Set-2

Model Question Paper-2

Advanced Java and J2EE

Sem: 06 Subject code: 18CS644

Staff: Saleem. H

Q1a. Define with example for each of the following:
i) Autoboxing/unboxing ii) Type wrapper

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.

Autounboxing is the process by which the value of a boxed object is automatically extracted (unboxed) from a type wrapper when its value is needed. There is no need to call a method such as intValue() or doubleValue()

Example:

```
class Autobox
{ public static void main (String args[])
  { Integer iob = 100;
    int i = iob;
    System.out.println (i + " " + iob);
  }
}
```

ii) Type Wrapper

— Java provides type wrappers which are classes that encapsulate a primitive type within an object. The Type Wrappers are Double, Float, Long, Integer Short, Byte, Character and Boolean These classes offer a wide array of methods that allow you to fully Integrate the primitive types into Java's object hierarchy.

Example:

```
class Wrap
{ public static void main (String args[])
  { Integer iob= new Integer(100);
    int i = iob.intValue();
    System.out.println(i+" "+iob);
  }
}
```

Q1b:- Explain Single member annotation with a program Example

Ans:- A Single Member annotation contains only one member. It works like a normal annotation except that it allows a shorthand form of specifying the value of the member.

When only one member is present, you can simply specify the value for that member when the annotation is applied — You don't need to specify the name of the member.

Program Example

```
import java.lang.annotation.*;
import java.lang.reflect.*;

@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 (NoSuchMethodException exc)
      {System.out.println("Method Not
                          Found");
      }
    }
    public static void main(String args[])
    { myMeth();
    }
}
```

2.a. Define Enum. Write a program to display the enumeration constants using values() and ValueOf() method.

Ans:- Enumeration defines a class type.
An Enumeration is created using the enum keyword.
e.g:
enum Apple { Jon, Gold, RedDel, Winesap }

Program:

```
enum Apple { Jonathan, GoldenDel, RedDel,
             Winesap, Cortland
           }
class EnumDemo
{ public static void main (String args[])
  { Apple ap;
    Apple allapples[] = Apple.values();
    for (Apple a : allapples)
        System.out.println(a);
    ap = Apple.valueOf ("winesap");
    System.out.println ("ap contains " +ap);
} }
```

2b. How Annotations are obtained at runtime by use of reflection

Ans:- Although annotations are designed mostly for use by other development or deployment tools, if they specify a retention policy of RUNTIME, then they can be queried at run time by any Java program through the use of reflection. Reflection is the feature that enables information about a class to be obtained at run time.

The first step to using reflection is to obtain a class object that represents the class whose annotations you want to obtain. One of the easiest way is to call getClass() which is a method defined by Object. Its general form is shown below:

final Class getClass()

It returns the class object that represents the invoking object.

Class supplies the getMethod(), getField() and getConstructors() methods which obtain information about a method, field and Constructor respectively.

The getMethod has the following general form:

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

3a. Explain exceptions and methods that are provided in collections interface

Exceptions: ClassCastException: is generated when one object is incompatible with another, such as when an attempt is made to add an incompatible object to a collection.

NullPointerException is thrown if an attempt is made to store a null object and null elements are not allowed in the collection.

IllegalArgumentException is thrown if an invalid argument is used.

IllegalStateException is thrown if an argument attempt is made to add an element to a fixed length collection that is full.

Methods:

(1) boolean add(E obj)
(2) boolean addAll(Collection<? extends E> c)
(3) void clear()
(4) boolean contains(Object obj)
(5) boolean equals(Object obj)
(6) int hashCode()
(7) boolean isEmpty()
(8) Iterator<E> iterator()
(9) int size()

boolean add(E obj) - Adds obj to the invoking collection.

boolean addAll(Collection<? extends E> c) - Adds all the elements of c to the invoking collection

void clear() : Removes all elements from the invoking collection

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

int hashCode() — Returns the hash code for the invoking collection

boolean isEmpty() — Returns true if the invoking collection is empty

int size() — Returns the no. of elements held in the invoking collection

3b. List the recent changes made to collection in collection frameworks and explain

Ans:- (1) Generics
(2) autoboxing/unboxing
(3) for-each style for loop

Generics Fundamentally change the collections framework

All collections are now generic, and many of the methods that operate on collections take generic type parameters.

Generics add the one feature that collections had been missing : type safety.

