Advanced Java and J2EE

Sem: 06 Subject code: 18CS644

Staff: Saleem. H

Q1a. What are Enumerations? Explain values() and valueOf() methods with an Example program

A: An enumeration is a list of named constants. An Enumeration defines a class Type. In java an Enumeration can have constructors, methods and instance variables. An Enumeration is created using the enum keyword

The values() and valueOf() methods

The General forms are:

```
public static enum_type[] values()
public static enum_type valueOf(String str)
```

The values() method returns an array that contains a list of the enumeration constants. The valueOf() method returns the enumeration constant whose value corresponds to

the String passed in str.

The following program demonstrates the values() and valueOf() methods:

```
enum Apple
  {
    Jonathan, GoldenDel, RedDel,
    Winesap, CostLand
  }

class EnumDemo2
  {
    public static void main(String args[])
    { Apple ap;
      System.out.println("Here are all Apple
                          Constants");
      Apple allapples[]= Apple.values();
      for(Apple a : allapples)
          System.out.println(a);

      System.out.println();
      ap=Apple.valueOf("Winesap");
      System.out.println("ap contains "+ap);
    }
}
```

7b.) What is Autoboxing ? write a Java program that demonstrates how autoboxing and unboxing takes place in expression evaluation

Ans:

Autoboxing is the process by which a primitive type is automatically encapsulated (boxed) into its equivalent type wrapper whenever an object of that type is needed. There is no need to explicitly construct an object.

```
Class AutoBox3
{
  public static void main(String args[])
  {
    Integer iob, iob2;
    int i;
    iob=100;
    System.out.println(iob);
    ++iob;
    System.out.println(iob);
    iob2 = iob + (iob/3);
    System.out.println(iob2);

    i = iob + (iob/3);
    System.out.println(i);
  }
}
```

Output

```
100
101
134
134
```

1c) What are Annotations? Explain the following Built-in annotations with an example program: @Override @Inherited @Retention

Ans: Java allows to embed supplemental information into a source code (file). This information called an annotation does not change the actions of a program. Thus an annotation leaves the semantics of a program unchanged. However, this information can be used by various tools during both development and deployment.

@override: This is a marker annotation that can be used only on methods. A method annotated with @override must override a method from a superclass. If it doesn't a compile time error will result.

Example:

```
class Base
  { public void display()
      {
        System.out.println("Base display()");
      }
  }
```

```
class Derived extends Base
{
  @Override
    public void display(int x)
    {
      System.out.println("Derived display");
    }
  public static void main(String args[])
    { Derived ob = new Derived();
      ob.display();
} }
```

@Inherited : This is a marker annotation that can be used only on another annotation declaration. @Inherited causes the annotation for a superclass to be Inherited by a subclass

Example :

```
@Inherited
@Interface A{}
@Interface B{}
@A
@B
class superclass
    {...}
Class subclass extends superclass
    {---}
```

@Retention : is designed to be used only as an annotation to another annotation. It specifies the retention policy

The General form is :

```
@Retention (retention-policy)
```

Example: An Example that creates and uses a single-member annotation

```
@Retention ( RetentionPolicy. RUNTIME)
@ interface MySingle
  { int value();
  }
class Single
  { @MySingle (100)
    Public static void myMeth()
     { Single ob = new Single();
       try { method m = ob.getClass().getMethod("myMeth");
             mySingle anno = m.getAnnotation (mySingle.class);
             System.out.println (anno.value());
           }
       catch ( NoSuchMethod Exception exe)
         {
           System.out.println ("Method Not Found");
         }
     }
  public static void main (String args[])
     { myMeth();
     }
}
```

Q2a Explain the following methods of java.lang.Enum with an Example program

(i) ordinal() (ii) compareTO() (iii) equals()

Ans: One can obtain a value that indicates an enumeration constant's position in the list of constants. This is called its ordinal value, and it is retrieved by calling the ordinal() method, shown here:

```
final int ordinal()
```

It returns the ordinal value of the invoking constant, ordinal values begin at zero.

ii) One can compare the ordinal value of two constants of the same enumeration by using the compareTO method.

It has this general form

```
final int compareTO( enum-type e)
```

Here, enum-type is the type of the enumeration and e is the constant being compared to the invoking constant

(iii) equals() : Two Enumeration class objects compared for equality. Returns True if both are same else false.

