Q.2a. The signature for *main()* function in every Java application program is*public static void main(String args[])*
What is the role of keyword *'static' in the* above declaration? What happens if we don't write keyword *static* in the above declaration?

Answer
The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class. This is necessary since **main( )** is called by the Java interpreter before any objects are made.
If we omit the keyword static from the main function declaration, the program will compile but there will be runtime error such as *"Exception in thread "main" java.lang.NoSuchMethodError : Main"*.

b .Write a program to find the number of and sum of all integers greater than 100 and less than 200 that are divisible by 7.

Answer
```
public class SumOfNumber {

    public static void main(String args[]) {
      int i;
      int sum = 0;
      int count = 0;
      System.out.println("Program to find the sum of all numbers
      between 100 and 200 which are divisible by 7");

      for (i=100; i<200; i++){
            if ( i % 7 == 0){

                  sum = sum + i;
                  count++;
            }
      }
      System.out.println("Number of INTEGERS divisible by 7 are :
            " + count);
      System.out.println("Required Sum is : " + sum);
    }
}
```

c. Explain the syntax of *for* loop. Also, describe with the help of an example, the *enhanced for* loop.

Answer

The ***for*** loop is entry controlled loop that provides a more concise loop control structure. The syntax of the *for* loop is as follows:

```
for (initialization; test_condition; increment) {
      // body of loop
}
```

The execution of the ***for*** statement is as follows:

1. *Initialization* of control variables is done first, using assignment statements such as i=1. The variable **i** is known as loop-control variable.
2. The value of the control variable is tested using the *test_condition*. The test condition is a relational expression, such as i < 10 that determines when the loop will exit. If the condition is true, the body of the loop will executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed, the control is transferred back to the **for** statement after evaluating the last statement in the loop. Now, the control variable is *incremented* using assignment statement such as *i = i + 1* (or *i++*) and the new value of the control variable is again tested to test whether it satisfies the loop condition. It the condition satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

Consider the following program segment:

```
for (i = 0; i < 10; i++) {
      System.out.println(i);
}
```

This **for** loop is executed 10 times and prints the digit 0 to 9 in one line. The three sections enclosed within parenthesis must be separated by semicolons. But there is no semicolon at the end of the increment section.

The **for** statement allows for *negative increments*. For example, the loop can be written as:

```
for (i = 9; i >= 0; i=i-1)
      System.out.println(i);
```

This loop is also executed for 10 times, but the output would be from 9 to 0. The braces are optional when the body of the loop contains only one statement.

Since the conditional test is always performed at the beginning of the lop, the body of the loop may not be executed at all, if the condition fails at the start. For example,

```
for (i = 10; i < 10; i++) {
      System.out.println(i);
}
```

will never be executed because the test condition fails at the very beginning itself.

<u>Enhanced for loop</u>

The enhanced **for** loop, also called *for each* loop, is an extended language feature introduced with the J2SE 5.0 release. This feature helps us to retrieve the array elements efficiently rather than using array indexes. We can also use this feature to eliminate the iterators in a for loop and to retrieve the elements from a collection. The enhanced for loop takes the following form:

```
for (Type Identifier : Expression) {
      // statements;
}
```

where, *Type* represents the data type or object used; *Identifier* refers to the name of a variable; and

Expression is an instance of the java.lang.Iterable interface or an array.
For example, consider the following statements:

```
int numArr[3] = {56, 48, 79};
for (int i=0; i<3; i++) {
      if (numArr[i]>50 && numArr[i]<100) {
            System.out.println("Value is " + numArr[i]);
      }
}
```

which is equivalent to the following code:

```
int numArr[3] = {56, 48, 79};
for (int i : numArr) {
      if (i>50 && i<100) {
            System.out.println("Value is " + i);
      }
}
```

Thus, we can use the enhanced for loop to track the elements of an array efficiently. In the same manner, we can track the collection elements using the enhanced for loop as follows:

```
Stack ss = new Stack();
ss.push(new Integer(56));
ss.push(new Integer(48));
ss.push(new Integer(79));
for(Object ob : ss) {
      System.out.println(ob);
}
```

Q.3a. Explain the use of '**final'** keyword in respect of inheritance.

Answer

<u>Final method is used to Prevent Overriding</u>

While method overriding is one of Java's most powerful features, there will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify **final** as a modifier at the start of its declaration. Methods declared as **final** cannot be overridden. The following fragment illustrates **final**:

```
class A {
      final void meth() {
            System.out.println("This is a final method.");
      }
}

class B extends A {
      void meth() { // ERROR! Can't override.
            System.out.println("Illegal!");
      }
}
```

Because **meth( )** is declared as **final**, it cannot be overridden in **B**. If you attempt to do so, a compile-time error will result.

Methods declared as **final** can sometimes provide a performance enhancement: The compiler is free to

inline calls to them because it "knows" they will not be overridden by a subclass. When a small **final** method is called, often the Java compiler can copy the bytecode for the subroutine directly inline with the compiled code of the calling method, thus eliminating the costly overhead associated with a method call. Inlining is only an option with **final** methods. Normally, Java resolves calls to methods dynamically, at runtime. This is called *late binding*. However, since **final** methods cannot be overridden, a call to one can be resolved at compile time. This is called *early binding*.

