Q.2 a. List out the properties of static variables and methods in Java. (4)

Answer:

Unlike instance variables, static variables belong to a class, rather than to specific instances of it. This means that there is only one copy of a static variable, which is accessible from all members of the class, as well as from external "customer" code via objects of the class. For example, a static variable could keep track of a property within the application which remains the same for all class members. The following sample code demonstrates declaring a static variable inside a class declaration:

```
private static int numWomen = 0;
```

Within the class constructor or another method, the variable can be accessed and updated as follows:

```
numWomen++;
```

Class declarations can include static methods. As with variables, static methods provide some functionality that is the same across all object instances of a class. Static methods commonly carry out processing that involves static variables. The following sample static method returns the value of a static variable within a class declaration:

```
public static int getNumWomen() {

return numWomen;

}
```

b. Explain various primitive data types supported by Java. (4)

Answer:

int

Int is the data type used to declare integers. The int data type is 32 bits and has a maximum value of 2,147,483,647; anything beyond that requires a long data type, which is also primitive. Int is used frequently in Java for calculations. It can also be used to count cycles of a loop, a common programming construct. To declare an int, use int variableName, substituting variableName with the name of your choice.

Double

Double is used to store decimals. It is used frequently in programs that deal with money and finances. You can also use float, another primitive type, for decimal numbers; the choice depends on the situation or personal preference. To declare a double, use double variableName, substituting variableName with the name of your choice.

Boolean

The boolean data type holds two values: true and false. It is a tiny 1 bit in size, reflecting the implied value of 0 for false and 1 for true. Boolean variables are often used in tests; for example, setting a value to true would cause a loop to execute.

Char

The char type is a single character. Sixteen bits in size, char is useful for manipulating string variables. For example, you could break a word into the letters it comprises, storing each in a char variable. Char is also useful as a test in switch statements and anything else requiring a single letter.

c. "One object in Java can be assigned as reference to another object of the same type." To explain this concept write a complete Java program and explain how a reference object works. (8)

Answer:

Q.3 a. Explain the concept of Classes and Objects. Explain with examples as how are they created in Java? (5)

Answer:

Objects in Java:

If we consider the real-world we can find many objects around us, Cars, Dogs, Humans, etc. All these objects have a state and behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging, running

If you compare the software object with a real world object, they have very similar characteristics.

Software objects also have a state and behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java:

A class is a blue print from which individual objects are created.

A sample of a class is given below:

public class Dog{

```
  String breed;
  int age;
  String color;
  void barking(){
  }
  void hungry(){
  }
  void sleeping(){
  }
}
```

A class can contain any of the following variable types.

Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

Instance variables: Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

Class variables: Class variables are variables declared with in a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

b. What is a constructor? Write a Java program to explain how super class constructors are called in their subclasses. (6)

Answer:

Constructors are like special methods that are called implicitly as soon as an object is instantiated (i.e. on new ClassName()).

- Constructors have no return type (not even void).
- The constructor name must match the class name.
- If you don't define an explicit constructor, Java assumes a default constructor
- The default constructor accepts no arguments.

- The default constructor automatically invokes its base class constructor with no arguments, as discussed later in this module.

You can provide one or more explicit constructors to:

- Simplify object initialization (one line of code to create and initialize the object)
- Enforce the state of objects (require parameters in the constructor)
- Invoke the base class constructor with arguments, as discussed later in this module.
- Adding any explicit constructor disables the implicit (no argument) constructor.
- We can add a constructor to our Employee class to allow/require that it be constructed with a name and a social security number:

```
class Employee {
   String name;
   String ssn;
   …
   Employee(String name, String ssn) {
      this.name = name; // "this." helps distinguish between
      this.ssn = ssn; // instance and parameter variables
   }
   …
}
```

Then we can modify EmployeeDemo.main() to call the specified constructor:

```
public class EmployeeDemo {
   public static void main(String[] args) {
      Employee e1 = new Employee("John", "555-12-345");
      e1.emailAddress = "john@company.com";
      Employee e2 = new Employee("Tom", "456-78-901");
      e2.setYearOfBirth(1974);
      …
   }
}
```

c. What is a package? Explain, with an example, how name conflicts are resolved during package import. (5)

Answer:

Packages in Java

- A Java package name consists of a set of name components separated by periods (`.`).
  - Each name component must be a valid Java identifier.
  - A component name must not start with an upper-case letter.

