only if the form processing is idempotent, which typically means a pure query form. Generally it is advisable to do so. There are, however, problems related to long URLs and non-ASCII character repertoires which can make it necessary to use "POST" even for idempotent processing.

The GET Method:
In GET method the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&). In general, a URL with GET data will look like this:
http://www.example.com/action.php?**name=***john*&**age=***24*
The bold parts in the URL are the GET parameters and the italic parts are the value of those parameters. More than one parameter=value can be embedded in the URL by concatenating with ampersands (&). One can only send simple text data via GET method.
The POST Method:
In POST method the data is sent to the server as a package in a separate communication with the processing script. Data sent through POST method will not visible in the URL.

The POST method has no size limit

In the form, it is specified as follows:

```
<form method="post" action="page.php">
</form>
```

This method sends a header and a body message to the server. The body usually consists of data entered into the form fields by the user.

The form data do not appear in the URL. Accordingly, it is not possible to retrieve data directly in JavaScript, you should add some PHP code in the page:

```
<?php
 $color = $_POST['color'];
 $shape = $_POST['shape'];
?>
...  HTML code ...
```

You can however assign data retrieved through PHP to a JavaScript script:

```
<script>
 var color = <?php echo $color;?>;
 var shape = <?php echo $shape;?>;
</script>
```

Q9: Explain the Polymorphism?

A: Polymorphism is the ability of an object to take on many forms. The most common use of **polymorphism** in OOP occurs when a parent class reference is used to refer to a child class object. ... In **Java**, all **Java** objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

Polymorphism in Java is closely associated with the principle of inheritance. The term "polymorphic" means "having multiple forms." Polymorphism in Java simplifies programming by providing a single interface overlaid with multiple meanings as it goes through the rigor of sub classing.

Polymorphism in java is a concept by which we can perform a *single action by different ways*.

Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Q10. What is Exception?

A: An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. Exceptions are events that occur during the execution of programs that disrupt the normal flow of instructions (e.g. divide by zero, array access out of bound, etc.). In Java, an exception is an object that wraps an error event that occurred within a method and contains: Information about the error including its type.

- Exceptions are events that occur during the execution of programs that disrupt the normal flow of instructions (e.g. divide by zero, array access out of bound, etc.).

- In Java, an exception is an **object** that wraps an error event that occurred within a method and contains:
  - Information about the error including its type
  - The state of the program when the error occurred
  - Optionally, other custom information
- Exception objects can be *thrown* and *caught*.
- Exceptions are used to indicate many different types of error conditions.
- JVM Errors:
  - OutOfMemoryError
  - StackOverflowError
  - LinkageError
- System errors:
  - FileNotFoundException
  - IOException
  - SocketTimeoutException
  - Programming errors:
  - NullPointerException
  - ArrayIndexOutOfBoundsException
  - ArithmeticException

Q11. Define Packages in Java and why these are used?

A: PACKAGE in Java is a collection of classes, sub-packages, and interfaces. It helps organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve code reusability. A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another, and scripts or applications in yet another. Because software written in the Java programming language can be composed of hundreds or thousands of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

- Java packages can be stored in compressed files called JAR files, allowing classes to **download** faster as a **group** rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

OR

A package as the name suggests is a pack(group) of classes, interfaces and other packages. In java we use packages to organize our classes and interfaces. We have two **types of packages in Java**: built-in packages and the packages we can create (also known as user defined package). In this guide we will learn what are packages, what are user-defined packages in java and how to use them.

In java we have several built-in packages, for example when we need user input, we import a package like this:

```
import java.util.Scanner
```

Here:

→ **java** is a top level package

→ **util** is a sub package

→ and **Scanner** is a class which is present in the sub package **util**.

LONG QUESTIONS:

Q.12: What is an Applet? Explain its working and give an example of it.

A: An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

- An Applet class does not have any main() method.
- It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.
- JVM creates an instance of the applet class and invokes init() method to initialize an Applet.

An Applet Skeleton

Most applets override these four methods. These four methods forms Applet lifecycle.

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

Design applet program

We can design our own applet program by **extending applet class** in the user defined class.

Syntax:

```
class className extends Applet

{

  ......


  // override lifecycle methods


......

  }
```

Running of applet programs

Applet program can run in two ways.

1. **Using html** (in the web browser)
2. **Using applet viewer tool** (for testing purpose)

1. Running of applet using html

Html support a predefined tag called <applet> to load the applet program on the browser window.

The <applet> Tag

Syntax

<applet code="class-file" width=size height=size>

</applet>

- The applet tag has several attributes; currently we focus only three important attributes:

- code
- width
- height

- With code attribute, you will assign class name where the applet resides, and width and height represent the size of the JApplet in pixels.

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6. public void paint(Graphics g){
7. g.drawString("welcome",150,150);
8. }
9.
10.}
```

myapplet.html

```
1. <html>
2. <body>
3. <applet code="First.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Running of applet using appletviewer

Some browser does not support <applet> tag so that Sun MicroSystem was introduced a special tool called appletviewer to run the applet program.