Program Example

```
enum Apple
  {
    Jonathan, GoldenDel, RedDel
    Winesap, Costland.
  }
class EnumDemo4
  {
    public static void main(String args[])
      {
        Apple ap, ap2, ap3;
        System.out.println("Here are all apple
          Constants" + "and their ordinal
          values");

        for (Apple a : Apple.values())
          System.out.println(a + " " + a.ordinal());
        ap = Apple.RedDel;
        ap2 = Apple.GoldenDel;
        ap3 = Apple.RedDel;
        System.out.println();
        // Demo of compareTO()
        if (ap.compareTo(ap2) < 0)
          System.out.println(ap + "comes before" + ap2);
        if (ap.compareTo(ap2) > 0)
          System.out.println(ap2 + "comes before" + ap);
        if (ap.compareTo(ap3) == 0)
          System.out.println(ap + "equals" + ap3);
```

```
System.out.println();
if (ap.equals(ap2))
    System.out.println ("Errors");
if (ap.equals(ap3))
    System.out.println(ap + "equals" + ap3);
if (ap==ap3)
    System.out.println(ap + "==" + ap3);
}
```

2b. Explain how to obtain Annotations at Run Time by use of Reflection

Ans: Reflection is the feature that enables information about a class to be obtained at run time. The reflection API is contained in the java.lang.reflect package

There are a number of ways to use reflection; The first step to using reflection is to obtain a class object that represents the class whose annotations you want to obtain.

There are various ways to obtain a class object. one of them easiest is to call getClass() which is a method defined by Object. Its general form is shown here:

final class getClass()

It returns the class Object that represents the invoking object.

To obtain the annotations associated with Method, you first obtain a class object that represents the class, and then call getMethod() on that class object specifying the name of the method. getMethod() has this general form:

Method getMethod(String methName, Class ... param)

From a class, Method, Field or Constructor object, you can obtain a specific annotation associated with that object by calling getAnnotation(). Its general form is shown here:

Annotation getAnnotation(Class annoType)

Here, annoType is a class object representing the annotation in which you are interested.

Here, is a program that assembles all of the pieces shown earlier and uses reflection to display the annotation associated with a Method

```
import java.lang.annotation.*;
import java.lang.reflect.*;
// An annotation type declaration
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno
{
    String str();
    int val();
}
class Meta
{
    @MyAnno(str="Annotation Example", val=100)
    public static void myMeth()
    {
        Meta ob=new Meta();
        try
        {
            Class c= ob.getClass();
            Method m= c.getMethod("myMeth");
            MyAnno anno = m.getAnnotation(MyAnno.class);
            System.out.println(anno.str() + " " +
                               anno.val());
        }
        catch(NoSuchMethodException exe)
        {
            System.out.println("Method Not Found");
        }
    }
    public static void main(String args[])
    {
        myMeth();
    }
}
```

Q3a What is collection framework? Explain the methods defined by collection interface

The Collections Framework is a sophisticated hierarchy of interfaces and classes that provide state-of-the-art technology for managing groups of objects.

The Java Collections Framework standardizes the way in which groups of object are handled by the programs

The collection interface declares the core Methods

(1) boolean add (E obj)

- Adds obj to the invoking collections. Returns true if obj was added to the collection. Returns false if obj is already a member of the Collection and the collection does not allow duplicates

(2) boolean addAll (Collection <? extends E> c)

- Adds all the elements c to the invoking collection. Returns true if the operation succeeded (i.e the elements were added). Otherwise returns false.

(3) void clear() — Removes all elements from the invoking Collection

(4) boolean contains(Object obj) — Returns true if obj is an element of the invoking Collection. Otherwise, returns false.

(5) boolean equals(Object obj) — Returns true if the invoking collection and obj are equal; otherwise, returns false

(6) boolean isEmpty() — Returns true if the invoking Collection is empty otherwise, returns false.

(7) int size() :- Returns the no. of elements held in the invoking collection

(8) int hashCode(); Returns the hash code for the invoking Collection.

(9) boolean remove(Object obj): Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false

(10) Object[] toArray(): Returns an array that contains all the elements stored in the invoking collection.

Q3b. Explain the constructors of HashSet class with an example program

A:- The following constructors are defined

(1) HashSet() :- This form constructs a default hash set.

(2) HashSet(Collection<? extends E> c) :- This form initializes the hash set by using the elements of C.

(3) HashSet(int capacity) :- This form initializes the capacity of the hash set to capacity. (default capacity is 16).

