Solutions

Internal Assessment Test 2 – June 2021

Sub:	Advanced Java & J2EE							Code:	18CS644
Date:	23.06.2021	Duration:	60mins	Max Marks:	50	Sem:	VI	Branch:	CSE

Note: Answer Any 3 Questions from Part A and 2 questions from Part B

		Question	Max Marks
1	a	Write short notes on the Set interface and the HashSet class. **Interface Set<E>** The Set interface extends collection. It implements the behaviour of Set, i.e. to not allow duplicate elements. The behavior of the add() method is as follows : **boolean add(E e)** Adds the specified element to this set if it is not already present If this set already contains the element, the call leaves the set unchanged and returns false. **class HashSet<E>** HashSet class extends AbstractSet and implements the Set interface. It creates a collection that uses **HashTable** for storage. Hash table stores information by using a mechanism called hashing. A key is used to determine a unique value, called its hash code. Hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically. The code can't directly index the hash table. Advantage of hashing: allows the execution time of add( ) contains( ), remove( ), and size( ) to remain constant even for large sets. HashSet can be created using the following **constructors**. In order to improve the performance, HashSet can be constructed with a fill ratio.The fill ratio is between 0.0 and 1.0. It determines how full the hash set can be before it is resized upward. When the number of elements is greater than the capacity of the hash set multiplied by its fill ratio, the hash set is expanded. For constructors that do not take a fill ratio, 0.75 is used. HashSet( ) :constructs a default hash set HashSet(Collection<? extends E> c): initializes the hash set by using the elements of c HashSet(int capacity) initializes the capacity of the hash set to capacity, default: 16 HashSet(int capacity, float fillRatio) : initializes both the capacity	8

	b	Demonstrate HashSet with a suitable example.
		`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("Beta"); hs.add("Alpha");`
		`hs.add("Eta"); hs.add("Gamma");`
		`hs.add("Epsilon"); hs.add("Omega");`
		`System.out.println(hs);`
		`//contains`
			4

```
        System.out.println("hs contains 'Alpha'?"+hs.contains("Alpha"));
            //removed items
        h.remove("Gamma"); System.out.println(h);
            }
}
```

Output:

```
[Gamma, Eta, Alpha, Epsilon, Omega, Beta]
Hs contains 'Alpha'? True
[Eta, Alpha, Epsilon, Omega, Beta]
```

2 What are legacy classes? Explain in detail the 4 legacy classes with code. 12

In J2SE1.2 several of the original classes were reengineered to support the collection interfaces. None of the modern collection classes are synchronized. All legacy classes are synchronized

There are 5 Legacy classes :

Vector: a dynamic array and is similar to ArrayList. It is reengineered to extend AbstractList and to implement the List interface.

Stack : subclass of Vector and implements LIFO order

Dictionary: An abstract class that represents a key/value storage repository and operates like Map. It is fully superseded by Map

HashTable: a concrete implementation of a Dictionary. It stores key/value pairs in a hash table. Neither keys nor values can be null. You can specify an object that is used as a key, the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index

Properties: subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String. It can be easily stored to or loaded from disk with the store( ) and load( ) methods.

Vector

A vector can be constructed by any of the following constructors.

Vector( ) : initial size of 10.
Vector(int *size*) :
Vector(int *size*, int *incr*)
incr specifies the number of elements to allocate each time that a vector is resized upward
By default it is ***doubled***.
Vector(Collection<? extends E> *c*)

A vector has protected data members

int capacityIncrement; **// stores the increment** value
int elementCount; **// number of elements currently in the vector**
Object[ ] elementData; **//array that holds the vector**

The legacy methods of a vector are as follows.

void addElement(E el) – add element
E elementAt(int idx) – add at an index position
E firstElement( )
E lastElement()
int indexOf(Object o) and int **lastIndexOf(Object o)**.
boolean removeElement(Object o) or void **removeElementAt(int idx)**.

```
// Demonstrate various Vector operations.
import java.util.*;
class VectorDemo {
        public static void main(String args[]) {
        // initial size is 3, increment is 2
        Vector<Integer> v = new Vector<Integer>(3, 2);
        System.out.println("Initial size: " + v.size());
        System.out.println("Initial capacity: " +
        v.capacity());
        v.addElement(1); v.addElement(2);
        v.addElement(3); v.addElement(4);
        System.out.println("Capacity after four additions: " + v.capacity());
```

```
            System.out.println("First element: " + v.firstElement());
            System.out.println("Last element: " + v.lastElement());
            if(v.contains(3))
                    System.out.println("Vector contains 3.");
            // Enumerate the elements in the vector.
            Enumeration<Integer> vEnum = v.elements();
            System.out.println("\nElements in vector:");
            while(vEnum.hasMoreElements())
                    System.out.print(vEnum.nextElement() + " ");

            // recommended to use for loop
            // System.out.println("\nElements in vector:");
            //for(int i : v)
                    //System.out.print(i + " ");
            System.out.println();
            }
}
```