The package name corresponds to a directory structure on disk where the classes are stored.

- For example, the class and interface files for `org.xml.sax` are located in the directory`org/xml/sax`, relative to the directory where Java looks for classes (we'll discuss this later in this module).

- To add a class or interface to a package:
  - Add `package myPackageName;` as the first Java statement of the source file
  - In your development directory, store the source file in a directory structure corresponding to your package name.

Java does not require you to store your `.java` source files in in package-based directories, but it is a common convention. However, once the source files are compiled, the generated `.class` files are (and must be) stored in the package-based directories.
It's also common to have separate, parallel directory hierarchies for your `.java` and `.class` files reflecting the package structure. Having your source files separate from your class files is convenient for maintaining the source code in a source code control system without having to explicitly filter out the generated class files. Having the class files in their own independent directory hierarchy is also useful for generating JAR files, as discussed later in this module.

Most modern IDEs can create and manage your source and class file directory hierarchies automatically for you.

Importing Package Members

- Instead of using fully-qualified names for classes and interfaces, you can *import* the package member.

- To import a specific member from a package, include an `import` statement specifying the fully-qualified name of that member. For example:

```
import com.marakana.utils.MyClass;
```

You can then use the simple (unqualified) name of the member throughout the rest of the source file.

- You can import **all** of the members of a package by importing the package with an asterisk (`*`) wildcard. For example:

```
import com.marakana.utils.*;
```

You can then use the simple (unqualified) name of all the package members throughout the rest of the source file.

Using the wildcard `import` does **not** import any sub-packages or their members.

- The `import` statements must appear after any `package` statement and before all class/interface definitions in your file.

The `java.lang` package is imported automatically into all source files.

In case there are two or more classes with the same name but defined in different imported (or assumed) packages, then those classes must be referenced by their fully-qualified-class-name (FQCN).

For example, you could define:

```
com.mycompany.calc.AdditionOperation,
```

but someone else could implement:

```
com.othercompany.calculator.AdditionOperation.
```

If you wanted to use both addition operations from a class defined in `com.mycompany.calc` package, then doing

```
import com.othercompany.calculator.AdditionOperation;
```

would make AdditionOperation ambiguous. Instead of importing it, you would reference it by its FQCN.

Even though importing an entire package using the asterisk (*) notation is convenient, it is not generally recommended — especially when multiple packages are imported. It makes it harder to track down the classes if you do not know exactly which package they come from. Modern IDEs help with resolving classes (as well as with organizing imports), but you should not assume that everyone will always have an IDE handy.

Q.4 a. What is an exception? Write an exception subclass which throws an exception if the variable age passed as argument to a method and the value of age is less than 20. (8)

Answer:

Exception Handling Examples

- An exception is an event, which occurs during the execution of a program, that interrupts the normal flow of the program. It is an error thrown by a class or method reporting an error in code.
- The ***'Throwable'*** class is the superclass of all errors and exceptions in the Java language
- Exceptions are broadly classified as ***'checked exceptions'*** and ***'unchecked exceptions'***. All RuntimeExceptions and Errors are unchecked exceptions. Rest of the exceptions are called checked exceptions. Checked exceptions should be handled in the code to avoid compile time errors.
- Exceptions can be handled by using ***'try-catch'*** block. Try block contains the code which is under observation for exceptions. The catch block contains the remedy for the exception. If any exception occurs in the try block then the control jumps to catch block.
- If a method doesn't handle the exception, then it is mandatory to specify the exception type in the method signature using ***'throws'*** clause.
- We can explicitly throw an exception using ***'throw'*** clause.