(4) HashSet(int capacity, float fillRatio) :- This form initializes both the capacity and the fill ratio (also called load capacity) of the hash set from its arguments. The fill ratio must be between 0.0 and 1.0 and determines how full the hash set can be before it is resized upward. For constructors that do not take a fill ratio, 0.75 is used.

Here is an example that demonstrates HashSet:

```
// Demonstrate HashSet
import java.util.*;
class HashSetDemo
{
  public static void main(String args[])
  { // Create a hash set
    HashSet<String> hs = new HashSet<String>();
    // Add elements to the hash set
    hs.add("B");
    hs.add("A");
    hs.add("D");
    hs.add("E");
    hs.add("C");
    hs.add("F");
    System.out.println(hs);
  }
}
```

Q4a) Explain the constructors of Treeset class and write java program to create TreeSet collection.

Ans:- TreeSet has the following Constructors:-

(1) TreeSet() :- This form of constructor Constructs an empty treeset that will be sorted in ascending order according to the natural order of its elements.

(2) TreeSet(Collection<? extends E> C) :-
This form builds a tree set that contains the elements of C

(3) TreeSet(Comparator<? super E> comp):
This form Constructs an empty tree set that will be sorted according to the Comparator Specified by Comp.

(4) TreeSet(SortedSet<E> ss):
This form builds a treeset that contains the elements of ss

Here, is an Example that demonstrates a TreeSet.

```
// Demonstrate TreeSet
import java.util.*;
class TreeSetDemo
{
    public static void main(String args[])
    {
        // Create a tree set
        TreeSet<String> ts = new TreeSet<String>();
        // Add elements to the tree set
        ts.add("C");
        ts.add("A");
        ts.add("B");
        ts.add("E");
        ts.add("F");
        ts.add("D");
        System.out.println(ts);
    }
}
```

Q 4b> Explain any Four Legacy classes of Java's collection Framework

(1) vector : vector implements a dynamic array. It is similar to ArrayList and it contains many Legacy methods that are not past of the collections Framework.

With the Advent of collections, Vector was reengineered to extend AbstractList and to implement the List Interface.

Vector is declared as follows:

class Vector<E>

Here, E specifies the type of element that will be stored.

Here are the Vector Constructors:-

Vector() // initial size 10

Vector(int size)

Vector(int size, int incr)

Vector(Collection<? Extends E> c)

Methods defined by vector class:

void addElement(E element)

int capacity() - Returns the capacity of vector

Stack : Stack is a subclass of Vector that implements a standard last-in, first-out stack. Stack only defines the default constructor, which creates an empty stack.

Stack is declared as follows:

```
class Stack<E>
```

Here, E specifies the type of element stored in the stack.

The methods defined by stack are :-

Methods

boolean empty() :- Returns true if the stack is empty, and returns false if the stack contains elements

E peek() :- Returns the element on top of the stack but does not remove it.

E pop() :- Returns the element on the top of the stack removing it in the process.

E push(Element) : pushes element onto the stack, element is also returned

Dictionary

Dictionary is an abstract class that represents a key/value storage repository and operates much like Map. Given a key and value you can store the value in a Dictionary object. Once the value is stored, you can retrieve it by using its key.

It is declared as

class Dictionary<K, V>

Here, K specifies the type of keys and V specifies the type of values.

The Abstract methods defined by Dictionary

(1) Enumeration<V> elements() :-
Returns an enumeration of the values contained in the dictionary

(2) V get(Object key) : Returns the object that contains the value associated with key. If key is not in the dictionary, a null object is returned.

(3) int size() - Returns the number of entries in the dictionary

Hashtable

Hashtable was part of the original java.util and is a concrete implementation of a Dictionary. Hashtable was reengineered to also implement the Map interface.