Output:

```
Initial size: 0
Initial capacity: 3
Capacity after four additions: 5
First element: 1
Last element: 12
Vector contains 3.
Elements in vector:
1 2 3 4
```

Stack

E push(E element)
E pop( ).
E peek( ).
EmptyStackException is thrown if you call **pop( )** or **peek( )** when the invoking stack is empty.
boolean empty( ) :returns **true** if nothing is on the stack.
search( ) :determines whether an object exists on the stack. Returns the number of pops that are required to bring it to the top of the stack

```
        public class stackDemo {
                    public static void main(String args[])) {
                          Stack<Integer> st = new Stack<Integer>();
                          System.out.println("stack: " + st);
                          st.push(42); st.push(66); st.push(99);
                           System.out.println("After pushing 3 items: " + st);
                          st.pop();st.pop();
                        System.out.println("After popping two items: " + st);
         }
}
run:
stack: []
After pushing 3 items: [42, 66, 99]
After popping two items: [42]
```

HashTable

V put(K key, V value)
V get(Object key)
keys( ) and **elements( ) :**
The keys and values can each be returned as an
Enumeration elements()
Enumeration<K> keys( )
size( ) : returns the number of key/value pairs stored in a dictionary
isEmpty( ) returns **true** when the dictionary is empty.
remove(Object key) method to delete a key/value pair.
Constructors:

Hashtable( ) : default size is 11
Hashtable(int *size*)
Hashtable(int *size*, float *fillRatio*): *fillRatio* determines how full the hash table can be before it is resized upward. 0.75 is used by default
Hashtable(Map<? extends K, ? extends V> *m*) : hash table that is initialized with the elements in *m*
- A hash table can only store objects that override the **hashCode( ) and equals( )**
- *If key is String, it already has these.*
- *does not directly support iterators.*

```
import java.util.Enumeration;
import java.util.Hashtable;

public class hashTableDemo {

   hashTableDemo() {
      Hashtable<String, Double> balance =
new Hashtable<String, Double>();
      Enumeration<String> names;
      String str;
      double bal;
      balance.put("John Doe", 3434.34);        balance.put("Tom Smith", 123.22);
      balance.put("Jane Baker", 1378.00);       balance.put("Tod Hall", 99.22);
      balance.put("Ralph Smith", -19.08);
      // Show all balances in hashtable.
      names = balance.keys();
      while(names.hasMoreElements()) {
         str = names.nextElement();
         System.out.println(str + ": " +
         balance.get(str));
      }
     System.out.println();
      // Deposit 1,000 into John Doe's account.
      bal = balance.get("John Doe");
      balance.put("John Doe", bal+1000);
      System.out.println("John Doe's new balance: " +
      balance.get("John Doe"));

      }
}
```

Output:

```
Output:
Tod Hall: 99.22
Ralph Smith: -19.08
John Doe: 3434.34
Jane Baker: 1378.0
Tom Smith: 123.22

John Doe's new balance: 4434.34
```

Properties
Properties( ) - has no default values
Properties(Properties *propDefault)*
Important Use :
specify **a default property** that will be returned if no value is associated with a certain key
Both cases above : the property list is empty
deprecated method: **save( )**.
This was replaced by **store( )**
because **save( )** did not handle errors correctly.
getProperty( ) : a default value can be specified along with the key.
getProperty("name" ,"default value")
If the "name" value is not found, then "default value" is returned

```
import java.util.Properties;
import java.util.Set;
public class PropertiesDemo {
   PropertiesDemo() {
      Properties capitals = new Properties();
      capitals.put("Illinois", "Springfield");
      capitals.put("Missouri", "Jefferson City"); capitals.put("Washington", "Olympia");
      capitals.put("California", "Sacramento");capitals.put("Indiana", "Indianapolis");
      // Get a set-view of the keys.
      Set<?> states = capitals.keySet();
      // Show all of the states and capitals.
      for(Object name : states)
      System.out.println("The capital of " +   name + " is " +  capitals.getProperty((String)name)   + ".");
      System.out.println();
      // Look for state not in list -- specify default.
      String str = capitals.getProperty("Florida", "Not Found");
      System.out.println("The capital of Florida is " + str + ".");
   }
}
```