Example: Use this code for check:

```
try
{
   System.out.println("How old are you?");
   int age = in.nextInt();
   System.out.println("Next year, you'll be " + (age + 1));
}
catch (InputMismatchException exception)
{
   exception.printStackTrace();
}
```

b. What is multithreaded programming? Explain how threads are created in Java. Explain the need of thread synchronization, with an example. (8)

Answer:

Here is an example that creates a new thread and starts it running:

```
// Create a new thread.
class NewThread implements Runnable {
  Thread t;
  NewThread() {
    // Create a new, second thread
    t = new Thread(this, "Demo Thread");
    System.out.println("Child thread: " + t);
    t.start(); // Start the thread
  }
  // This is the entry point for the second thread.
  public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
        // Let the thread sleep for a while.
        Thread.sleep(50);
      }
    } catch (InterruptedException e) {
      System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
  }
}
public class ThreadDemo {
  public static void main(String args[]) {
```

```
    new NewThread(); // create a new thread
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Main Thread: " + i);
        Thread.sleep(100);
      }
    } catch (InterruptedException e) {
      System.out.println("Main thread interrupted.");
    }
    System.out.println("Main thread exiting.");
  }
}
```

This would produce the following result:

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
```

Main thread exiting.

Need For thread Synchronisation:

Thread SynchrNeed ForWhen two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time.

The process by which this synchronization is achieved is called *thread synchronization*.

The synchronized keyword in Java creates a block of code referred to as a critical section. Every Java object with a critical section of code gets a lock associated with the object. To enter a critical section, a thread needs to obtain the corresponding object's lock.

This is the general form of the synchronized statement:

```
synchronized(object) {
  // statements to be synchronized
}
```

Here, object is a reference to the object being synchronized. A synchronized block ensures that a call to a method that is a member of object occurs only after the current thread has successfully entered object's monitor.

Q.5 a. Write a program in Java to read a text and count all the occurrences of a given word. Also, display their positions. (8)

Answer:

```
import java.io.*;

class TextSearch

{

public static void main(String as[]) throws Exception

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

int i=0,count=0;

String text="",s="";
```

```
System.out.println("Enter Text:(press ENTER twice to stop)\n");
s=br.readLine();
while(s.length()!=0)
{
text+=s;
s=br.readLine();
}
System.out.println("Enter search word:");
s=br.readLine();
while(true)
{
i=text.indexOf(s,i);
if(i==-1) break;
System.out.println("Word found at position:"+i);
count++;
i+=s.length();
}
System.out.println("Number of occurrences of given word:"+count);
}
}
```

/* Output: */