One other point : To gain the advantages that generics bring collections older code will need to be rewritten.

Autoboxing facilitates the use of primitive Types :

Autoboxing/unboxing facilitates the storing of primitive types in collections. a collection can store only references, not primitive values. In the past, if you wanted to store a primitive value, such as in int, in a collection, you had to manually box it into its type wrapper. When the value was retrieved, it needed to be

manually unboxed (by using an explicit cast) into its proper primitive type.
Because of auto boxing / unboxing, Java can automatically perform the proper boxing and unboxing needed when storing or retrieving primitive types. There is no need to manually perform these operations.

The for-each style for loop

All collection classes in the Collections framework have been retrofitted to implement the Iterable interface, which means that a collection can be cycled through by use of the for-each style for loop. In the past, cycling through a collection required the use of an iterator, with the programmer manually constructing the loop. Although iterators are still needed for some uses, in many cases, iterator-based loops can be replaced by for loops

4a. Write a program to demonstrate the List interface using ArrayList class

Ans:-

```
import java.util.*;
class ArrayListDemo
{ public static void main(String args[])
{
ArrayList<String> al = new ArrayList<
                        String>();
System.out.println("Initial size of al:"
                    + al.size());
al.add("C");
al.add("A"); al.add("E");
al.add("B"); al.add("D");
al.add("F"); al.add(1, "A2");
System.out.println("size of al after
                    additions:" + al.size());
System.out.println("Contents of al: +al);
al.remove("F");
al.remove(2);
System.out.println("size of al after deletions:"
                    + al.size());
System.out.println("Contents of al: "+al);
}
}
```

4b. What are the exceptions and methods in Map interface

Exceptions :- ClassCastException when an object is incompatible with the elements in a map.

NullPointerException : is thrown if an attempt is made to use a null object and null is not allowed in the map.

UnsupportedOperationException is thrown when an attempt is made to change an unmodifiable map.

IllegalArgumentException is thrown if an invalid argument is used,

Methods defined by Map

void clear() — Removes all key/value pairs from the invoking map

boolean containsKey(Object k) — Returns true if the map contains k as ~~a class~~ key. Otherwise, returns false

boolean containsValue(Object v) — Returns true if the map contains v as a value. Otherwise, returns false.

boolean equals (Object obj) — Returns true if obj is a Map and contains the same entries. otherwise, returns false.

V get (Object k) — Returns the value associated with the key k.

int hashCode() — Returns the hashCode for the invoking map

5.a. Explain the different types of string constructors with example

Ans: ✓ Default constructor will create an instance of string with no characters in it

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

✓ 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);
```

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

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

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

String(String strObj)

5b. what are the methods to modify a String with example? Explain each

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.

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)

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

replace() : 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)

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

The second form of replace() replaces one character sequence with another. It has this general form:

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 this general form:

String trim(); Here is an example:

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

6.a Explain StringBuffer in detail

Ans:— StringBuffer represents growable and writable character sequence. StringBuffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed.

StringBuffer Constructors

StringBuffer defines these four constructors:

(1) StringBuffer()
(2) StringBuffer(int size)
(3) StringBuffer(String str)
(4) StringBuffer(CharSequence chars);

length() and capacity() : The current length of a StringBuffer can be found via the length() method, while the total allocated capacity can be found through the capacity() method. They have the following general forms:

```
int length()
int capacity()
```

ensureCapacity() : This is useful if you know in advance that you will be appending a large number of small strings to a StringBuffer. ensureCapacity() has this general form:

```
void ensureCapacity(int capacity)
```

SetLength() :- used to set the length of the buffer within a StringBuffer object

charAt() and setCharAt() : The value of a single character can be obtained from a StringBuffer via the charAt() method. The general forms are shown here:

```
char charAt(int where)
void setCharAt(int where, char ch)
```

insert() : The insert() method inserts one string into another. It is overloaded to accept values of all the simple types, plus strings, and CharSequence. These are a few of its forms:

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

6b. Define String Handling. Explain special String operations with example

Ans:- String handling deals with the various operations to be performed on strings

Special string operations

String Literals :- You can use a string literal to initialize a string object. e.g the following code fragment creates two equivalent strings :

```
char chars[] = { 'a', 'b', 'c' };
String s1 = new String(chars);
String s2 = "abc";
String System.out.println("abc".length());
```

String Concatenation - Concatnating two or more strings