Final class is used to Prevent Inheritance

Sometimes you will want to prevent a class from being inherited. To do this, precede the class declaration with **final**. Declaring a class as **final** implicitly declares all of its methods as **final**, too. As you might expect, it is illegal to declare a class as both **abstract** and **final** since an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementations.
Here is an example of a **final** class:

```
final class A {
      // ...
}

// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
      // ...
}
```

As the comments imply, it is illegal for **B** to inherit **A** since **A** is declared as **final**.

b.Under what circumstances the compiler insists that a class must be declared as an abstract class? Can an abstract class be declared as final also?)

Answer

The compiler insists that a class must be declared as an abstract class if any of the following conditions is true:

(i) the class has one or more abstract methods
(ii) the class inherits one or more abstract methods (from an abstract parent) for which it does not provide implementation.
(iii) The class declares that it implements an interface but does not provide implementations for every method of that interface.

Thus, according to these three conditions, the abstract class is in some sense is incomplete. Thus, this class must be subclassed. But a class declared as final class can't be subclassed. Hence, a class cannot be declared as abstract and final simultaneously.

c.Define Interface. How it is similar and different from a class? Can an interface be extended? If yes, explain with the help of example.

Answer

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements. Unless the class that implements the interface is

abstract, all the methods of the interface need to be defined in the class.

The general form of an interface:

```
access interface name {
      return-type method-name1(parameter-list);
      return-type method-name2(parameter-list);
      type final-varname1 = value;
      type final-varname2 = value;
            // ...
      return-type method-nameN(parameter-list);
      type final-varnameN = value;
}
```

Here, *access* is either **public** or not used. When no access specifier is included, then default access results, and the interface is only available to other members of the package in which it is declared. When it is declared as **public**, the interface can be used by any other code. *name* is the name of the interface, and can be any valid identifier. The methods which are declared within the interface have no bodies. They end with a semicolon after the parameter list. They are, essentially, abstract methods; there can be no default implementation of any method specified within an interface. Each class that includes an interface must implement all of the methods.
Variables can be declared inside of interface declarations. They are implicitly **final** and **static**, meaning they cannot be changed by the implementing class. They must also be initialized with a constant value. All methods and variables are implicitly **public** if the interface, itself, is declared as **public**.
Here is an example of an interface definition. It declares a simple interface which contains one method called **callback( )** that takes a single integer parameter.

```
interface Callback {
      void callback(int param);
}
```

An interface is similar to a class in the following ways:

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The bytecode of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

An interface is different from a class in several ways, such as:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

One interface can inherit another by use of the keyword **extends**. The syntax is the same as for inheriting

classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain. Following is an example:

```
// One interface can extend another.
      interface A {
            void meth1();
            void meth2();
      }

// B now includes meth1() and meth2() -- it adds meth3().
      interface B extends A {
            void meth3();
      }
// This class must implement all of A and B

      class MyClass implements B {
            public void meth1() {
                  System.out.println("Implement meth1().");
            }
            public void meth2() {
                  System.out.println("Implement meth2().");
            }
            public void meth3() {
                  System.out.println("Implement meth3().");
            }
      }

      class IFExtend {
            public static void main(String arg[]) {
                  MyClass ob = new MyClass();
                  ob.meth1();
                  ob.meth2();
                  ob.meth3();
            }
      }
```

If you might want to try removing the implementation for **meth1( )** in **MyClass**. This will cause a compile-time error. Any class that implements an interface must implement all methods defined by that interface, including any that are inherited from other interfaces.

Q4 (a) With the help of a suitable figure describe the life cycle of a thread?

Answer

During the life time of a thread, there are many states it can enter. They include:

- Newborn state
- Runnable state
- Running state
- Blocked state
- Dead state

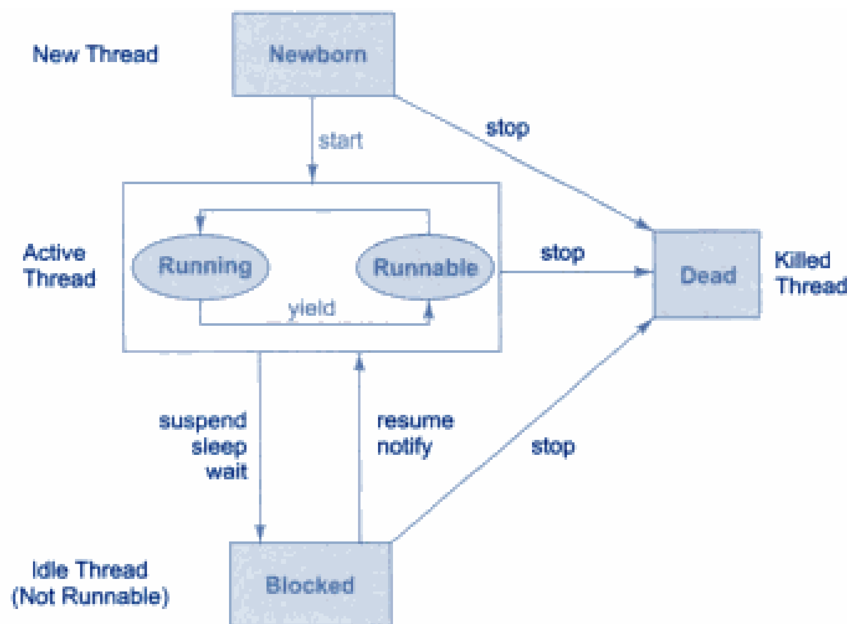
A thread is always in one of these five states. It can move from one state to another via a variety of ways as shown in following figure:

Newborn state

When we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state, we can do only one of the following things with it:

- Schedule it for running using **start()** method.
- Kill it using **stop()** method.