Hash table is declared as:

class Hashtable <K, V>

Here, K specifies the type of keys and V specifies the type of values

Hash table constructors are:

Hashtable()
Hashtable (int size)
Hashtable (int size, float fillRatio)
Hashtable (Map <? extends K, ? extends V> m)

Methods defined

void clear() — Resets and empties the Hash table

Object clone() — Returns a duplicate of the invoking object

boolean isEmpty() — Returns true if the hash table is empty; returns false if it contains at least one key

int size() — Returns the number of entries in the hash table

Q5a what is string in java? write a java program that demonstrates any four constructors of String class

Ans:— String is a sequence of characters. Java implements strings as objects of type String

String Constructors

(1). The string class supports several constructors

(1) To create an empty string you call the default constructor.

e.g

```
String s = new String();
```

(2) To create a string initialized by an array of characters, use the constructor shown here:

```
String (char chars[])
```

Here is an example:

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
```

(3) You can specify a subrange of a character array as an initializer using the following constructor:

```
String(char chars[], int startIndex, int numChars)
```

For example:

```
char chars[] = {'a', 'b', 'c', 'd', 'e', 'f'};
String s = new String(chars, 2, 3);
```

(b) You can construct a String object that contains the same character sequence as another String object using this constructor:

```
String(String strObj)
```

Here, strObj is a String object.

Consider this example:

```
// Construct one string from another
class MakeString
{ public static void main(String args[])
  { char c[] = {'J', 'a', 'v', 'a'};
    String s1 = new String(c);
    String s2 = new String(s1);
    System.out.println(s1);
    System.out.println(s2);
  }
}
```

5b. Differentiate between equals() and == with respect to String Comparison

Ans:- equals() versus ==

It is important to understand that 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 reference 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));
  }
}
```

Output:

```
Hello equals Hello -> true
Hello == Hello -> false
```

5c. Explain any Two character extraction Methods of String class

The string class provides a no. of ways in which characters can be extracted from a string object.

(1) To Extract a Single Character from a String, you can refer directly to an individual character via the charAt() method. It has general form:

```
char charAt(int where)
```

Here, where is the index of the character that you want to obtain.

Example :-

```
char ch;
ch = "abc".charAt(1);
```

(2) getChars()

If you need to extract more than one character at a time, you can use the getChars() method. It has following general form :

```
void getChars(int sourceStart, int sourceEnd,
              char target[], int targetStart)
```

Example :

```
class getCharsDemo
{
  public static void main(String args[])
  {
    String s = "This is a demo of the
        getChars method.";
    int start = 10;
    int end = 14;
    char buf[] = new char[end - start];
    s.getChars(start, end, buf, 0);
    System.out.println(buf);
  }
}
```

Q6a> Explain any four String modification Methods of String class

Ans :-

Substring :- You can extract a substring using substring(). It has two forms. The first is

String substring(int startIndex)

Here, startIndex specifies the index at which the substring will begin. This form returns a copy of substring that begins at startIndex and runs to the end of the invoking String.

The second form of substring() allows you to specify both the beginning and ending index of the substring:

String substring(int startIndex, int endIndex);

concat() : You can concatenate two strings using concat() shown here:-

String concat(String str)

This method creates a new object that contains the invoking string with the contents of str appended to the end.

e.g
```
String s1 = "one";
String s2 = s1.concat("two");
```

replace(): The replace() method has two forms. The first replaces all occurrences of one character in the invoking string with another character. It has the following general form:

String replace(char original,
char replacement)

Here, original specifies the character to be replaced by the character specified by replacement.

e.g String s = "Hello".replace('l', 'w');

— This puts the String "Hewwo" into s

The second form of replace is

String replace(CharSequence original,
CharSequence replacement)

trim(): The trim() method returns a copy of the invoking String from which any leading and trailing whitespace has been removed. It has following General form:

String trim()

e.g:-

String s = "Hello world ".trim();

6b. Explain the following methods of StringBuffer class: 15

(i) append() (ii) insert() (iii) reverse()
(iv) replace

Ans :- **append()** :- The append() method concatenates the string representation of any other type of data to the end of the invoking StringBuffer object. It has several overloaded versions. Here are a few of its forms:

```
StringBuffer  append(String str)
StringBuffer  append(int num)
StringBuffer  append(Object obj)
```

Program :

```
class appendDemo
{ public static void main(String args[])
  { String s;
    int a=42;
    StringBuffer sb = new StringBuffer(40);
    s= sb.append("a=").append(a).
        append("!").toString();
    System.out.println(s);
  }
}
```

output :
a=42!

(ii) insert() :- The insert() method inserts one string into another.

The Different forms of insert are :-

```
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)
```

The following sample program inserts "like" between "I" and "java".

```
// Demonstrate insert()
class insertDemo
{
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("I java");
        sb.insert(2, "like");
        System.out.println(sb);
    }
}
```

Output
I like java

(iii) reverse() :

You can reverse the characters within a stringbuffer object using reverse() shown here:

```
StringBuffer reverse()
```

This method returns the reversed object on which it was called.

