# JAVA IAT 3 SCHEME & SOLUTIONS

1. What is exception? Explain the different types of exception handling mechanisms with an example.

This is the general form of an exception-handling block:

```
try {
  // block of code to be mentioned for errors
}
catch (ExceptionType1 exOb){
   // Exception handler for ExceptionType1
}
catch (ExceptionType2 exOb){
   //exception handler for ExceptionType2
}
//...
finally
{
   //block of code to be executed before try block ends
}
```

DIVIDE-BY-ZERO EXCEPTION

The Java run-time system constructs a new exception object when it detects an attempt to divide-by-zero. It then throws exception. If there is no exceptional handler to catch the exception, then default handler provided by the java run-time system will process any exception that is not caught by the program.

The following program includes a try block and a catch clause, which processes the Arithmatic Exception generated by the divide-by-zero:

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

```
int d, a;
try
{
    d = 0;
    a = 42/d;
    System.out.println("This will not be printed.");
}
catch (ArithmeticException e)
{
    System.out.println("Division by zero.");
}
System.out.println("After Catch Statement.");
}
```

This program generates the following output:

Division by zero.

After catch Statement.

The call to println() inside the try block is never executed. Once an exception is thrown, program control transfers out of the try block. Once the catch statement has executed, the program continues with the next line in the program following the entire try/catch mechanism.

(2) What is synchronization? Explain the producer and consumer problem with a program.

When 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.

The producer-consumer problem is a classical example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed buffer size.

The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and the consumer won't try to remove data from an empty buffer.

```
class BufferItem {
  public volatile double value = 0;
    = false; }

class BoundedBuffer {
 thread
 private int numSlots = 0;
 private BufferItem[] buffer = null;
 private int putIn = 0, takeOut = 0;
 public BoundedBuffer(int numSlots){
     if (numSlots <= 0) throw new IllegalArgumentException("numSlots <= 0");
        this.numSlots = numSlots;
  buffer = new BufferItem[numSlots];
  for(int i=0; i<numSlots; i++) buffer[i] = new BufferItem();
  putIn = (putIn + 1) % numSlots;
}
public double fetch() {
    double value;
    while (!buffer[takeOut].occupied)
    value = buffer[takeOut].value;
} buffer[takeOut].occupied = false;
```

(3) Discuss the various steps of JDBC process with suitable exception handling blocks.

The JDBC process can be broadly divided into 5 routines.

(i) Loading the JDBC driver
(ii) Connecting to the DBMS
(iii) Creating and executing a statement
(iv) Processing data returned by the DBMS
(v) Terminating the connection with DBMS

Loading the JDBC driver is the first step in the JDBC process. The class.forName() method is used to load the JDBC driver. The name of the driver to be loaded must be specified as shown belo

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

After the driver is loaded, the J2EE component must connect to the DBMS. The DriverManager.getConnection() method is used for this purpose. The DriverManager.getConnection() method must be given the url of the database along with the userid and password (if required). The driverManager.getConnection() returns a connection interface which is used by the J2EE component to reference the database.

```
String url = "jdbc:odbc:CustomerInformation";
String userId = "jim"
String password = "keogh";
Statement holder;
Connection con;
try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    con = DriverManager.getConnection(url, userId, password);
}
```

After loading the driver and connecting to the database, the next step would be to send an SQL query from the J2EE component to the DBMS for execution. The createStatement() method is used to create a statement object. This object is then used to execute a query using executeQuery() method. The result sent by the DBMS would be collected in a ResultSet Object as shown below.

```
Statement holder;
ResultSet result;
try
{
    String s= "SELECT * FROM CUSTOMERS"
    holder = con.createStatement();
    result = holder.executeQuery(s);
}
```

After receiving the result from the DBMS, the J2EE component would process the data (normally by displaying the data). The ResultSet object has methods such as the next() method and getXXX() method using which the received information can be processed as shown below:-

```
ResultSet result;
String Fname;
String Lname;
while (result.next()==true)
{
    Fname = result.getString(1);
    Lname = result.getString(2);
    System.out.println(Fname+" "+Lname);
}
```

Finally after the J2EE component finishes accessing the database, the connection with the database must be terminated using the close() method of the connection object as shown below.