If scheduled, it moves to the runnable state. If we attempt to use any other method at this stage, an exception will be thrown.



State transition diagram of a thread

Runnable state

The runnable state means that the thread is ready for execution and in waiting for the availability of the processor. That is, the thread has joined the queue o0f threads that are waiting for execution. If all threads have equal priority, then they are given equal time slots for execution in round robin fashion, i.e., first-come, first-serve manner. The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as time-slicing.

But, if we want a thread to relinquish control to another thread to equal priority before its turn comes, we can do so by using the **yield()** method.

Running state

Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is pre-empted by a higher priority thread. A running thread may relinquish its control in one of the following situations:

1. It has been suspended using **suspend()** method. A suspended thread can be revived by using the **resume()** metho0d. This approach is useful when we want to suspend a thread for sometime due to certain reasons, but do not want to kill it.
2. It has been made to sleep. We can put a thread to sleep for a specified time period using the

method **sleep(time)** where time is in milliseconds. This means that the thread is out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.

3. It has been told to wait until some event occurs. This is done using the **wait()** method. The thread can be scheduled to run again using the **notify()** method.

Blocked state

A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain conditions. A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

Dead state

Every thread has a life cycle. A running thread ends its life when it has completed executing its **run()** method. It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it. A thread can be killed as soon as it is born, or while it is running, or even when it is in "not runnable" condition.

(b) Create a try block that is likely to generate exception at three different places. Provide the necessary catch blocks to catch and handle those exceptions.

Answer

```
class ThreeException {
      static void threeExcep(int a) {
            try {
                  /* If one command-line arg is used, then a divide-by-
            zero exception will be generated by the following code. */
                  if(a==1) a = a/(a-a); // division by zero
                  /* If two command-line args are used, then generate an
            out-of-bounds exception. */
                  if(a==2) {
                        int c[] = { 1 };
                        c[42] = 99; // generate an out-of-bounds
                  exception
                  }
            } catch(ArrayIndexOutOfBoundsException e) {
                  System.out.println("Array index out-of-bounds: " + e);
      }
}
      public static void main(String args[]) {
            try {
                  int a = args.length;
                        /* If no command-line args are present, the
                  following statement will generate a divide-by-zero
                  exception. */
                  int b = 42 / a;
```

```
                System.out.println("a = " + a);
                threeExcep(a);
            } catch(ArithmeticException e) {
                System.out.println("Divide by 0: " + e);
            }
        }
}
```

Q5 (a) Suppose that you are writing an applet, and you want the applet to respond in some way when the user clicks the mouse on the applet. What are the four things you need to remember to put into the source code of your applet?

Answer

Following are the four things that should be put into source code of an applet:

(1) Since the event and listener classes are defined in the java.awt.event package, you have to put "import java.awt.event.*;" at the beginning of the source code, before the class definition.

(2) The applet class must be declared to implement the MouseListener interface, by adding the words "implements MouseListener" to the heading. For example: "public class MyApplet extends JApplet implements MouseListener". (It is also possible for another object besides the applet to listen for the mouse events.)

(3) The class must include definitions for each of the five methods specified in the MouseListener interface. Even if a method is not going to do anything, it has to be defined, with an empty body.

(4) The applet (or other listening object) must be registered to listen for mouse events by calling addMouseListener(). This is usually done in the init() method of the applet.

(b) With the help of suitable program code explain the following methods:

(i) equals() versus ==

The **equals( )** method and the == operator perform two different operations. The **equals( )** method compares the characters inside a **String** object. The == operator compares two object references to see whether they refer to the same instance. The following program shows how two different **String** objects can contain the same characters, but references to these objects will not compare as equal:

```
// equals() vs ==

class EqualsNotEqualTo {
    public static void main(String args[]) {
        String s1 = "Hello";
        String s2 = new String(s1);
        System.out.println(s1 + " equals " + s2 + " -> "
            + s1.equals(s2));
        System.out.println(s1 + " == " + s2 + " -> " +
            (s1 == s2));
    }
}
```

The variable **s1** refers to the **String** instance created by "**Hello**". The object referred to by **s2** is created with **s1** as an initializer. Thus, the contents of the two **String** objects are identical, but they are distinct objects. This means that **s1** and **s2** do not refer to the same objects and are, therefore, not ==, as is shown here by the output of the preceding example:

OUTPUT
Hello equals Hello -> true
Hello == Hello -> false

(ii) equals and equalsIgnoreCase()

To compare two strings for equality, use **equals( )**. It has this general form:

```
boolean equals(Object str)
```

Here, *str* is the **String** object being compared with the invoking **String** object. It returns **true** if the strings contain the same characters in the same order, and **false** otherwise. The comparison is case-sensitive.

To perform a comparison that ignores case differences, call **equalsIgnoreCase( )**. When it compares two strings, it considers **A-Z** to be the same as **a-z**. It has this general form:

```
boolean equalsIgnoreCase(String str)
```

Here, *str* is the **String** object being compared with the invoking **String** object. It, too, returns **true** if the strings contain the same characters in the same order, and **false** otherwise.