Program Example

```
class ReverseDemo
{ public static void main(String args[])
  { StringBuffer s = new StringBuffer("abcdef");
    System.out.println(s);
  }
}
```

output

```
abcdef
fedcba
```

(iv) replace(): You can replace one set of characters with another set inside a StringBuffer object by calling replace()

Its signature is shown below:

```
StringBuffer replace(int startIndex,
                     int endIndex, String str)
```

The following program demonstrates replace()

```
class replaceDemo
{ public static void main(String args[])
  { StringBuffer sb = new StringBuffer("This
                  is a test");
    sb.replace(5, 7, "was");
    System.out.println("After replace:"+sb);
  }
}
```

7a. Explain the differences between Servlets and CGI

Ans:-

CGI	Servlet
1. Platform dependent	1. Does not rely on the platform
2. Each client request creates its own process	2. processes are created depending on the type of client request
3. Runs on Separate process	3. Runs on JVM
4. Slower	4. Faster
5. CGI is process based	5. Servlet is thread based
6. More vulnerable to attacks	6. less vulnerable
7. Less powerful	7. More powerful [Support Extensive Infrastructure]

7b: Write a Java servlet program that demonstrates how parameters can be accessed from HTML.

Ans:-

PostParameters.htm

```
<html>
<body>
<form name = "form1"
method = "post"
action = "http://localhost:8080/
servlets-examples/servlet/
postParametersServlet">

<table>
<tr>
<td> Employee </td>
<td> <input type=textbox
name= "e"
size = "25"
value = ""> </td>
</tr>
<tr>
<td> phone </td>
<td> <input type= textbox
name= "p"
size = "25"
value = "">
</td>
</tr>
</table>
<input type = submit
value = "submit">
</body>
</html>
```

PostParameterServlet.java

```
import java.io.*;
inpost java.util.*;
inpost javax.servlet.*;

public class postParameterServlet extends
        GenericServlet
        {
          public void Service(ServletRequest
                              request,
                              ServletResponse
                              response)
          throws ServletException, IOException
          {
          PrintWriter pw = response.getWriter();
          Enumeration e = request.getParameterNames();
          while(e.hasMoreElements())
          {
            String pname = (String)e.nextElement();
            pw.print(pname + " = ");
            String pvalue = request.getParameter(pname);
            pw.println(pvalue);
          }
        pw.close();
        }
}
```

7c. Explain any two cookies methods

① String getName() – Returns name
② String getValue() – Returns the value

Following program invokes the getCookies() method to read any cookies that are included in the HTTP GET request. The names and values of these cookies are then written to the HTTP response. The getName() and getValue() methods are called to obtain this information

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetCookiesServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,
      HttpServletResponse response)
    throws ServletException, IOException
  {
    Cookie[] cookies = request.getCookies();
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>");
    for(int i=0; i<cookies.length; i++)
    {
      String name = cookies[i].getName();
      String value = cookies[i].getValue();
      pw.println("name="+name+"; value="
        +value);
    }
    pw.close();
  }
}
```

8 a) Define Jsp. Explain different types of Jsp tags by taking suitable example.

Ans :- Jsp (Java Server page) is a server side program that is similar in design and functionality to a Java Servlet. A Jsp is called by a client to provide a web service. A Jsp is written in HTML, XML or in the client's format that is interspersed with Scripting elements, directives and actions comprised of Java programming language and Jsp Syntax.

There are Five Types of Jsp tags as Mentioned below:

1) Comment tag : A Comment tag opens with <%-- and closes with --%> and is followed by a Java comment that usually describes the functionality of statements that follow the Comment tag.

2) Declaration statement tags :- A declaration Statement tag opens with <%! and is followed by a Java declaration statement(s) that define variables, objects and methods that are available to other Components of the Jsp program.

(3) Directive tags : A directive tag opens with <%@ and commands the JSP virtual engine to perform a specific task such as importing a Java package required by objects and methods used in a declaration statement. The directive tag closes with %>
There are three commonly used directives.

1) <%@ page import = " import java.sql.*"; %>
2) <%@ include file = " keogh\books.html " %>
3) <%@ taglib uri = " myTags.tld " %>

(4) Expression tags : An expression tag opens with <%= and is used for an expression statement whose result replaces the expression tag when the JSP virtual engine resolves JSP tags. An expression tags closes with %>

(5) Scriptlet tags :- A scriptlet tag opens with <% and contains commonly used Java control statements and loops. A scriptlet tag closes with %>

Example of JSP program :-

