Internal Assessment I – Sept 2019

Scheme and Solutions

Sub:	Advanced Java & J2EE					Sub Code:	17CS553	Branch:	ISE	
Date:	07-9-2019	Duration:	90 min's	Max Marks:	50	Sem / Sec:	V			OBE

Q. 1 a) What is string in Java? Write a java program that demonstrates any four constructors of string class.

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

The String class supports several constructors.

a. To create an empty String
   the default constructor is used.

Ex: String s = new String();
will create an instance of String with no characters in it.

b. To create a String initialized by an array of characters, use the constructor shown here:

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

   This constructor initializes s with the string “abc”.

c. To specify a subrange of a character array as an initializer using the following constructor:
   String(char chars[ ], int startIndex, int numChars)

   Here, startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use. Here is an example:

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

   This initializes s with the characters cde.

d. To 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.
```

```
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);
}
}
```

The output from this program is as follows:

```
Java
Java
```

As you can see, s1 and s2 contain the same string.

e. To Construct string using byte array:

Even though Java's char type uses 16 bits to represent the basic Unicode character set, the typical format for strings on the Internet uses arrays of 8-bit bytes constructed from the ASCII character set.

Because 8-bit ASCII strings are common, the String class provides constructors that initialize a string when given a byte array.

Ex: String(byte asciiChars[ ])
String(byte asciiChars[ ], int startIndex, int numChars)

The following program illustrates these constructors:

```
class SubStringCons
{ public static void main(String args[])
{
byte ascii[] = {65, 66, 67, 68, 69, 70 };
String s1 = new String(ascii);
System.out.println(s1);
String s2 = new String(ascii, 2, 3);
System.out.println(s2);
}
}
```

This program generates the following output:

```
ABCDEF
CDE
```


f. To construct a String from a StringBuffer by using the constructor shown here:

Ex: String(StringBuffer strBufObj)

g. Constructing string using Unicode character set and is shown here:

String(int codePoints[ ], int startIndex, int numChars)

codePoints is an array that contains Unicode code points.

Q. 2 a) Explain how to modify the string using following methods

i) substring( ) ii) concat( ) iii) replace( ) iv) trim( )

i) substring( )- A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.

```
public class TestSubstring{

 public static void main(String args[]){

 String s="SachinTendulkar";

  System.out.println(s.substring(6));//Tendulkar

  System.out.println(s.substring(0,6));//Sachin

 }

}
```

Output:

```
Tendulkar
Sachin
```

ii) concat( ) - string concatenation forms a new string *that is* the combination of multiple strings.

```
 class TestStringConcatenation3{

  public static void main(String args[]){

  String s1="Sachin ";

  String s2="Tendulkar";

  String s3=s1.concat(s2);

  System.out.println(s3);//Sachin Tendulkar

 }

}
```

Output: Sachin Tendulkar

iii)replace( )

The **java string replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

```
public class ReplaceExample1{

public static void main(String args[]){

String s1="javatpoint is a very good website";

String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'

System.out.println(replaceString);

}

}
```

Output: jevetpoint is e very good website

iv) trim( )
The **java string trim()** method eliminates leading and trailing spaces.

```
public class StringTrimExample{

public static void main(String args[]){

String s1="  hello string   ";

System.out.println(s1+"javatpoint");//without trim()

System.out.println(s1.trim()+"javatpoint");//with trim()

}

}
```

Output:
hello string   javatpoint
hello stringjavatpoint

Q.3 a) Explain the following StringBuffer methods with an example
i) insert( ) ii) append( ) iii)delete ( ) iv) reverse v) capacity

i) insert() method inserts the given string with this string at the given position.

```
class StringBufferExample2{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}
```

ii) append( )

The append() method concatenates the given argument with this string.

```
class StringBufferExample{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}
```

iii) delete ( )

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class StringBufferExample4{

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

```
StringBuffer sb=new StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}
```

iv) reverse( )

The reverse() method of StringBuilder class reverses the current string.

```
class StringBufferExample5{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}
```

v)capacity( )

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
class StringBufferExample6{

public static void main(String args[]){

StringBuffer sb=new StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16
```

```
sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2

}

}
```

Q. 4 a) Differentiate between equals and = = with respect to string comparison

equals( ) Versus ==

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

```
// equals() vs ==
class EqualsNotEqualTo {


   public static void main(String args[]) {
     String s1 = "Hello";
     String s2 = new String(s1);

     System.out.println(s1 + " equals " + s2 + " -> " +
                        s1.equals(s2));
     System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));
   }
}
```