Here is an example that demonstrates **equals( )** and **equalsIgnoreCase( )**:

```
// Demonstrate equals() and equalsIgnoreCase().
class equalsDemo {
	public static void main(String args[]) {
		String s1 = "Hello";
		String s2 = "Hello";
		String s3 = "Good-bye";
		String s4 = "HELLO";
		System.out.println(s1 + " equals " + s2 + " -> " +
			s1.equals(s2));
		System.out.println(s1 + " equals " + s3 + " -> " +
			s1.equals(s3));
		System.out.println(s1 + " equals " + s4 + " -> " +
			s1.equals(s4));
		System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> "
		+ s1.equalsIgnoreCase(s4));
	}
}
```

The output from the program is shown here:

```
Hello equals Hello -> true
Hello equals Good-bye -> false
Hello equals HELLO -> false
Hello equalsIgnoreCase HELLO -> true
```

(c) Write a program that generates random integers and stores them in a file named "rand.dat". The program then reads the integers from file and display on the screen.

Answer

```
import java.io.*;
public class TestReadWriteIntegers {
```

```
    public static void main (String args[]) {
      DataInputStream dis = null;
      DataOutputStream dos = null;

      File intFile = new File("rand.dat");

      try {
            dos = new DataOutputStream(new
                        FileOutputStream(intFile));
            for (int i=0; i<20; i++)
                  dos.writeInt((int)(Math.random()*100));
      }
      catch (IOException ioe){
            System.out.println(ioe.getMessage());
      }
      finally{
            try {
                  dos.close();
            } catch (IOException ioe){
            }

      }

      try {
            dis = new DataInputStream(new FileInputStream
                  intFile));
            for (int i=0; i<20; i++){
                  int n = dis.readInt();
                  System.out.print(n + " ");
            }
      } catch (IOException ioe) {
                  System.out.println(ioe.getMessage());
            } finally {
                  try {
                        dis.close();
                  } catch (IOException ioe) {
            }
            }

    }

}
```

Q6 (a) Give reasons why you might want to convert a collection into an array? Write a program to illustrate how to convert an ArrayList into an array and find the sum of elements of an ArrayList.

Answer

The reasons are as follows:

- To obtain faster processing times for certain operations
- To pass an array to a method that is not overloaded to accept a collection
- To integrate collection-based code with legacy code that does not understand collections

/* The following program Convert an ArrayList into an array and find the sum of elements of an ArrayList. */

```
import java.util.*;

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

            // Create an array list.
            ArrayList<Integer> al = new ArrayList<Integer>();

            // Add elements to the array list.
            al.add(1);
            al.add(2);
            al.add(3);
            al.add(4);
            System.out.println("Contents of al: " + al);

            // Get the array.
            Integer ia[] = new Integer[al.size()];
            ia = al.toArray(ia);
            int sum = 0;

            // Sum the array.
            for(int i : ia) sum += i;
            System.out.println("Sum is: " + sum);
      }
}
```

(c) Describe the two features that define the essence of Swing?
Answer
Swing Components Are Lightweight
With very few exceptions, Swing components are *lightweight*. This means that they are written entirely in Java and do not map directly to platform-specific peers. Because lightweight components are rendered using graphics primitives, they can be transparent, which enables nonrectangular shapes. Thus, lightweight components are more efficient and more flexible. Furthermore, because lightweight components do not translate into native peers, the look and feel of each component is determined by Swing, not by the underlying operating system. This means that each component will work in a consistent manner across all platforms.
Swing Supports a Pluggable Look and Feel
Swing supports a *pluggable look and feel* (PLAF). Because each Swing component is rendered by Java code rather than by native peers, the look and feel of a component is under the control of Swing. This fact means that it is possible to separate the look and feel of a component from the logic of the component, and this is what Swing does. Separating out the look and feel provides a significant advantage: it becomes

possible to change the way that a component is rendered without affecting any of its other aspects. In other words, it is possible to “plug in” a new look and feel for any given component without creating any side effects in the code that uses that component. Moreover, it becomes possible to define entire sets of look-and-feels that represent different GUI styles. To use a specific style, its look and feel is simply “plugged in.” Once this is done, all components are automatically rendered using that style.

Pluggable look-and-feels offer several important advantages. It is possible to define a look and feel that is consistent across all platforms. Conversely, it is possible to create a look and feel that acts like a specific platform. For example, if you know that an application will be running only in a Windows environment, it is possible to specify the Windows look and feel. It is also possible to design a custom look and feel. Finally, the look and feel can be changed dynamically at run time.

Java SE 6 provides look-and-feels, such as metal and Motif, that are available to all Swing users. The metal look and feel is also called the *Java look and feel*. It is platform-independent and available in all Java execution environments. It is also the default look and feel. Windows environments also have access to the Windows and Windows Classic look and feel.

Q7 (a) What are the major tasks in the overall website development process? What knowledge and skills do you expect to learn to be able to participate in this process?

Answer

To create a website, there are many tasks involved. The overall website development process can be summarized here. Subsequent chapters provide in-depth coverage of these tasks.