(4) What is a cookie? List out the methods defined by cookie. Write a program to add a cookie

Cookies refer to a small piece of information that is stored on the client's hard-disk by the browser. Consider the following html file.

```
<html>
 <body>
   <center>
      <form name = "Form"
      action = " http://localhost:8080/servlets-examples/servlet/
                                AddCookieServlet ">
      <B>Enter a value for mycookie </B>
     <input type = "text box" name = "data" size = 25 value = " ">
     <input type = submit value = "submit">
 </form>
 </body>
</html>
```

The above html file contains a text field in which a value can be entered. There is also a submit button which must be pressed, the value of the text field is sent to the AddCookieServlet via a http request. The source code for AddCookieServlet is as shown below.

```
import java.io.*;
import javax.servlet.*;
import java.servlet.http.*;
public class AddCookieServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse
                                                    response)
    throws ServletException, IOException.
    {
```

```
        String data = request.getParameter("data");
        Cookie cookie = new cookie("myCookie", data);
        response.addCookie(cookie);
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>MyCookie has been set to");
        pw.println(data);
        pw.close();
    }
}
```

(5) What is an applet? With a skeletal code explain the methods that constitute the life cycle of an applet.

An applet is a program. Applets are not standalone programs, ie they cannot exist independently. Rather, they have to be embedded within a html file.

The Browser can interface with the applet and control its execution by over-riding a set of functions such as init(), start(), paint(), stop() and destroy(). These 5 functions can be assembled into a skeleton as shown below.

```
import java.awt.*;
import java.applet.*;
public class AppletSkel extends Applet
{
    public void init()
    { }
    public void start()
    {
    }
    public void paint(Graphics g)
    {
    }
```

```
    public void stop()
    {

    }
    public void destroy()
    {

    }
}
```

when the applet begins, the following functions are called in a sequence -

init() start() and paint()

when the applet is terminated, the following functions are called in a sequence.

stop() destroy()

init() method is the first method to be called. This method is normally used to initialize variables. This method is called only once during the life-time of the applet.

start() method is the second method that gets called soon after init(). It also gets called each time after the applet has been stopped and again restarted. So if the user leaves the web page and comes back, then the applet would resume its execution at the start().

paint() method is called soon after the start() method. It accepts one parameter of type graphics. This parameter would contain the graphics reference which would describe the graphics environment in which the applet is running. This context would be used whenever the output to the applet is required.

stop() : method is called when a web browser leaves the HTML document containing the applet and goes to another page. This function is normally used to suspend such threads that don't need to run when the applet is not visible.

destroy() method is called when the run-time environment decides to completely remove the applet from memory. This method must be used to release all resources that the applet is using.

(6) Describe the simple html file to pass the parameter to servlet and display the parameter values accepted by servet.

The servlet can read the name and values of parameters that are given by the client using the ServletRequest interface. Consider the following example.

```
<html>
  <body>
    <form name = "form"
      action = "http://localhost:8086/examples/servlet/ColorGetServlet1">
     <B>Color :</B>
     <select name = "color" size = "1">
     <option value = "Red">Red</option>
     <option value = "Green">Green</option>
     <option value = "Blue">Blue</option>
     </select>
     <br><br>
     <input type = submit value = "submit">
    </form>
  </body>
</html>
```

The above html file when opened using a browser would provide various color options out of which a color must be selected and the submit button is pressed. This would invoke the colorgetServlet1 class file which would be made available to the server. The corresponding java file is as shown below.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorGetServlet1 extends GenericServlet
{
    public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException
    {
        String color = request.getParameter("color");
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B> The selected color is : ");
        pw.println(color);
        pw.close();
    }
}
```

The above servlet has been designed by extending the genericServlet class and by overriding the service() method. The above servlet uses the getParameter() method to obtain the selection made by the client. It would then formulate a response and sends it to the browser. However, it is important to note that a servlet can also be designed by extending the HttpServlet class and by overriding the doget() or dopost() methods.