Output:

```
The capital of Missouri is Jefferson City.
The capital of Illinois is Springfield.
The capital of Indiana is Indianapolis.
The capital of California is Sacramento.
The capital of Washington is Olympia.

The capital of Florida is Not Found.
```

3	a	Differentiate equals and == when used on strings with a suitable example.	6

equals( ) method compares the characters inside a String object.
boolean equals(Object str)
str is the String object being compared with the invoking String object
true if the strings contain the same characters in the same order
false otherwise

== operator compares two object references to see whether they refer to the same instance
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 ==

```
System.out.println("\n** equals() vs == **");
String str1 = "Java";
String str2 = new String(str1);

System.out.println(str1+" equals "+str2+ "->"+str1.equals(str2));
System.out.println(str1+" == "+str2+ "->"+str1==str2);
```

Output:

```
** equals() vs == **
Java equals Java->true
false
```

	b	Explain the following string methods : regionMatches(), split(), trim()	6

regionMatches()
boolean regionMatches(int startIndex, String str2,
int str2StartIndex, int numChars)
compares a specific region inside a string with another specific region in another string. The startIndex starting index of invoking String object.

str2 – string being compared
str2StartIndex – starting index for comparison of str2.
numChars - length of the substring being compared.
Overloaded version
boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars)
if ignoreCase is true, the case of the characters is ignored

```
System.out.println("\n** Region Matches **");
  s1 = "Java Programming";
  s2 = "programming";
System.out.println(s1 + " equals " + s2 + " for region 5 to 10 -> " + s1.regionMatches(true, 5, s2,0,3));
 //Specialized form  of region Matches
System.out.println("\n** Specialized form  of region Matches == **");
System.out.println("String 'Arrival' starts with 'Ar'->"+ "Arrival".startsWith("Ar"));
System.out.println("String 'ended' ends with 'ed'->"+ "ended".endsWith("ed"));
```

Output:
** Region Matches **
Java Programming equals programming for region 5 to 10 -> true
** Specialized form of region Matches == **
String 'Arrival' starts with 'Ar'->true
String 'ended' ends with 'ed'->true
String 'Arrival' has 'rival' starting at index 2->true

split()
String[ ] split(String regExp)
Decomposes the invoking string into parts and returns an array that contains the result.
Each part is delimited by the regular expression passed in regExp.
String[ ] split(String regExp, int max)

Decomposes the invoking string into parts and returns an array that contains the result.
Each part is delimited by the regular expression passed in regExp.

The number of pieces is specified by max.
If max is negative, then the invoking string is fully decomposed.
Otherwise, if max contains a nonzero value, the last entry in the returned array contains the remainder of the invoking string.
If max is zero, the invoking string is fully decomposed, but no trailing empty strings will be included.

```
String s = "Programming in Java during Java class";
System.out.println(Arrays.toString(s.split("J\\w*",1)));
System.out.println(Arrays.toString(s.split("J\\w*",2)));
System.out.println(Arrays.toString(s.split("J\\w*",3)));

String st = "A,B,C,E,";
System.out.println(Arrays.toString(st.split(",",0)));
System.out.println(Arrays.toString(st.split(",",-1)));
```

Output:

```
[Programming in Java during Java class]
[Programming in ,  during Java class]
[Programming in ,  during ,  class]
[A, B, C, E]
[A, B, C, E, ]
```

String trim( )
returns a copy of the invoking string from which any leading and trailing whitespace has been removed.

```
import java.util.*;
```

```
public class Main{
        public static void main(String[] args) {
                HashMap<String, Integer> marks = new HashMap<String, Integer>();
                marks.put("1CR14CS001", 85);marks.put("1CR14CS003", 86);
                marks.put("1CR14CS045",75); marks.put("1CR14CS085",40);
                marks.put("1CR14CS096", 87);

                 Scanner in = new Scanner(System.in);
                System.out.print("Enter USN: ");
                String s = in.next();
                s= s.trim();
                System.out.println(marks.get(s));
        }
}
```