Requirement Analysis and Development Plan: What are the requirements for the finished site? What exactly will your finished website achieve for the client? What problems does your client want you to solve? Who are the target audiences of the website? Can you realistically help the client? What is the scope and nature of work? What are the design and programming tasks involved? What resources and information will be needed and what problems you foresee? Who will provide content information for the site, in what formats? What resources are needed or available: textual content, photos, imagery, audio, video, logos, corporate identity standards, copyrights, credits, footers, and insignia?

Answer the preceding questions and make a plan, create and group content, functional, and look and feel requirements, and set clear goals and milestones for building and developing the site.

Site Architecture: Decide on an appropriate architecture for the site. The site architecture is influenced by the nature of the information being served and the means of delivery. Ordinary sites involve static pages with text and images and online forms. Specialized sites may involve audio, video, steaming media, and dynamically generated information, or access to databases.

Website information architecture (IA) deals with the structuring, the relationship, the connectivity, the logical organization, and the dynamic interactions among the constituent parts of a website.

Within each Web page, consider the placement, layout, visual effect, font and text style, etc. These are also important but may be more \interior decoration" rather than architecture. However, architecture and interior/exterior decoration are intimately related.

The site architecture phase produces a blueprint for building the website. The blueprint is a specification of the components and their contents, functionalities, relations, connectivity, and interactions. Website implementation will follow the architecture closely.

An important aspect of site architecture is the navigation system for visitors to travel in your site. The goal is to establish site-wide (primary), intra-section (secondary), and intra-page (tertiary) navigation schemes that are easy and clear.

Text-only Site Framework: Follow these steps to prepare a skeletal site as a foundation for making adjustments and for further work to complete the site.

- Content: create content list or inventory; prepare content files ready to be included in Web pages.
- Site Map: draw a relationship diagram of all pages to be created for the site, give each page

appropriate titles, show page grouping, on-site and off-site links, distinguish static or dynamic pages, identify forms and server-side support. Major subsections of the site can have their own submaps.

- Skeletal site: entry page, homepage, typical subpages and sub-subpages, textual contents (can be in summary form), HTML forms with textual layout and descriptions of server-side support, structure of the file hierarchy for the site, well-defined HTML coding standards for pages.
- Navigation: follow the site architecture and site map to link the pages, use textual navigation links with rough placements (top, left, right, or bottom), avoid dead end pages and avoid confusing the end-user.

Visual Communication and Artistic Design:

- Design concepts: features, characteristics and look and feel of the site. The design must reflect client identity and site purpose.
- Story boards: simple layout sketches based on text-only site for typical pages, HTML forms, and HTML form response pages; header, footer, margins, navigation bar, logo, and other graphical elements to support the delivery of content; client feedback and approval of story boards.
- Page layout: (for pages at all levels) content hierarchy and grouping; grids, alignments, constants and variables on the page; placement and size of charts, graphs, illustrations, and photos; creative use of space and variations of font, grid, and color; style options and variations.
- Homepage/entry page: Visuals to support the unique function and purpose of entry to the site and homepage as required by site architecture.

Site Production:

- Page Templates: Create templates for typical pages at all levels. Templates are skeleton files used to make finished pages by inserting text, graphics, and other content at marked places in the templates. In other words, a template is a page frame with the desired design, layout and graphics, ready to receive text, links, photos, and other content. A template page may provide HTML, style sheets, Javascript, head, body, meta, link, and script tags as well as marked places for page content. Templates enable everyone on the project team to complete pages for the site. Advanced templates may involve dynamic server-side features.
- Prototype pages: Use the templates to complete typical pages in prototype form, test and examine page prototypes, present prototype pages to the client and obtain feedback and approval. Make sure that the layout system has been designed versatile and flexible enough to accommodate potential changes in content.
- Client-side programming: Write scripts for browsers and possibly other Web clients that will be delivered together with Web pages to the client side. Such scripts may include style sheets and Javascripts. Client side programs can make Web pages more interactive and responsive.
- Server-side programming: Write programs for form processing, dynamic page generation, database access, e-business, and e-commerce features. Make sure these follow the site architecture, user-orientation, and visual design.
- Finished pages: Following page prototypes, add text, graphics, photos, animations, audio and video to templates to produce all pages needed; make final adjustments and fine tuning.

Error checking and validation: Apply page checking tools or services on finished pages to remove spelling errors, broken links, and HTML coding problems. Check page loading times.

Testing: Put the site through its paces, try different browsers from different access locations, debug, fine tune, and check against architecture and requirements.

Deploying: Release the site on the Web, make its URL known, and register the site with search engines.

Documentation: Write down a description of the website, its design and functionalities, its file structure, locations for source files of art and programming, a site maintenance guide.

Maintenance: Continued operation and evolution of the website.

(b) Define HTML? What are the different categories under which HTML elements fall?
Answer
HTML is a markup language that provides tags for you to organize information for the Web. By inserting HTML tags into a page of text and other content, you mark which part of the page is what to provide structure to the document. Following the structure, user agents such as browsers can perform on-screen rendering or other processing. Thus, browsers process and present HTML documents based on the marked-up structure. The exact rendering is defined by the browser and may differ for different browsers. For example, common visual browsers such as Internet Explorer (IE) and Netscape Navigator (NN) render Web pages on screen. A browser for the blind, on the other hand, will voice the content according to its markup.
Hence, a Web page in HTML contains two parts: markup tags and content. HTML tags are always enclosed in angle brackets (< >). This way, they are easily distinguished from contents of the page.