```
<HTML>
<HEAD>
    <TITLE> JSP programming </TITLE>
</HEAD>
```

```
<Body>
  <%! int grade = 70; %>
  <% if ( grade >69} { %>
     <p> you passed </p>
    <% } else { %>
     <p> Better luck next time </p>
     <% } %>
    <% switch (grade) {
       case 90 : %>
         <p> your Final grade is a A <p>
       <% break ; %>
    <% case 80 : %>
          <p> your final grade is a B <p>
     <% break;
       case 70 : %>
          <p> your final grade is a C <p>
     <% break;
       case 60 : %>
         <p> your final grade is an F <p>
    <% break;
      }
  %>
 </body>
</Html>
```

8b. List and explain core classes and interfaces in javax.servlet package

Ans:- The Javax.servlet package contains a number of interfaces and classes that establish the framework in which servlets operate. The Interfaces are

Interface	Description
(1) Servlet	— Declares life cycle methods of a servlet
(2) ServletConfig	— Allows servlets to get initialization parameters
(3) ServletContext	— Enables servlets to log events and access information about their environment
(4) ServletRequest	— used to read data from a Client request
(5) ServletRequest	— used to write data to a client response

(1) Servlet Interface :

All servlets must implement the Servlet interface. It declares the init(), service() and destroy() methods that are called by the server during the life cycle of a servlet.

Methods defined by Servlet Interface are

(1) void destroy()

(2) ServletConfig getServletConfig()
(3) String getServletInfo()
(4) void init(ServletConfig sc) throws ServletException
(5) void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

(2) ServletConfig Interface : The ServletConfig interface allows a servlet to obtain configuration data when it is loaded

(3) ServletContext Interface : This interface enables servlets to obtain information about their environment. The methods defined by this ServletContext Interface are:

(i) Object getAttribute(String attr)
(ii) String getMimeType(String file)
(iii) String getRealPath (String vpath)
(iv) String getServerInfo()
(v) void log(String s)
(vi) void log(String s, Throwable e)
(vii) void setAttribute(String attr, Object val)

(4) The ServletRequest Interface

The ServletRequest interface enables a servlet to obtain information about a client request.

Some of the Methods defined by ServletRequest Interface are:

Method	Description
(1) Object getAttribute(String attr)	Returns the value of the attribute named attr.
(2) int getContentLength()	Returns the size of the Request. A null value is returned if the type cannot be determined.
(3) String getRemoteAddr()	Returns the string equivalent of the client IP address
(4) int getServerPort()	Returns the port number

Now, the list of core classes provided by Javax.servlet package are:

Class	Description
(1) GenericServlet	Implements the servlet and ServletConfig interfaces
(2) ServletInputStream	Provides an input stream for reading requests from a client
(3) ServletOutputStream	Provides an output stream for writing responses to a client

(4) ServletException — Indicates a servlet error occured

(5) UnavailableException — Indicates a servlet is unavailable.

The GenericServlet class : The GenericServlet class provides implementations of the basic life cycle methods for a servlet. GenericServlet implements the servlet and ServletConfig interfaces. In addition, a method to append a string to the server log file is available. The signature of this method is shown