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

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

Q. 4 b) Write a program to remove duplicate characters in a given string and display new string without any duplicates

```
public static void main(String[] args) {
        String stringWithDuplicates = "abcdafbd";
        char[] characters = stringWithDuplicates.toCharArray();
        boolean[] found = new boolean[256];
        StringBuilder sb = new StringBuilder();
        System.out.println("String with duplicates : " + stringWithDuplicates);
        for (char c : characters) {
```

```
                if (!found[c]) {
                    found[c] = true;
                    sb.append(c);
                }
            }
            System.out.println("String after duplicates removed : " + sb.toString());
    }
```

Output:

String with duplicates : abcdafbd

String after duplicates removed : abcdf

Q. 5 a) What are database drivers? Explain different JDBC driver types

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

Type 1 driver JDBC to ODBC Driver

1. It is also called JDBC/ODBC Bridge , developed by MicroSoft.
2. It receives messages from a J2EE component that conforms to the JDBC specifications
3. Then it translates into the messages understood by the DBMS.
4. This is DBMS independent database program that is ODBC open database connectivity.

Type 2 JAVA / Native Code Driver

1. Generates platform specific code that is code understood by platform specific code only understood by specific databases.
2. Manufacturer of DBMS provides both java/ Native code driver.
3. Using this provides lost of portability of code.
4. It won't work for another DBMS manufacturer

Type 3 JDBC Driver

1. Most commonly used JDBC driver.
2. Coverts SQL queries into JDBC Formatted statements.
3. Then JDBC Formatted statements are translated into the format required by the DBMS.
4. Referred as Java protocol

Type 4 JDBC Driver

1. Referred as Type 4 database protocol
2. SQL statements are transferred into the format required by the DBMS.
3. This is the fastest communication protocol.

Q. 6 a) Describe various steps of JDBC with code snippets

1. Loading the JDBC driver

   - The jdbc driver must be loaded before the J2EE compnet can be connected to the database.
   - Driver is loaded by calling the method and passing it the name of driver

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

2. Connecting to the DBMS.

   - Once the driver is loaded , J2EE component must connect to the DBMS using DriverManager.getConnection() method.
   - It is highest class in hierarchy and is responsible for managing driver information.

- It takes three arguments URL, User, Password
- It returns connection interface that is used through out the process to reference a database

```
String url="jdbc:odbc:JdbcOdbcDriver";
String userId="jim"
String password="Keogh";
Statement DatRequest;
Private Connection db;

try{
Class.forName("sun:jdbc.odbc.JdbcOdbcDriver");
Db=DriverManager.getConnection(url,userId,password);
}
```

3. Creating and Executing a statement.
    - The next step after the JDBC is loaded and connection is successfully made with a particular database managed by the dbms, is to end a particular query to the DBMS for processing.
    - SQL query consists series of SQL command that direct DBMS to do something example Return rows.
    - Connect.createStatement() method is used to create a statement Object.
    - The statement object is then used to execute a query and return result object that contain response from the DBMS

```
Statement DataRequest;
ResultSet Results;
try {
        String query="select * from Customers";
        DataRequest=Database.createStatement();
        Results= DataRequests.executeQuery(query);
  }
```

4. Processing data returned by the DBMS
    - **java.sql.ResultSet** object is assigned the result received from the DBMS after the query is processed.
    - **java.sql.ResultSet** contain method to interct with data that is returned by the DBMS to the J2EE Component.

```
Results= DataRequests.executeQuery(query);
do
{
Fname=Results.getString(Fname)
}
While(Results.next())
```

In the above code it return result from the query and executes the query. And getString is used to process the String retrived from the database.

5. Terminating the connection with the DBMS.
   To terminate the connection Database.close() method is used.

Q. 7 a) Explain with example
i) Callable statement object ii) Prepared statement object

i)callable object

1. The callableStatement object is used to call a stored procedure from with in J2EE object. A stored procedure is block of code and is identified by unique name. the code can bewritten in Transact-C ,PL/SQL.

2. Stored procedure is executed by invoking by the name of procedure.

3. The callableStatement uses three types of parameter when calling stored procedure. The parameters are IN ,OUT,INOUT.

4. IN parameter contains data that needs to be passed to the stored procedure whose value is assigned using setxxx() method.

5. OUT parameter contains value returned by stored procedure.the OUT parameter should be registers by using registerOutParameter() method and then later retrieved by the J2EE component using getxxx() method.

6. INOUT parameter is used to both pass information to the stored procedure and retrieve the information from the procedure.