HTML provides over 90 different elements. Generally, they fall into these categories:
Top-level elements: html, head, and body.
Head elements: elements placed inside head, including title (page title), style (rendering style), link (related documents), meta (data about the document), base (URL of document), and script (client-side scripting).
Block-level elements: elements behaving like paragraphs, including h1-h6 (headings), p (paragraph), pre (pre-formatted text), div (designated block), ul, ol, dl (lists), table (tabulation), and form (user input forms). When displayed, a block-level (or simply block) element always starts a new line and any element immediately after the block element will also begin on a new line.
Inline elements: elements behaving like words, characters, or phrases within a block, including a (anchor or hyperlink), br (line break), img (picture or graphics), em (emphasis), strong (strong emphasis), sub (subscript), sup (superscript), code (computer code), var (variable name), kbd (text for user input), samp (sample output), span (designated inline scope).
When an element is placed inside another, the containing element is the parent and the contained element is the child.
Comments in an HTML page are given as <!-- a sample comment -->. Text and HTML elements inside a comment tag are ignored by browsers. Be sure not to put two consecutive dashes (--) inside a comment. It is good practice to include comments in HTML pages as notes, reminders, or documentation to make maintenance easier.
In an HTML document certain characters, such as < and &, are used for markup and must be escaped to appear literally. Other characters you may need are not available on the keyboard. HTML provides entities (escape sequences) to introduce such characters into a Web page. For example, the entity < gives < and ÷ gives ÷.

Q8 (a) With the help of HTML code, discuss Page Forwarding?
Answer
Web pages sometimes must move to different locations. But visitors may have bookmarked the old location or search engines may still have the old location in their search database. When moving Web pages, it is prudent to leave a forwarding page at the original URL, at least temporarily.
A forwarding page may display information and redirect the visitor to the new location automatically. Use the meta tag

```
<meta http-equiv="Refresh" content="8; url=newUrl " />
```

The http-equiv meta tag actually provides an HTTP response header for the page. The above meta element actually give the response header

```
      Refresh 8; url=newUrl
```

The effect is to display the page and load the newUrl after 8 seconds. Here is a sample forwarding page

```
<head><title>Page Moved</title>
      <meta http-equiv="Refresh" content="8; url=target_url" />
</head><body>
<h3>This Page Has Moved</h3>
      <p>New Web location is: ... </p>
      <p>You will be forwarded to the new location automatically.</p>
</body></html>
```

The Refresh response header (meta tag) can be used to refresh a page periodically to send updated information to the end-user. This can be useful for displaying changing information such as sports scores and stock quotes. Simply set the refresh target url to the page itself so the browser will automatically retrieve the page again after the preset time period. This way, the updated page, containing the same meta refresh tag, will be shown to the user.

(b)Explain the six concrete steps of Information Architecture?
Answer
The six concrete steps to Information Architecture are:
(1) define goals,
(2) define audience,
(3) create and organize content,
(4) formulate visual presentation concepts
(5) develop site map and navigation, and
(6) design and produce visual forms.

Define Goals: The first step involves surveying key people to get a clear idea of what should appear on the site. Create questions to determine site's mission and purpose by involving everyone in the creative process. Next, you should define the scale of your project and time frame for completion. How to obtain these goals? Call meetings with key players; Prepare agenda and questions; Talk to client employee's one-on-one, record their responses; Get their thoughts, ideas, opinions; Get approval from key people.
Here are sample questions which may help you reveal the true objectives of the site:
1. What is the mission or purpose of the organization? Read the mission statements and business plans. Review client's literature. Remember, client's mission may change with time.
2. What are the short and long-term goals of the site? Key people may not be thinking in the long term. Immediate need may be to get the site up and running. Look toward the future; accommodate growth and change.
3. Who are the intended audiences? Inadequate analysis in this area is the number one mistake made in designing sites!
4. Why will people come to your site for the first time; for repeat visits?
5. Does the site provide a well-defined service? or sell specific products?
6. Does your client have an existing site? Find this out now.

Be sure to write down all answers and prioritize objectives in order of importance. Group goals into

categories and have people rank the importance of each category separately. By doing this step, you will establish a clear set of goals which will be used to design the site. Be sure to share these with your client and the employees involved in key decision in the company.

Define Audience : The purpose of this step is to determine who your users are and what are their goals and objectives? This also means that you need to define user experience; understand how users will react with the site. It is helpful to write scenarios for all intended audiences. Scenarios are stories which describe steps which someone may take in using the site. Scenarios define the users experience and enable a designer to better visualize and connect with their audience.
A good scenario depicts the activities, moves, and experiences of a possible visitor to your site. Writing scenarios may seem like a frivolous task but be assured, it is no such thing. In fact, it may be the single biggest factor in defining the user experience.
You may be uncertain how to begin. Here are some simple guidelines to follow. Refer to the audience list gathered in Information Architecture, Step 1. Create a set of users who represent the majority of visitors. Depending on the size of the site as well as the audience, determine how many user scenarios are needed. Write a scenario for each user: name, background, and task to accomplish; use a task from your list of audience needs and goals. Discuss the scenarios with your team members and with potential visitors. Once you have good scenarios, use them to define content and functional requirements. Next, prioritize each audience group, compile results and share them with key people. Get approvals from client.
The last step in defining your audience is knowing your competition. Be aware of what your competition is doing to list your competitors. One criteria for judging competition may include download time, page size, design and \feel" of site. To conclude this step, you may write a summary of the target audience and include it in your report for Steps 1 and 2.