```
string age = "9";
String s = "He is" + age + "years old";
System.out.println(s);
```

String Conversion and toString()

Every class implements toString() because it is defined by Object. For most important classes that you create, you will want to override toString() and provide your own string representations.

The toString() method has this general form:

String toString()

7a. Explain the life cycle of servlets

Ans:- Three methods are central to the life cycle of a servlet. These are init(), service(), and destroy().

init() method :- The server invokes the init() method of the servlet. This method is invoked only when the servlet is first loaded into memory

service() method : This method is called to process the HTTP request. The service() method is called for each HTTP request

destroy() method : The server calls the destroy() method to relinquish any resources such as file handles that are allocated for the servlet.

7b:- List and explain the core classes and interfaces that are provided 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 servlet to log events and access inform^n about their environment
(4) ServletRequest	— used to read data from a client request
(5) ServletRequest	— used to write data to a client request

The core classes are:

(1) GenericServlet — Implements the Servlet and ServletConfig interfaces

(2) ServletInputStream — provides an i/p stream for reading requests from a client

(3) ServletOutputStream — provides an o/p stream for writing responses to a client

(4) ServletException — Indicates a servlet error occurred

(5) UnavailableException — Indicates a servlet is unavailable

Note: For detail, Refer Q.8b from Set-I

8a: What is Jsp tag? Explain the different types of Jsp tags

Ans:- Java server page is a server side program that is similar in design and functionality to a Java servlet. A JSP is written in HTML, XML or in the client's format that is interspersed with scripting elements, directives and actions

(1) comment tag (2) declaration statement tag (3) Directive tags (4) Expression tags (5) scriptlet tags

Note:- For detail, Refer Q.8a from set-1

Q8b. What is Cookie? Explain the working of cookie in java with code Snippets

Ans:- A cookie is stored on a client and contains state information. Cookies are valuable for tracking user activities.

Three files are taken into account.

AddCookie.htm — Allows a user to specify a value for the cookie named MyCookie

AddCookieServlet.java — processes the submission of AddCookie.htm

GetCookiesServlet.java — Displays cookie values

now, consider AddCookie.htm

```
<html><body><form name="Form1"
                  method="post"
                  action="http://localhost:8080/
                          servlet-Examples/servlet/
                          AddCookieServlet">
Enter a value for MyCookie:
<input type=textbox
       name="data"
       size=25 value="">
<input type=submit
       value="Submit">
</form>
</body>
</html>
```

AddCookieServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AddCookieServlet extends HttpServlet
{
    public void doPost(HttpServletRequest req,
        HttpServletResponse res) throws
        ServletException, IOException)
    {
        String data = req.getParameter("data");
        Cookie cookie = new Cookie("MyCookie",
                                   data);
        response.addCookie(cookie);
        response.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.println("MyCookie has been set to");
        pw.println(data);
        pw.close();
    }
}
```

GetCookiesServlet.java

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

public class GetCookiesServlet extends
    HttpServlet
{
    public void doGet(HttpServletRequest req,
        HttpServletResponse res) throws
        ServletException, IOException
```