Output:
Enter USN: 1CR14CS001
85
Enter USN:1CR14CS003 (enter)
86
Enter USN: 1CR14CS085 (enter)
40

4 a Explain the construction of String with all possible constructors. Use code snippets. 6

String s = new String();
will create an instance of String with no characters in it
String(char chars[ ])
Creates a String initialized by an array of characters
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
String(String strObj)
construct a String object that contains the same character sequence as another String object.
String s1 = new String(c);
String s2 = new String(s1);
s1 and s2 contain the same string

String(char chars[ ], int startIndex, int numChars)
specify a subrange of a character array as an initializer
startIndex specifies the index at which the subrange begins
numChars specifies the number of characters to use

char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
String s = new String(chars, 2, 3); //cde
String(StringBuffer strBufObj)
- construct a String from a StringBuffer

String(StringBuilder strBuilderObj)
- construct a String from a StringBuilder

String(int codePoints[ ], int startIndex, int numChars)
- extended Unicode character set
- codePoints is an array that contains Unicode code points

There are also constructors that let you specify Charset
String(byte chrs[ ])
- chrs specifies the array of bytes

String(byte chrs[ ], int startIndex, int numChars)
- specify a subrange In each of these constructors
- byte-to-character conversion is done by using the default character encoding

byte a[]={65,66,67,68,69,70};
String s = new String(a);

System.out.println(s):

String s = new String(a,1,3);
System.out.println(s):
ABCDEF
BCD

b

Compare and contrast Strings, StringBuffer and StringBuilder. 6

String	StringBuffer	StringBuilder
String is immutable	StringBuffer is mutable	StringBuilder is mutable
Cannot increase or decrease in size	Can grow or decrease in size, allows for preallocation to decrease time taken for allocating extra characters as the buffer grows.	Can grow or decrease in size, allows for preallocation to decrease time taken for allocating extra characters as the buffer grows
Can be created by directly assigning a primitive String. String s = "Hello"	One of the constructors need to be called to create a StringBuffer.	One of the constructors need to be called to create a StringBuffer.
+ operator can be used to concatenate strings together and also strings with other primitive types or objects.	+ operator cannot be used to append or perform operations on StringBuffer with other data types	+ operator cannot be used to append or perform operations on StringBuilder with other data types
String is thread safe	StringBuffer is thread safe. It is synchronized.	**StringBuilder is not thread safe. It is not synchronized**
Efficient when working on individual string. But when operations such as append is performed, it becomes slow because each time, new String object is created.	**Efficient when working on operations that modify the string because new String does not need to be created everytime.**	**Performance wise faster than StringBuffer because StringBuilder is not synchronized. It is more efficient to use if synchronization is not required.**

Program to demonstrate behavior of String and StringBuffer.

```
//To append World to String
s = s+"World"; // A new string has to be created.
System.out.println("After concatenating and creating new object:"+s);
//append World to StringBuffer
sb.append("World");
System.out.println("After appending to existing object of StringBuffer:"+sb);
   }
```

Output:

After concatenating and creating new object: HelloWorld

After appending to existing object of StringBuffer: HelloWorld

5 a

Explain with code snippet the following methods of StringBuffer : ensureCapacity(), insert(), trimToSize(). Give complete syntax of each method and all overloaded versions. 8

ensureCapacity()

void ensureCapacity(int minCapacity)

to preallocate room for a certain number of characters after a StringBuffer has been

constructed.
Use this to increase the size if you know the buffer is going to increase in size.

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());

Sb.ensureCapacity(200)
System.out.println(sb.capacity());
Output:
        16
        200
```

In the above code snippet, StringBuffer is initially created with 16 by default. If we know that the StringBuffer is going to grow to size of 200 at some point, we can use ensureCapacity to increase the minimum capacity to 200 so that it does not need to be relocated later.

insert()
inserts one string into another.
The overloaded versions are given below:
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)

It is overloaded to accept values of all the primitive types, plus Strings, Objects, and CharSequences
The index specifies the index at which point the string will be inserted into the invoking StringBuffer

```
StringBuffer str2 = new StringBuffer("Hello Jim. How are you?");
str2.insert(6, "Cara and "); // inserts the string at index 6.
                // The remaining portion of the StringBuffer is pushed to the right.

System.out.println(str2.toString());
```

Output: Hello Cara and Jim. How are you?

void trimToSize( )
Requests that the size of the character buffer for the invoking object be reduced to better fit the current contents. This improves performance if we know that the StringBuffer is not going to grow any further.