Create and Organize Content: Most of the time, programmers and designers are not responsible for creating content for clients. Content, in this case, refers to written text and images which appear on the site. Text is usually written by marketing people, copy writers or public relations staff on your team or the client's team. Images may be supplied by the client or generated by designers.
It is your job as a designer to organize content into major sections. In the initial stages of development, you need to answer two questions regarding content: “What content does the site need?” and “What functionality will be required?” Then, you need to create a content and functional requirements list where you label and group content.
Here is a list of sample questions which may help you determine the functional requirements for the site:

1. Which pages will be static and which dynamic?
2. What will be the function of these pages?
3. What transactions will users perform?
4. Copyright notices and privacy statements?
5. What about membership rules, member login pages, sign-up pages for e-mail newsletters, and other pages involving forms or transactions?

Complete and prioritize content and functional requirements list: Rank the importance of each item and ask yourself: Do you have the technology and the skills to meet each requirement? Do you have the time and money to buy or build the functionality? You may have to drop some items in order to meet your deadlines.
Organize site content by grouping and labeling content items you have gathered. The content organization will help define the site architecture. Try organizing content in different ways. After grouping items, name each group and be descriptive. Discuss grouping and labeling within the team and with other key people to see if they match expectations and functional requirements. Final groupings and labels will be used to define sections of the site. Consider major sections as transient {their names and content may

change in next stage of IA process. Get client approval. Revise if necessary.
By completing this step, you will inventory, group and label the content into majors sections and determine what sort of functionality will be required. This content and functionality will be the basis of your site.

Formulate Visual Presentation Concepts: Concept can be defined as a visual direction, an idea or theme for the site. Concept is the idea you want to communicate and present. Design is how you express that idea.
It is important to keep concepts simple. Our rule of thumb regarding concepts is this: Articulate your concept in one or two sentences. If you have to write two paragraphs to explain your concept, you're probably trying to say too much. Remember you can't be everything at once, even though the client may insist that you can. If you try to be everything to everyone, your design will be a mess, lacking focus and saying nothing. By trying to be everything, you will confuse your visitors. You need to prioritize your objectives and then proceed accordingly. As with other aspects of design, knowing what to leave out is just as important as knowing what to include. The old adage "less is more" is especially true in Website design.
So how do we get started on creating concepts? Here is one method for creating concepts. The following process is a simple method for creating concepts that will lead to ideas for your site. These ideas can then be translated into visuals.

- **Word Associations:** Make a list of words associated with a particular subject or idea. Write them all down. Don't edit your ideas at this point, rather, write all words or phrases associated with the subject and try to write as many words as possible.
- **Word Links:** Connect the words or phrases from one list with another. Be sure to do this randomly! Choose combinations which have the most promising possibilities. Don't necessarily rule out odd or implausible solutions-they may lead to interesting concepts later.
- **Written Rough**: In this stage, you will simply try to write down a few random thoughts which may later become visuals. At this point, you are still only writing and not sketching your ideas. Be sure to explore a wide range of possibilities and not just ones you think you can find or draw yourself. You should approach this problem as if you have a whole staff of illustrators, photographers and production artists at your disposal. This way, you won't limit your thinking!

Develop Site Map and Navigation: The site map is a comprehensive, diagramed layout of the site, which describes its organization structure. Site maps are synonymous with site structure and at their best, map out major sections of a site and construct pattern in layers and levels. At their worst, site maps can be confusing, and fail to indicate key sections and subsections. Sometimes they may be incomplete, and not include vital cross links or depth levels within the site. Sometimes site maps are simply chaotic and difficult to read. These errors can lead to lots of frustration for clients as well as internal miscommunication about the site's requirements. It is vital that site maps provide a clear road map to all section of the site; that clients understand and approve them; and that everyone on the design and programming team understand exactly what is going to appear on the site and where it will appear.
Site maps provide a clear road map to all sections of the site, that clients understand and approve them and everyone on the design and programming team understands exactly what is going to appear and where on the site.
A good site map should provide viewers with all key site sections as well as descending levels in the site.
Site maps should:

- Provide cross links indicating all ways for getting from point A to point B
- Provide major sections or "roots" of structure listings
- Map out organization of each section with items from content inventory

- Have a legend that defines how on-site and off-site links work
- Distinguish function, transaction and dynamically generated pages from simple text pages
- Make provisions for large sites by making several maps starting with a generalized overview followed by all subsequent subsections

Navigation is a method of informing the viewer on three pieces of information: Where am I? Where have I been? and Where can I go from here? If these three things are clearly indicated, you are half way there.

- Here are some other things to consider when designing an interface which employs effective navigation.
- How will users use the site?
- How will they get from one place to another?
- How do you prevent them from getting lost?
- Make sure that major sections are included in the global navigation system.
- Local navigation should include a list of topics, menu, and a list of a few related items.
- Use title of section as a link to that section.
- Identify each page of your site (logo, name of company, symbol, etc.)
- Accurately describe each page which will appear in the browser bar for effective book marking.
- Be sure to orient the viewer to your site as well as to the Web as a whole.
- Be aware that users prefer to control where they want to go. Confining viewers to certain sections does not promote friendly usability.
- Design for flexibility of movement.