In this Scenario java program should contains <applet> tag in the commented lines so that appletviewer tools can run the current applet program.

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
1. //First.java

2. import java.applet.Applet;
```

```
3. import java.awt.Graphics;

4. public class First extends Applet{

5.
6. public void paint(Graphics g){

7. g.drawString("welcome to applet",150,150);

8. }

9.

10. }

11. /*

12. <applet code="First.class" width="300" height="300">

13. </applet>

14. */
```

15. To execute the applet by appletviewer tool, write in command prompt:

16. **c:\>**javac First.java

17. **c:\>**appletviewer First.java

Output in Applet viewer

Q.13: Explain any four methods of String class with at least one example.

A: **1: charAt()**

charAt() function returns the character located at the specified index.Example:

```
String str = "studytonight";
System.out.println(str.charAt(2));
```

Output: u

2: length()

length() function returns the number of characters in a String.

```
String str = "Count me";
System.out.println(str.length());
```

Output: 8

3: toLowerCase()

toLowerCase() method returns string with all uppercase characters converted to lowercase.

```
String str = "ABCDEF";
System.out.println(str.toLowerCase());
```

Output: abcdef

4: toUpperCase()

This method returns string with all lowercase character changed to uppercase.

```
String str = "abcdef";
System.out.println(str.toUpperCase());
```

Output: ABCDEF

Q.14: Write a program that will take following information from the user through GUI.

1. Student Name
2. CGPA
3. Semester

The GUI will have the following buttons

1- Save 2 - Cancel

If user press the save button, the information will be added in the file. The new information should be added to existing data in the file. On cancel, application should terminate.

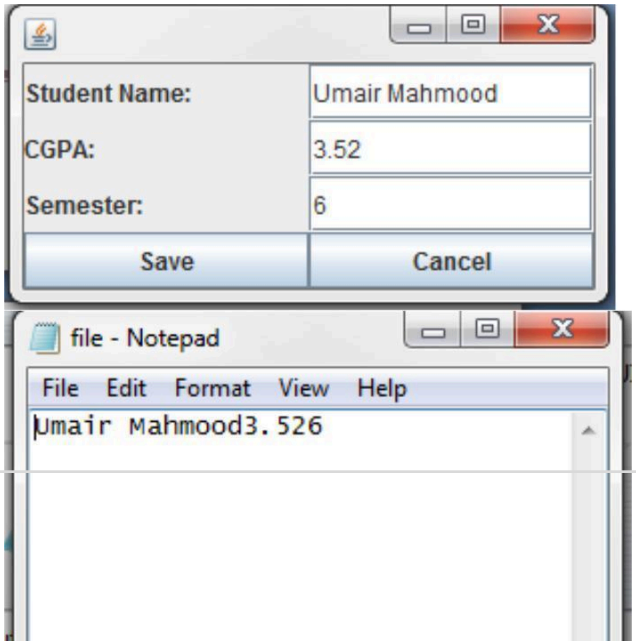
A: // Step 1: import packages

```
import java.awt.*;
import javax.swing.*;

import java.awt.event.*;
import java.io.*;
```

```
public class GUITest implements ActionListener{
JFrame myFrame;
JButton b1;
JButton b2;
JTextField t1;
JTextField t2;
JTextField t3;


public void initGUI()
{

Step 2: set the top level container myFrame=new JFrame();

Step 3: Get the component area of top-level Container(JFrame) Container c=myFrame.getContentPane();

Step 4: Apply layouts
c.setLayout(new GridLayout(4,2));

Step 5: creating GUI components JLabel l1=new JLabel("Student Name:"); JLabel l2=new
JLabel("CGPA:"); JLabel l3=new JLabel("Semester:"); b1=new JButton("Save");

b2=new JButton("Cancel"); t1=new JTextField(); t2=new JTextField(); t3=new JTextField();

b1.addActionListener(this);
b2.addActionListener(new CancelBtn());

Step 6: adding components to container c.add(l1);
c.add(t1);

c.add(l2);
c.add(t2);

c.add(l3);
c.add(t3);

c.add(b1);

c.add(b2);

Step 7: set size of frame and make it visible
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); myFrame.setSize(300,150);

myFrame.setVisible(true); } // end of initGUI method

constructor of class public GUITest()
{
```

```
initGUI();

}

public void actionPerformed(ActionEvent event){
try{
String line;
FileWriter fw=new FileWriter("file.txt",true);
PrintWriter pw=new PrintWriter(fw);

line=t1.getText()+t2.getText()+t3.getText();
pw.println(line);

pw.flush();

pw.close();
fw.close();
}catch(IOException ex){
System.out.println(ex);
}

}
class CancelBtn implements ActionListener{
public void actionPerformed(ActionEvent event){
System.exit(0);

}
}
public static void main(String args[])
{
GUITest gt=new GUITest();

}

}
```

OUTPUT OF THE PROGRAM:



When u again use this program, the new information will be added to existing data in the file.

Regards: MUDDASIR RASHEED