```
Cookie[] cookies = req.getCookies();
res.setContentType("text/html");
PrintWriter pw = res.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();
}
```

9a. Write a program to execute a database transaction.

Ans:- Ref Q10a from set-I

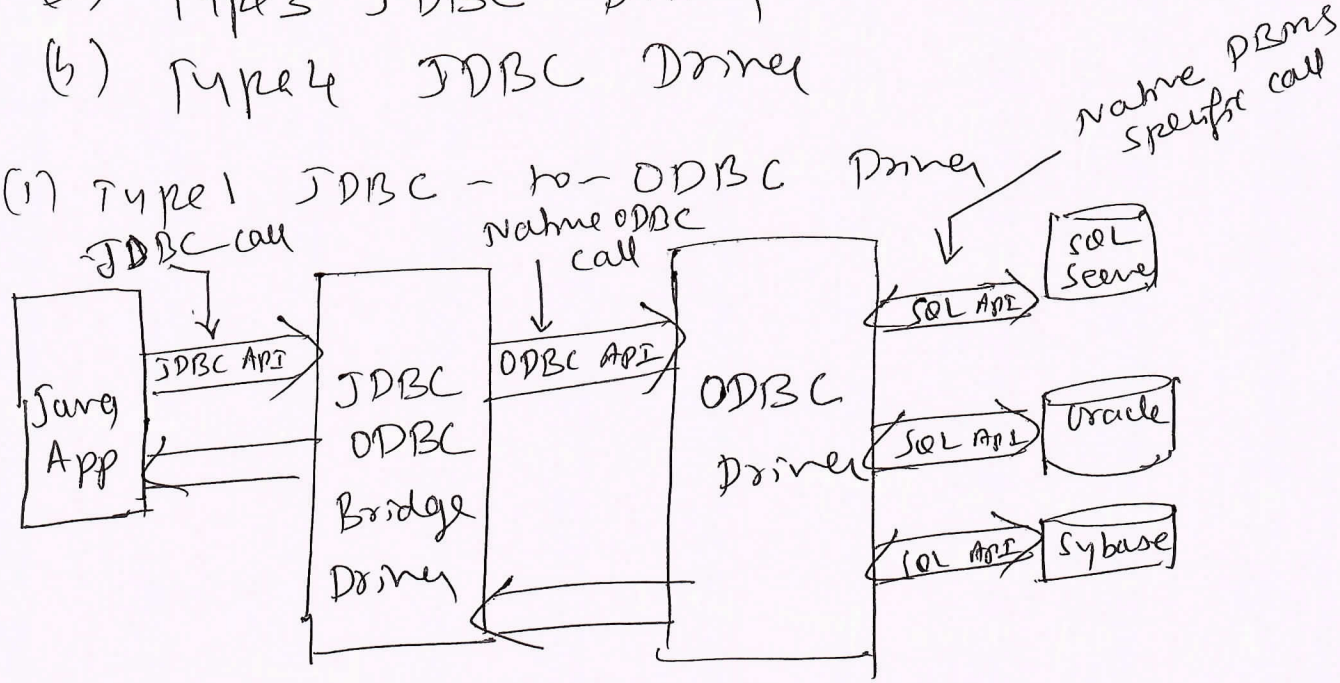
```
String url="jdbc:odbc:customerInformation";
String userID ="jim";
String pw = "Bush";
Statement DR1, DR2; Connection con;
try{ class.forname("sun.jdbc.odbc.JdbcOdbcDriver");
     con. DrivesManager.getConnection(url, userID, pw);
}
catch(ClassNotFoundException error)
  { System.err.println("unable to load JDBC/ODBC
                       bridge "+error);
    System.exit(1);
  }
catch(SQLException error)