In completing this step, write down the logical organization of the site, the structure of links in the site, the way the site can be navigated. Decide the organization of file folders, sub-folders, and files.
So far, you have successfully combined reasons for the site and target audience; you have categorized content; and in this fifth step, you have created a site structure that will provide a valuable foundation for form, which will be created in the next step.

Design and Produce Visual Forms: Visual form is the way your site looks. Visual form refers to all things visual on the site such as: layout, type, graphics, colors, logos, charts, photos and illustrations. Visual form is created when all previous steps have been completed. It is the part of your site design which requires your creative ability, as well as formal training. It is the part of your site which identifies the client, creates a brand, creates the look and feel of the site, and sets the mood for all you say about your client's goods and services.
Visual form often begins with mapping contents in a given space. To map contents onto Web pages, you need to consider a number of things.
Uniformly organize pages in different sections. Be sure to examine the best ways to display text; itemize bullet points for easy scanning. Although there are no steadfast rules for this, usability testing shows that readers prefer 20% of the page designated for navigation and 80% for content. Block out space for global and local navigation; integrate other aspects of the site which may not be part of the site structure. A company's brand identification should be present on each page. Examine your structure listing and make a list of all possible page types. History page, mission statement, ordering information and employee section are all examples of different kind of content which may need to be laid out differently, yet all must appear related to one another. Individual pages within sections should be formally consistent in order to preserve visual unity. Review content inventory and develop two or three generic page types. Consider branding, advertising and sponsorship, navigation, page titles, header graphics, and footers, including copyrights. Incorporate advertising and sponsorship — Do you put it at the top? Do you put it under the title of each page? How do you integrate sponsorship? Is sponsorship integrated into the

graphic headers on each page? Is there a small sponsorship logo at the bottom of each page? Global navigation must be consistent across every page of your site while local navigation systems can change, depending on the content. Be as consistent as possible. Create page mock-ups representing the actual site and integrate design sketches with layout grids. Use design sketches as temporary graphics.

Q9 (a) Create a HTML form that has the following controls:

- A TEXT control called `firstName` to collect the first name.
- A TEXT control called `lastName` to collect the last name.
- A TEXT control called `email` to collect the email address.
- A TEXT control called `phone` to collect the phone number.
- A SELECT control called `software` for displaying a combo box with software list.
- A SELECT control called `os` for displaying a combo box with operating systems.
- A TEXTAREA control called txtArea for displaying problem description.
- A SUBMIT control called submit for submitting the information.

Answer

```
<HTML>
<HEAD>
  <TITLE>My Web Page</TITLE>
</HEAD>

<BODY>
  <H1> TECHNICAL SUPPORT SYSTEM</H1>
  <BR>
  <CENTER>
    <FORM ACTION =”tss” Method=”post”>
    <TABLE ALING = “center” WIDTH =”100%” CELLSPACING=”2” CELLPADDING=”2”>
      <TR>
       <TD ALING=”right”>First Name:</TD>
       <TD><INPUT TYPE=”text” NAME=”firstName” ALING=”left” SIZE=”15”></td>
       <TD ALING=”right”>Last Name:</TD>
       <TD><INPUT TYPE=”text” NAME=”lastName” ALING=”left” SIZE=”15”></td>
      </TR>
      <TR>
       <TD ALING=”right”>Email:</TD>
       <TD><INPUT TYPE=”text” NAME=”email” ALING=”left” SIZE=”25”></td>
       <TD ALING=”right”>Phone:</TD>
       <TD><INPUT TYPE=”text” NAME=”phone” ALING=”left” SIZE=”15”></td>
      </TR>
      <TR>
       <TD ALING=”right”>Software:</TD>
       <TD>
         <SELECT NAME=”software” SIZE=”1”></td>
          <OPTION VALUE=”Word”>Microsoft Word</OPTION>
          <OPTION VALUE=”Excel”>Microsoft Excel</OPTION>
          <OPTION VALUE=”Access”>Microsoft Access</OPTION>
         </SELECT>
```

```
        </TD>
        <TD ALING="right">Operating system:</TD>
        <TD>
          <SELECT NAME="os" SIZE="1"></td>
           <OPTION VALUE="95">Windows 95</OPTION>
           <OPTION VALUE="98">Windows 98</OPTION>
           <OPTION VALUE="nt">Windows NT</OPTION>
          </SELECT>
        </TD>
       </TR>
     </TABLE>
  <BR> Problem Description
  <BR>
  <TEXTAREA NAME="problem" COLS="50" ROWS="4"></TEXTAREA>

  <BR><BR>

  <INPUT TYPE="Submit" NAME="submit" VALUE="SUBMIT REQUEST">
 </FORM>
</CENTER>
</BODY>
</HTML>
```

(b) Write a function in java script that validates if the content has the general syntax of an email.

Answer

```
function validateForm()
{
var x=document.forms["myForm"]["email"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
  {
  alert("Not a valid e-mail address");
  return false;
  }
}
```

TEXT BOOKS

I. **The Complete Reference Java, Herbert Schildt, TMH, Seventh Edition, 2007**

II. **An Introduction to Web Design + Programming, Paul S. Wang and Sanda S. Katila, Thomson Course Technology, India Edition, 2008**