```
void log(String s)
void log(String s, Throwable e)
```

Here, s is the string to be appended to the log and e is an exception that occurred

The ServletInputStream class :- This class extends InputStream. It is implemented by the servlet Container and provides an input stream that a servlet developer can use to read the data from a client request. It defines the default Constructor. In addition, a method is provided to read bytes from the stream. It is shown here:

int readLine(byte[] buffer, int offset, int size)
throws IOException

Here, buffer is the array into which size bytes are placed starting an offset. The method returns the actual number of bytes read or -1 if an end-of-stream condition is encountered.

The ServletOutputStream class : This class extends OutputStream. It is implemented by the servlet containes and provides an output stream a servlet developer can use to write data to a client response. A default constructor is defined. It also defines the print() and println() methods which output data to the stream.

The Servlet Exception classes

Javax.servlet defines two exceptions. The first is ServletException which indicates that a servlet problem has occured. The second is UnavailableException which extends ServletException

9a. Explain the Four types of JDBC drivers

Ans:- JDBC Driver Types

Type1 JDBC-to-ODBC Driver

- ✓ JDBC-to-ODBC driver also called the JDBC/ODBC Bridge is used to translate DBMS calls between the JDBC specification and the ODBC specification.
- ✓ The JDBC-to-ODBC driver receives messages from a J2EE Component that conforms to the JDBC specification. Those messages are translated by the JDBC-to-ODBC driver into ODBC message format, which is then translated into the message format understood by the DBMS.
- ✓ However avoid using the JDBC/ODBC in a mission-critical application because the extra translation might negatively impact performance.

Type 2 Java/Native code Driver

The Java/native code driver uses Java classes to generate platform-specific code i.e code only understood by a specific DBMS.

Then manufacturer of the DBMS provides both the Java/Native code driver and API classes so the J2EE Component can generate the platform-specific code. The obvious disadvantage of using a Java/native code driver is the loss of some portability of code. The API classes of for the Java/Native code driver probably won't work with another manufacturer's DBMS.

Type 3 JDBC Driver

The Type 3 JDBC driver, also referred to as the Java-protocol, is the most commonly used JDBC drives. The Type3 JDBC driver converts SQL queries into JDBC-formatted statements. The JDBC-formatted statements are translated into the format required by the DBMS.

Type 4 JDBC Driver

Type 4 JDBC driver is also known as the Type 4 database protocol. This driver is similar to the Type 3 JDBC driver except SQL queries are translated into the format required by the DBMS. SQL queries do not need to be converted to JDBC-formatted systems. This is the fastest way to communicate SQL queries to the DBMS.

9b. Describe the various steps of JDBC with Coden snippets

Ans :- Overview of the JDBC process

This process is divided into five routines. These include:—

- ✓ Loading the JDBC Driver
- ✓ Connecting to the DBMS
- ✓ Creating and executing a statement
- ✓ Processing data returned by the DBMS
- ✓ Terminating the connection with the DBMS

Loading the JDBC Driver

The JDBC Driver must be loaded before the J2EE Component can connect to the DBMS. The Class.forName() method is used to load the JDBC driver, as shown below:

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

Connect to the DBMS

Once the driver is loaded, the J2EE Component must connect to the DBMS using the DriverManager.getConnection() method.

The DriverManager.getConnection() method the URL of the database and the UserID and password if required by the DBMS.

The URL is a string object that contains the driver name and the name of the database that is being accessed by the J2EE Component. The DriverManager.getConnection() Method returns a Connection interface that is used throughout the process to reference the database.

Following snippet of code illustrates the use of the DriverManager.getConnection() method to load the JDBC/ODBC Bridge and Connect to the customerinfomation database.

```
String url = "jdbc:odbc:CustomerInfomation";
String userID = "jim";
String password = "keogh";
Statement DataRequest;
private Connection Db;
try{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     Db = DriverManager.getConnection(url, userID,
                                      password);
   }
```

Creating and Execute a SQL Statement

Now sends a SQL query to the DBMS for processing. A SQL query consists of a series of SQL Commands that direct the DBMS to do something such as to return rows of data to the J2EE component.

The Connect.createStatement() Method is used to create a Statement Object. The Statement Object is then used to execute a query and return a ResultSet object that contains the response from the DBMS

Typically, the query is assigned to a string object which is passed to the Statement Object's executeQuery() method. Once the ResultSet is received from the DBMS, the close() method is called to terminate the Statement.

Following snippet of code retrieves all the rows and columns from the Customers table.

```
Statement DataRequest;
ResultSet Results;
try {
    String query = "SELECT * FROM Customers";
    DataRequest = Database.CreateStatement();
    DataRequest = Db.createStatement();
    Results = DataRequest.executeQuery(query);
    DataRequest.close();
}
```

Process Data Returned by the DBMS

```
ResultSet  Results;
String  FirstName;
String  LastName;
String  printrow;
boolean  Records = Results.next();
if (!Records) {
    System.out.println("No data returned");
    return;
}
else
{
    do {
        FirstName = Results.getString(FirstName);
        LastName = Results.getString(LastName);
        printrow = FirstName + " " + LastName;
        System.out.println(printrow);
    }
    while (Results.next());
}
```

Terminate the Connection to the DBMS

The Connection to the DBMS is terminated by using the close() method of the Connection object once the J2EE Component is finished accessing the DBMS.