7. Suppose, you need to execute the following Oracle stored procedure:

```
CREATE OR REPLACE PROCEDURE getEmpName
        (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
 SELECT first INTO EMP_FIRST    FROM Employees   WHERE ID = EMP_ID;
END;
```

8. The following code snippets is used

```
CallableStatement cstmt = null;
try {     String SQL = "{call getEmpName (?, ?)}";
            cstmt = conn.prepareCall (SQL);catch (SQLException e) { }
```

9. Using CallableStatement objects is much like using PreparedStatement objects. You must bind values to all parameters before executing the statement, or you will receive an SQLException.
10. If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.
11. When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type to the data type the stored procedure is expected to return.
12. Once you call your stored procedure, you retrieve the value from the OUT parameter with the appropriate getXXX() method. This method casts the retrieved value of SQL type to a Java data type.

ii) Preparedstatement:

3. Setxxx() is used to replace the question mark with the value passed to the setxxx() method . xxx represents data type of the field.
   Example if it is string then setString() is used.
4. It takes two arguments on is position of question mark and other is value to the filed.
5. This is referred as late binding.

```
String url="jdbc:odbc:JdbcOdbcDriver";
String userId="jim"
String password="Keogh";
ResultSet rs;

// code to load driver
//code to connect to the database
 try{

String query="SELECT * FROM Customers where cno=?";
PreparedStatement   pstatement=db.preparedStatement(query);
pstatement.setString( 1,"123"); // 1 represents first place holder, 123 is value
rs= pstatement.executeQuery();


}catch(SQLException  err)
{
System.err.println("Error");
System.exit(1);
}
```

1. A SQL query must be compiled before DBMS processes the query. Query is precompiled and executed using Prepared statements.
2. Question mark is placed as the value of the customer number. The value will be inserted into the precompiled query later in the code.

Q 7 a) Explain the following character extraction methods chatAt( ) and tocharArray( )

There are several ways by which characters can be extracted from String class object. String is treated as an object in Java so we can't directly access the characters that comprise a string. For doing this String class provides various predefined methods

charAt()

charAt() method is used to extract a single character at an index. It has following syntax.

Syntax

char charAt(int index)

```
class temp
{
public static void main(String...s)
{
String str="Hello";
char ch=str.charAt(2);
System.out.println(ch);
}
}
```

Output

l

toCharArray()

It is an alternative of getChars() method. toCharArray() convert all the characters in a String object into an array of characters. It is the best and easiest way to convert string to character array. It has following syntax.

Syntax

char [] toCharArray()

```
class temp
{
public static void main(String...s)
{
String str="Hello World";
char ch[]=str.toCharArray();
System.out.println(ch);
}
}
```

Output

Hello World

Q. 8 b) Explain the following methods with suitable example

i)startsWith( ) & endsWith( ) ii) compareTo( ) iii) indexOf( )

i) startsWith( ) & endsWith

startsWith( ) and endsWith( )

String defines two routines that are, more or less, specialized forms of **regionMatches( )**. The **startsWith( )** method determines whether a given **String** begins with a specified string. Conversely, **endsWith( )** determines whether the **String** in question ends with a specified string. They have the following general forms:

boolean startsWith(String *str*)
boolean endsWith(String *str*)

Here, *str* is the **String** being tested. If the string matches, **true** is returned. Otherwise, **false** is returned. For example,

```
"Foobar".endsWith("bar")
```

and

```
"Foobar".startsWith("Foo")
```

are both **true**.

A second form of **startsWith( )**, shown here, lets you specify a starting point:

boolean startsWith(String *str*, int *startIndex*)

Here, *startIndex* specifies the index into the invoking string at which point the search will begin. For example,

```
"Foobar".startsWith("bar", 3)
```

returns **true**.

ii) compareTo( )

We have following two ways to use compareTo() method:

int compareTo(String str)

Here the comparison is between string literals. For example string1.compareTo(string2) where string1 and string2 are String literals.

int compareTo(Object obj)

Here the comparison is between a string and an object. For example string1.compareTo("Just a String object") where string1 is a literal and it's value is compared with the string specified in the method argument.

```
public class CompareToExample {
  public static void main(String args[]) {
     String str1 = "String method tutorial";
     String str2 = "compareTo method example";
     String str3 = "String method tutorial";

     int var1 = str1.compareTo( str2 );
     System.out.println("str1 & str2 comparison: "+var1);

     int var2 = str1.compareTo( str3 );
     System.out.println("str1 & str3 comparison: "+var2);

     int var3 = str2.compareTo("compareTo method example");
     System.out.println("str2 & string argument comparison: "+var3);
  }
}
```

Output:

```
str1 & str2 comparison: -16
str1 & str3 comparison: 0
str2 & string argument comparison: 0
```

iii) indexOf( )

The **java string indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

```
public class IndexOfExample3 {

    public static void main(String[] args) {

        String s1 = "This is indexOf method";

        // Passing substring and index

        int index = s1.indexOf("method", 10); //Returns the index of this substring

        System.out.println("index of substring "+index);

        index = s1.indexOf("method", 20); // It returns -1 if substring does not found

        System.out.println("index of substring "+index);
```

```
        }

    }
```

Output: index of substring 16
index of substring -1