```
Enter Text:(press ENTER twice to stop)
```

```
Hello hi , hi again

hi nice to see you

Enter search word:hi

Word found at position:6

Word found at position:11

Word found at position:19

Number of occurrences of given word:3
```

b. What is applet? Write a simple applet program that will print "HELLO IETE". Write the HTML code required to execute this applet program and write the command to execute this program using appletviewer? (8)

Answer:
Page No. 296 – 298

Q.6 a. Compare AWT and Swings. How timers, tables and borders components are created in event handling using Java Swings? (8)

Answer:
AWT is a Java interface to native system GUI code present in your OS. It will not work the same on every system, although it tries.

Swing is a more-or-less pure-Java GUI. It uses AWT to create an operating system window and then paints pictures of buttons, labels, text, checkboxes, etc., into that window and responds to all of your mouse-clicks, key entries, etc., deciding for itself what to do instead of letting the operating system handle it. Thus Swing is 100% portable and is the same across platforms (although it is skinnable and has a "pluggable look and feel" that can make it look more or less like how the native windows and widgets would look).

AWT is a cross-platform interface, so even though it uses the underlying OS or native GUI toolkit for its functionality, it doesn't provide access to everything that those toolkits can do. Advanced or newer AWT widgets that might exist on one platform might not be supported on another. Features of widgets that aren't the same on every platform might not be supported, or worse, they might work differently on each platform. People used to invest lots of effort to get their AWT applications to work consistently across platforms - for instance, they may try to make calls into native code from Java.

Because AWT uses native GUI widgets, your OS knows about them and handles putting them in front of each other, etc., whereas Swing widgets are meaningless pixels within a window from your OS's point of view. Swing itself handles your widgets' layout and stacking. Mixing AWT and Swing is highly unsupported and can lead to ridiculous results, such as native buttons that obscure everything else in the dialog box in which they reside because everything else was created with Swing.

AWT stands for Abstract Window ToolKit. The Abstract Window Toolkit supports GUI Java programming. It is a portable GUI library for stand-alone applications and/or applets. The Abstract Window Toolkit provides the connection between your application and the native GUI. The AWT provides a high level of abstraction for your Java program since it hides you from the underlying details of the GUI your program will be running on.

AWT features include:

- A rich set of user interface components.
- A robust event-handling model.
- Graphics and imaging tools, including shape, color, and font classes.
- Layout managers, for flexible window layouts that don't depend on a particular window size or screen resolution.
- Data transfer classes, for cut-and-paste through the native platform clipboard.

The AWT components depend on native code counterparts (called peers) to handle their functionality. Thus, these components are often called "heavyweight" components.

An Overview of Swing

Swing implements a set of GUI components that build on AWT technology and provide a pluggable look and feel. Swing is implemented entirely in the Java programming language, and is based on the JDK 1.1 Lightweight UI Framework.

Swing features include:

- All the features of AWT.
- 100% Pure Java certified versions of the existing AWT component set (Button, Scrollbar, Label, etc.).
- A rich set of higher-level components (such as tree view, list box, and tabbed panes).

- Pure Java design, no reliance on peers.
- Pluggable Look and Feel.

Swing components do not depend on peers to handle their functionality. Thus, these components are often called "lightweight" components.

AWT vs. Swing

There are, of course, both pros and cons to using either set of components from the JFC in your Java applications. Here is a summary:

AWT:
Pros

- Speed: use of native peers speeds component performance.
- Applet Portability: most Web browsers support AWT classes so AWT applets can run without the Java plugin.
- Look and Feel: AWT components more closely reflect the look and feel of the OS they run on.

Cons

- Portability: use of native peers creates platform specific limitations. Some components may not function at all on some platforms.
- Third Party Development: the majority of component makers, including Borland and Sun, base new component development on Swing components. There is a much smaller set of AWT components available, thus placing the burden on the programmer to create his or her own AWT-based components.
- Features: AWT components do not support features like icons and tool-tips.

Swing:
Pros

- Portability: Pure Java design provides for fewer platform specific limitations.
- Behavior: Pure Java design allows for a greater range of behavior for Swing components since they are not limited by the native peers that AWT uses.

- Features: Swing supports a wider range of features like icons and pop-up tool-tips for components.
- Vendor Support: Swing development is more active. Sun puts much more energy into making Swing robust.
- Look and Feel: The pluggable look and feel lets you design a single set of GUI components that can automatically have the look and feel of any OS platform (Microsoft Windows, Solaris, Macintosh, etc.). It also makes it easier to make global changes to your Java programs that provide greater accessibility (like picking a hi-contrast color scheme or changing all the fonts in all dialogs, etc.).

Cons

- Applet Portability: Most Web browsers do not include the Swing classes, so the Java plugin must be used.
- Performance: Swing components are generally slower and buggier than AWT, due to both the fact that they are pure Java and to video issues on various platforms. Since Swing components handle their own painting (rather than using native API's like DirectX on Windows) you may run into graphical glitches.
- Look and Feel: Even when Swing components are set to use the look and feel of the OS they are run on, they may not look like their native counterparts.

The `javax.swing.Timer` object generates single or multiple `ActionEvent` events at time intervals that you specify. Thus, a `Timer` is useful for performing a repeated operation like an animation. They are also useful for triggering operations that must occur at some point in the future. For example, an application might display a message in a status line and then set up a `Timer` object that erases the message after 5,000 milliseconds. These operations can also be performed with threads, of course, but since Swing is not designed for thread safety, it is usually more convenient to use a `Timer`.

- You use `Timer` objects just like regular components. A `Timer` has property accessor methods and an `addActionListener()` method that you can use to add event listeners. The `initialDelay` property specifies how many milliseconds the `Timer` waits before firing its first `ActionEvent`. If the `repeats` property is `true`, the `Timer` generates a new `ActionEvent` each time `delay` milliseconds passes. When an application (or the system in general) is very busy or when the `delay` property is very small, the timer

may fire events faster than the application can process them. If the `coalesce` property is `true`, the `Timer`combines multiple pending events into a single `ActionEvent`, rather than letting a queue of unprocessed events build up.

b. Write a program to display a button with an image on it using swing. The image on the button will change when the button is clicked. (8)

Answer:

```
import java.lang.String.*;
  import java.lang.Exception.*;
  import javax.swing.*;
  import java.awt.*;
  import java.awt.event.*;
  public class ImageProcessor extends JFrame implements ActionListener{
   // instance variables
     private static final int FRAME_WIDTH = 975;
     private static final int FRAME_HEIGHT = 725;
    private static final int FRAME_X_ORIGIN = 150;
     private static final int FRAME_Y_ORIGIN = 250;
  private static final int SIDE_BUTTON_WIDTH = 150;
     private static final int SIDE_BUTTON_HEIGHT = 150;
     private static final int BOTTOM_BUTTON_WIDTH = 80;
     private static final int BOTTOM_BUTTON_HEIGHT = 30;
  private JButton stats;
     private JButton resize;
     private JButton brighten;
     private JButton darken;
     private JButton load;
     private JButton quit;
  JFileChooser chooser;
```

```
    String choosertitle;
  public static void main(String[] args){
      ImageProcessor frame = new ImageProcessor();
      frame.setVisible(true);
  }
 /**
   * Constructor for objects of class Calculate
   */
    public ImageProcessor(){
      Container contentPane;
   //Set the frame properties
      setSize       (FRAME_WIDTH, FRAME_HEIGHT);
      setResizable    (false);
      setTitle       ("Image Processor");
      setLocation     (FRAME_X_ORIGIN, FRAME_Y_ORIGIN);
      contentPane = getContentPane();
      contentPane.setLayout(null);
      contentPane.setBackground( Color.white );
     //Create and Place the Buttons on the frame
      stats = new JButton("Stats");
      stats.setBounds(825,                0,                SIDE_BUTTON_WIDTH,
SIDE_BUTTON_HEIGHT);
      contentPane.add(stats);
    resize = new JButton("Resize");
      resize.setBounds(825,               150,              SIDE_BUTTON_WIDTH,
SIDE_BUTTON_HEIGHT);
      contentPane.add(resize);
    brighten = new JButton("Brighten");
```

```
        brighten.setBounds(825,          300,          SIDE_BUTTON_WIDTH,
SIDE_BUTTON_HEIGHT);

        contentPane.add(brighten);

      darken = new JButton("Darken");

        darken.setBounds(825,          450,          SIDE_BUTTON_WIDTH,
SIDE_BUTTON_HEIGHT);

        contentPane.add(darken);

      load = new JButton("Load");

        load.setBounds(392, 650, BOTTOM_BUTTON_WIDTH,
BOTTOM_BUTTON_HEIGHT);

        contentPane.add(load);

      quit = new JButton("Quit");

        quit.setBounds(503,          650,          BOTTOM_BUTTON_WIDTH,
BOTTOM_BUTTON_HEIGHT);

        contentPane.add(quit);

      //Register this frame as an Action listener of the buttons

        stats.addActionListener(this);

        resize.addActionListener(this);

        brighten.addActionListener(this);

        darken.addActionListener(this);

        load.addActionListener(this);

        quit.addActionListener(this);

        add(load);

            //Exit program when the viewer is closed

        setDefaultCloseOperation(EXIT_ON_CLOSE);