```
String s = "Hello";
        StringBuffer sb = new StringBuffer(200);
        sb.append("Hello");
        System.out.println("Capacity "+sb.capacity());
        sb.trimToSize();
        System.out.println("Capacity after trimming "+sb.capacity());
        // some operation
```

Output:
Capacity 200
Capacity after trimming 5

	b	Differentiate length() and capacity() of the StringBuffer class.
		length() returns the current length of the StringBuffer whereas capacity() returns the total allocated capacity.
		`StringBuffer sb = new StringBuffer(200);`
		`sb.append("Hello");`
		`System.out.println("Capacity of sb: "+sb.capacity());`
		`System.out.println("Length of sb: "+sb.length());`
		Capacity of sb: 200
			4

Length of sb: 5

In the above code snippet a StringBuffer with minimum capacity of 200 is created. A string of length 5 is appended. As can be observed from the output, the capacity of the StringBuffer is still 200, but the length of the content actually in the string is 5.

6

Create a HashMap of data type(String, Integer). Read a string from the user. Convert the string to lowercase. Assume the string does not have any punctuation. Count the frequency of each word and store it in HashMap. Display the frequency of each word inthe string.

7

Input String: HashMap provides the basic implementation of the Map interface of Java and stores data in Key and Value pairs andyou can access them by an index of another type

Program :

```
import java.util.*;
public class Main
{
	public static void main(String[] args) {
		HashMap<String, Integer> hm = new HashMap();
		Scanner sc = new Scanner(System.in);
		String s = sc.nextLine();
		s= s.toLowerCase();

		for (String st: s.split(" ") ) {
		   int freq = hm.getOrDefault(st,0);
		   hm.put(st,freq+1);
		}

		//entryset
		Set<Map.Entry<String, Integer>> kset = hm.entrySet();
		//Display
		for(Map.Entry<String, Integer> me: kset){
		   System.out.println(me.getKey()+" : "+me.getValue() );
		}

	}
}
```

Output:

```
HashMap provides the basic implementation of the Map interface of Java and stores data in Key
and Value pairs and you can access them by an index of another type
access : 1
data : 1
another : 1
interface : 1
type : 1
pairs : 1
can : 1
java : 1
provides : 1
and : 3
of : 3
by : 1
hashmap : 1
map : 1
value : 1
key : 1
you : 1
```

		in : 1 stores : 1 implementation : 1 index : 1 them : 1 an : 1 the : 2 basic : 1 Explanation :	

7		Given a list of strings with names of the first names of your classmates seperated by a comma. Convert everything to uppercase.Add each name to a priority queue with Comparator that orders the names in descending lexicographical order. Display all the names.
		
		Sample input String : Lisa, Ron, Adam, Stanley, Cassie, Jenna.
		Program:
		
		import java.util.*;
		public class Main
		{
		public static void main(String[] args) {
		PriorityQueue<String> pq = new PriorityQueue(Comparator.reverseOrder());
		Scanner sc = new Scanner(System.in);
		String s = sc.nextLine();
		s= s.toUpperCase();
		
		for (String st: s.split(", ")) {
		pq.add(st);
		}
		System.out.println("\nPoll the queue\n");
		while(!pq.isEmpty())
		System.out.println(pq.poll());
		
		}
		}
		
		Output:
		Lisa, Ron, Adam, Stanley, Cassie, Jenna
		Poll the queue
		
		STANLEY
		RON
		LISA
		JENNA
		CASSIE
		ADAM
			7

8		Write a program to check if a string is a palindrome without reversing the string.
		
		Sample Input String that are palindromes:
		Never odd or even
		Racecar
		
		import java.util.*;
		public class Main
		{
		public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
			7

```
                String s = sc.nextLine();
                s= s.toLowerCase();
                s = s.replaceAll(" ","");
                System.out.println(s);
                boolean isPalim = true;
                int len = s.length();
                for(int i = 0; i<len/2; i++){
                    if (s.charAt(i) != s.charAt(len-i-1)){
                        isPalim = false;
                    }
                }
                if(isPalim)
                    System.out.println("It is a palindrome");
                else
                    System.out.println("It is not a palindrome");

        }
}
```

Output:

```
Never odd or even
It is a palindrome
Racecar
It is a palindrome
Reverse
It is not a palindrome
```