```
Db.close();
```

10a) Write a Java program to execute a database Transaction

Ans:- Following program illustrates how to process a transaction. The transaction in this example consists of two SQL statements, both of which update the street address of rows in the Customer table. Each SQL statement is executed separately and then the commit() method is called.

```
String url = "jdbc:odbc:CustomerInformation";
String userID = "jim";
String password = "keogh";
Statement DataRequest1, DataRequest2;
Connection Database;
try { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Database = DriverManager.getConnection(url, userID,
                                             password);
    }
catch (ClassNotFoundException error)
    { System.err.println("unable to load the JDBC/ODBC
      bridge" + error);
      System.exit(1);
    }
catch (SQLException error)
    { System.err.println("Cannot Connect to the
      database." + error);
      System.exit(2);
```

10b. Explain
(i) Callable Statement Object
(ii) Prepared Statement Object

Ans:- CallableStatement

This a CallableStatement Object is used to call a stored procedure from within a J2EE object. A stored procedure is a block of code and is identified by a unique name.

The CallableStatement object uses three types of parameters when calling a stored procedure. These parameters are IN, OUT and INOUT. The IN parameter contains any data that needs to be passed to the stored procedure and whose value is assigned using the setXxx() method

The OUT parameter contains the value returned by the stored procedure. INOUT parameter is a single parameter that is used to both pass information to the stored procedure and retrieve information from a stored procedure using the techniques.

Program Example

```
String url = "jdbc:odbc:CustomerInformation";
String userID = "jim";
String password = "keogh";
String lastOrderNumber;
```

```
Connection Db;
try{ Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
     Db = DriverManager.getConnection(url, UserID,
                                      password);
   }
catch(ClassNotFoundException error)
  { System.err.println("unable to load the
                        JDBC/ODBC bridge" +error);
    System.exit(1);
  }
catch (SQLException error) {
   System.err.println("cannot Connect to the
                       database " + error);
} System.exit(2);
try {
    String query= "{ CALL LastOrderNumber(?)}";
    CallableStatement csstatement = Db.prepareCall(query);
    csstatement.registerOutparameter(1, Types.VARCHAR);
    csstatement.execute();
    LastOrderNumber= csstatement.getString(1);
    csstatement.close();
  }
catch ( SQLException error)
  {
    System.err.println("SQL error." + error);
    System.exit(3);
  }
Db.close();
```

```
try{
    Database.setAutoCommit(false)
    String query1 = "UPDATE Customers SET
                    Street = '5 main Street' " +
                    "Where FirstName = 'Bob' ";

    String query2 = "UPDATE Customers SET
                    Street = '10 main Street' " +
                    "Where FirstName = 'Tim' ";
    DataRequest1 = Database.createStatement();
    DataRequest2 = Database.createStatement();
    DataRequest.executeUpdate(query1);
    DataRequest.executeUpdate(query2);

    Database.commit();
    DataRequest1.close();
    DataRequest2.close();
    Database.close();
}
catch(SQLException ex)
{System.err.println("SQLException: " +
                    ex.getMessage());
 if(con!=null){
    try{ System.err.println("Transaction is being rolledback");
         con.rollback();
    }
    catch(SQLException excep)
        { System.err.print("SQLException:");
          System.err.println(excep.getMessage());
    }
}
```

(h2) Savepoints

(ii) PreparedStatement Object

A SQL query can be precompiled and executed by using the preparedStatement Object. However, a question Mark is used as a placeholder for a value that is inserted into the query after the query is compiled. It is this value that changes each time the query is executed.

Program Example

```
String url = "jdbc:Odbc:CustomerInfolmation";
String userID = "jim";
String password = "keogh";
ResultSet Results;
Connection Db;
try { Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
      Db = DriverManager.getConnection(url, userID,
                                       password);
    }
catch ( classNotFoundException error)
  { System.err.println ("unable to load the
                         JDBC|ODBC bridge " +
                         error);
    System.exit(1);
  }
catch ( SQLException error)
  { System.err.println ("Cannot Connect to the
                         database " + error);
    System.exit(2);
  }
```

```
try{
    String query = "SELECT * FROM
                    customers WHERE
                    CustNumber = ?";
    PreparedStatement pstatement = Db.prepaledStatement(
                                    query);
    pstatement.setString(1,"123");
    Results = pstatement.executeQuery();
     // place code here to interact with the ResultSet
    pstatement.close();
}
catch ( SQLException error)
   {  System.err.println("SQL error " + error);
      System.exit(3);
   }
Db.close();
```