         //Event handler

      public void actionPerformed(ActionEvent event){

        if(event.getSource() instanceof JButton){

        JButton clickedButton = (JButton) event.getSource();
```

```
        String buttonText = clickedButton.getText();
          if(buttonText.equals("Stats")){
            }
            if(buttonText.equals("Resize")){
            }
            if(buttonText.equals("Brighten"))
            }
            if(buttonText.equals("Darken")){
           }
          if(buttonText.equals("Load"))
              chooser = new JFileChooser();
              chooser.setCurrentDirectory(new java.io.File("."));
              chooser.setDialogTitle(choosertitle);
              chooser.setAcceptAllFileFilterUsed(true);
              if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
{
                System.out.println("getCurrentDirectory(): "
                  +  chooser.getCurrentDirectory());
                System.out.println("getSelectedFile() : "
                  +  chooser.getSelectedFile());
              }
              else {
                System.out.println("No Selection ");
              }
 }
 if(buttonText.equals("Quit")){
              System.exit(0);
```

}}}

PART B
Answer any TWO Questions. Each question carries 16 marks.

Q.7 a. Explain various list styles and presentation styles in XHTML. Give examples. (8)

Answer:

Inline Styles: The style Attribute

i.The inline style is the simplest way to attach a style to a tag -- just include a style attribute with the tag along with a list of properties and their values. The browser uses those style properties and values to render the contents of just this instance of the tag

For instance, the following style tells the browser to display the level-1 header text, "I'm so bluuuuoooo!", not only in the <h1> tag style characteristic of the browser, but also in the color blue and italicized (if the browser is capable)

<h1 style="color: blue; font-style: italic">I'm so bluuuuoooo!</h1>

This type of style definition is called "inline" because it occurs with the tag as it appears in the document. The scope of the style covers the contents of that tag only. Since inline styles are sprinkled throughout your document, they can be difficult to maintain. Use the style attribute sparingly and only in those rare circumstances when you cannot achieve the same effects otherwise.

ii. Document-Level Style Sheets

The real power of style sheets becomes more evident when you place a list of presentation rules within the head of a document. Enclosed within their own <style> and </style> end tags, so-called "document-level" style sheets affect all the same tags within that document, except for tags that contain an overriding inline style attribute

iii. External Style Sheets

You may also place style definitions, like our document-level style sheet example for the <h1> tags, into a text file with the MIME type of text/css and import this "external" style sheet into your documents. Because an external style sheet is a separate file and is loaded by the browser over the network, you can store it anywhere, reuse it often, and even use others' style sheets. But most important, external style sheets give you the power to influence the display styles not only of all related tags in a single document, but for an entire collection of documents.

For example, suppose we create a file named gen_styles.css containing the style rule:

h1 {color: blue; font-style: italic}

For each and every one of the documents in our collections, we can tell the browser to read the contents of the gen_styles.css file, which in turn will color all the <h1> tag contents blue and render the text in italic. Of course, that will be true only if the user's machine is capable of these style tricks, they are using a styles-conscious browser like Netscape or Internet Explorer, and the style isn't overridden by a document-level or inline style definition.

ii. Linked external style sheet

One way to load an external style sheet is to use the <link> tag:

```
 <head>
<title>Style linked</title>
<link rel=stylesheet type="text/css"
    href="http://www.kumquats.com/styles/gen_styles.css"
    title="The blues">
</head>
<body>
<h1>I'm so bluuuuoooo!</h1>
...
<h1> I am ba-looooo, tooooo!<h1>
```

Recall that the <link> tag creates a relationship between the current document and some other document on the Web. In the example, we tell the browser that the document named in the href attribute is a stylesheet and that its contents conform to the CSS2 standard, as indicated by the type attribute. We also provide a title for the style sheet, making it available for later reference by the browser. Section 6.7.2, "The <link> Header Element"

v.. Imported external style sheets

The second technique for loading an external style sheet imports the files with a special command (aka "at-rule") within the <style> tag:

```
 <head>
<title>Imported style sheet</title>
<style>
  <!--
    @import url(http://www.kumquats.com/styles/gen_styles.css);
    @import "http://www.kumquats.com/styles/spec_styles.css";
    body {background: url(backgrounds/marble.gif)}
  -->
</style>
</head>
```

The @import at-rule expects a single URL parameter that names the network path to the external style sheet. The URL may be a string enclosed in double-quotes and ending with a semicolon, or be the contents of the url keyword, enclosed in parentheses and with a trailing semicolon. The URL may be absolute or relative to the document's base URL.

The @import at-rule must appear before any conventional style rules, either in the <style> tag or in an external style sheet. Otherwise, the standard insists that the browser ignore the errant @import. By first importing all the various style sheets, then processing document-level style rules, the CSS2 standard cascades: the last one standing wins. Section 8.4.1.4, "URL property values"

The @import at-rule can appear in a document-level style definition or even in another external style sheet, letting you create nested style sheets.

vi.. Media-Specific Styles

Besides the media attribute for the <style> tag, the CSS2 standard has two other features that let you apply different style sheets depending on the agent or device that will render your document. This way, for instance, you can have one style or whole style sheet take effect when your document gets rendered on a computer screen, and another set of styles for when the contents get punched out on a braille printer. And what about those cell phones on the Web?

Like the media attribute for the style tag that affects the entire style sheet, you can specify whether the user's document processor[50] will load and use an imported style sheet.

[50]A.k.a. "user agent." Web documents get rendered on all kinds of devices these days, including the popular browser, braille printers, televisions, and projectors, to name just a few.

Do that by adding a media-type keyword or a series of comma-separated keywords to the end of the @import at-rule. For instance, the following example lets the user-agent decide to import and use the speech-synthesis style sheet or a common PC display and print style sheet if it is able to render the specified media types:

@import url(http://www.kumquats.com/styles/visual_styles.css) screen,print;

@import "http://www.kumquats.com/styles/speech_styles.css" aural;

The CSS2 media types include all, aural (speech synthesizers, for example), braille (tactile), embossed (braille printers), handheld, print, projection, screen, tty (fixed-width fonts), and tv.

Another CSS2 way to select media is through the explicit @media at-rule, which lets you include media-specific rules within the same style sheet, either at the document level or in an external style sheet. At the document level, like @import, the @media at-rule must appear within the <style> tag contents. And the at-rules may not appear within another rule. Unlike @import, @media may appear subsequent to other style rules, and indeed its style-rule contents may override previous rules according to the cascading standard.

The contents of @media include one or more comma-separated media-type keywords followed by a curly-braces ({}) enclosed set of style rules. For example:

```
body {background: white}

@media tv, projection {

    body {background: lt_blue}

    }
```

The general style rule for the document's body background color of white gets changed to light blue by the one within the @media at-rule, but only if the document gets rendered on a television or projection system instead of some other medium. (Notice the extra set of curly braces that contain the @media style rules?)

b. What is meant by Fully Qualified Domain Name (FQDN)? Distinguish between primary DNS and Secondary DNS. Give examples. (8)

Answer:

A fully qualified domain name (FQDN), sometimes also referred as an absolute domain name, is a domain name that specifies its exact location in the tree hierarchy of the

Domain Name System (DNS). It specifies all domain levels, including the top-level domain and the root zone. A fully qualified domain name is distinguished by its lack of ambiguity: it can only be interpreted one way. FQDNs first arose out of the need for uniformity as the Internet was quickly growing in size in the late 1980s.

For example, given a device with a local hostname myhost and a parent domain name example.com, the fully qualified domain name is myhost.example.com. The FQDN therefore uniquely identifies the device —while there may be many hosts in the world called myhost, there can only be one myhost.example.com. In the Domain Name System, and most notably, in DNS zone files, a fully qualified domain name is specified with a trailing dot. For example,

somehost.example.com.

specifies an absolute domain name that ends with an empty top level domain label.

The DNS root domain is unnamed, which is expressed by an empty label, resulting in a domain name ending with the dot separator. However, many DNS resolvers process a domain name that contains a dot in any position as being fully qualified or add the final dot needed for the root of the DNS tree. Resolvers process a domain name without a dot as unqualified and automatically append the system's default domain name and the final dot.

Some applications, such as web browsers, try to resolve the domain name part of a Uniform Resource Locator (URL) if the resolver cannot find the specified domain or if it is clearly not fully qualified by appending frequently used top-level domains and testing the result. Some applications, however, never use trailing dots to indicate absoluteness, because the underlying protocols require the use of FQDNs, such as Simple Mail Transfer Protocol (SMTP, an e-mail protocol).

By definition, a primary DNS server holds the "master copy" of the data for a zone, and secondary servers have copies of this data which they synchronize with the primary through zone transfers at intervals or when prompted by the primary.

Only one DNS server should be configured as primary for a zone, but you can have any number of secondary servers for redundancy.

Both primary and secondary servers for a zone serve exactly the same data to clients.

Because of this you could easily "simulate" a secondary server on a single computer with 2 IP addresses.

Simply configure the zone (as primary), and the server will function as both the primary (on one IP address) and secondary (on the other IP address).

The recommended practice is to configure the primary and secondary DNS servers on separate machines, on separate Internet connections, and in separate geographic locations (for the purpose of redundancy).

You can use a secondary DNS service such as www.ns2service.net - you will still have full control over your domains as you run the primary DNS server.

Many new "broadband" Internet connections (such as cable modems and DSL) only come with one IP address, so this setup is often used not so much because of redundancy, but because the registrar requires two DNS servers (with separate IP addresses).

When using separated primary and secondary DNS servers, zone transfers are used to synchronize the zone data from the primary DNS server to the secondary server(s).

With other DNS server software, a zone must initially be created on both the primary and secondary servers (creating individual DNS records and any subsequent changes to a zone need only be done on the primary server).

However, Simple DNS Plus has a unique option to automatically create and remove zones on secondary servers whenever you do this on the primary.

We call this a "Super Master/Slave" pair and is configured through the Options dialog / DNS / Super Master/Slave section.

Both servers must be running Simple DNS Plus (no other DNS servers we know of currently support this).

The secondary server must be listed as a "slave" on the primary server, and the primary server must be listed as a "master" on the secondary.

One Simple DNS Plus server can be master and/or slave for any number of other Simple DNS Plus servers.

To create the zone on the primary server, you can use the Quick Zone Wizard.

If you are not using the Super Master/Slave setup, or if either of your DNS servers are not Simple DNS Plus, you will also need to create the zone on the secondary server.

Use the New Zone function, select the "Secondary Zone" option, and specify the zone name and the IP address of the primary DNS server.

Once a zone is configured on both primary and secondary servers, zone transfers should automatically occur when needed.

To verify, use the Look Up function against the secondary server, or check the records on the secondary server through the DNS Records window on that server.

You can later change the primary/secondary status using the Zone Properties dialog.

The Zone Properties dialog "zone transfers" tab can be used to secure the zone, so only authorized secondary servers are allowed to request zone transfer.

Q.8 a. What is Design? What are elements of design? Explain. (2+6)

Answer:

Page No. 107, 109 – 110

b. How is a table created in XHTML? Give sample code and explain the concept of positioning tables and grouping rows/columns in it. (8)

Answer:

Example

<table border="1" cellpadding="1" cellspacing="2" summary="This table charts the number of cups of coffee consumed by each person, the type of coffee (decaf or regular), and whether taken with sugar.">

<caption>Cups of coffee consumed by each person</caption>

<tr>

<th>Name</th>

<th>Cups</th>

<th>Type</th>

<th>Sugar</th>

</tr>

<tr>

<td>Wendy</td>

<td>10</td>

<td>Regular</td>

<td>yes</td>

</tr>

<tr>

<td>Jim</td>

```
<td>15</td>
<td>Decaf</td>
<td>no</td>
</tr>
</table>
```

Q.9 a. Write a CGI code to process a sample form and to send it by mail. Explain the attributes also. (8)

Answer:

```
$ENV{PATH} = '';
sendmail(
   'Target <to@perlmaven.com>',
   'hello world',
   'submitted: ' . Dumper(\%data),
   'Source <from@perlmaven.com>');
 sub sendmail {
   my ($tofield, $subject, $text, $fromfield) = @_;
   my $mailprog = "/usr/lib/sendmail";
   open my $ph, '|-', "$mailprog -t -oi" or die $!;
   print $ph "To: $tofield\n";
   print $ph "From: $fromfield\n";
   print $ph "Reply-To: $fromfield\n";
   print $ph "Subject: $subject\n";
   print $ph "\n";
   print $ph "$text";
   close $ph;
   return ;
}
```

b. Implement the following logic in JavaScript: Click on the "Start count!" button above to start the timer. The input field will count

forever, starting at 0. Click on the "Stop count!" button to stop the counting. Click on the "Start count!" button to start the timer again. (8)

Answer:

```
var c=0;

var t;

var timer_is_on= false;

function timedCount() {

   document.getElementById('txt').value=c;

   c++;

   if (timer_is_on) {

      t= setTimeout(timedCount,1000);

   }

}

function doTimer() {

   if (!timer_is_on) {

      timer_is_on=true;

      timedCount();

   }

   else {

      clearTimeout(t);

      timer_is_on=false;

   }

}
```

TEXT BOOK

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