  { System.err.println("Cannot Connect to the
                       database"+error);
    System.exit(2);
  }
```

9b. Explain JDBC multitier architecture with neat diagram i.e driver types? Detail

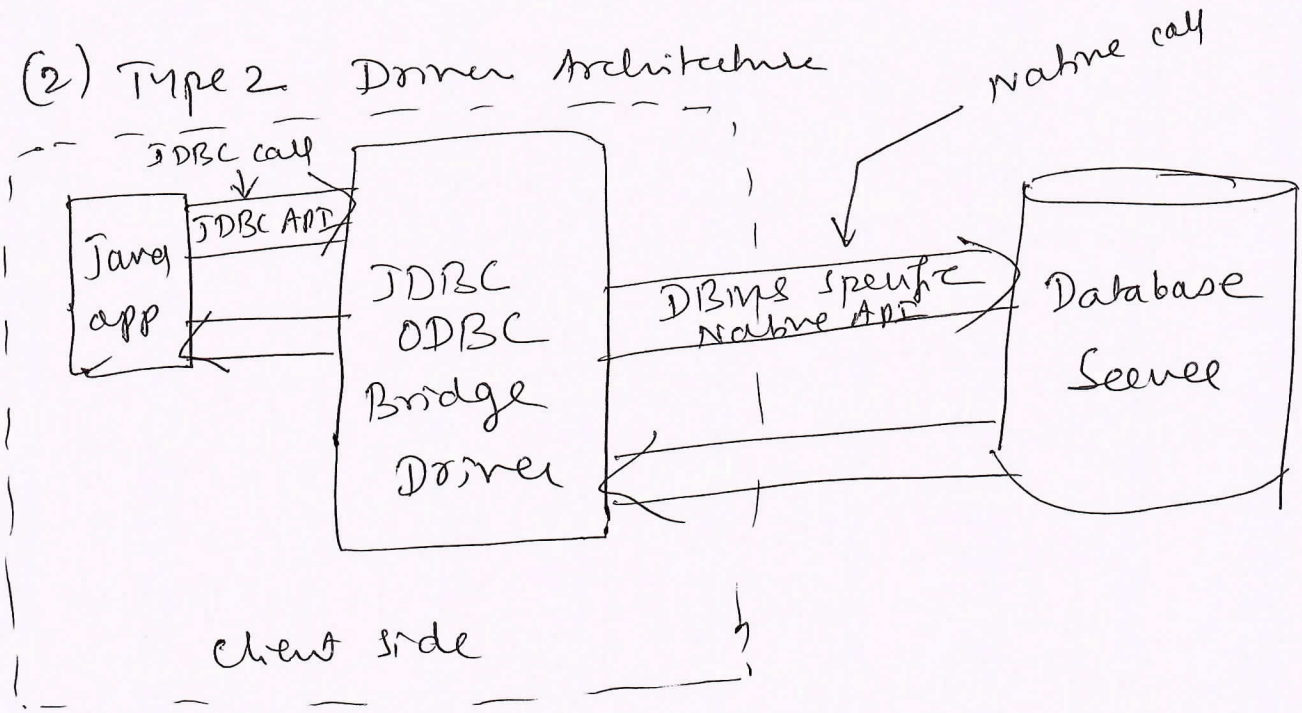
Ans:- Ref Q 9a from set-1

(1) Type 1 JDBC - to - ODBC Driver
(2) Type 2 Java / native code Driver
(3) Type 3 JDBC Driver
(4) Type 4 JDBC Driver

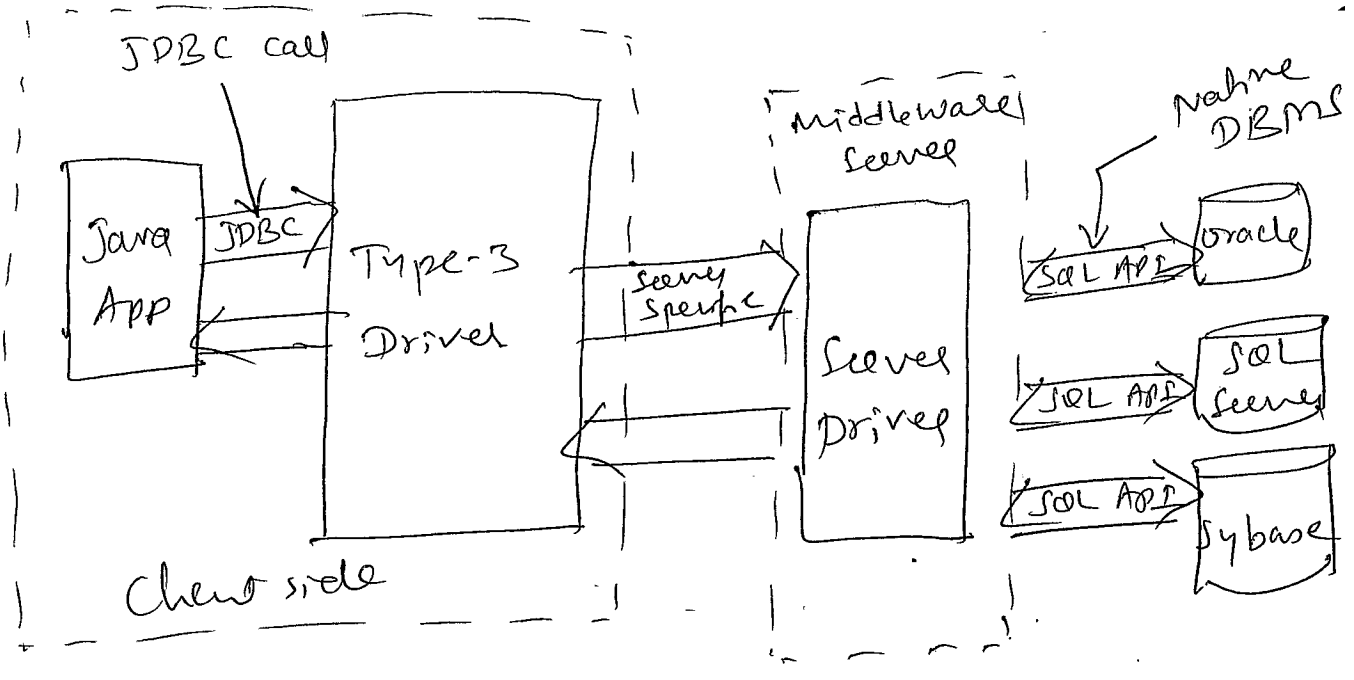
(1) Type 1 JDBC - to - ODBC Driver



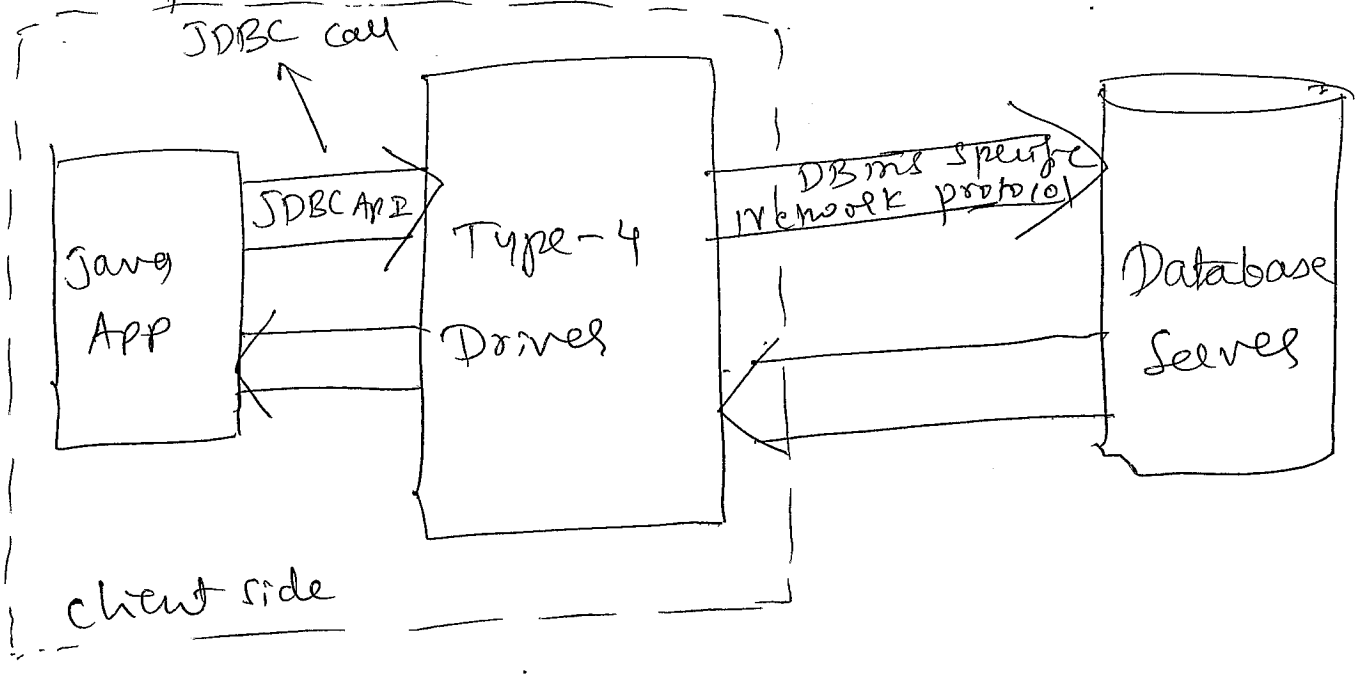
(2) Type 2 Driver Architecture



(3) Type 3 JDBC Driver Architecture



(4) Type 4 Driver Architecture



10a. Describe the various steps of JDBC process with code snippets

Ans: (1) Loading the JDBC Driver
(2) Connecting to the DBMS
(3) Creating and executing a statement
(4) processing data returned by the DBMS
(5) Terminating the connection with the DBMS

Ref. Q. 9b from set-1

10b. Write a note on Database metadata Object methods and Resultset metadata Object methods.

DatabaseMetaData object methods:-

- getDatabaseProductName() - Returns the product name of the database
- getUserName() - Returns the user name
- getURL() - Returns the URL of the database
- getSchemas() - Returns all the schema names available in this database
- getPrimaryKeys() Returns primary key

Resultset metadata methods

- getColumnCount()
- getColumnName(int number)
- getColumnType(int number)

Prepared by:
Saleem. H

H.O.D

Dean Academic