USN | | | | | | | | | | |

Internal Assessment Test 1 – May 2022

Solutions

Sub:	Advanced Java and J2EE					Sub Code:	18CS644	Branch:	CSE / ISE
Date:	10.05.2022	Duration:	90 mins	Max Marks:	50	Sem / Sec:	6 A,B,C		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT

1 (a)	In Java, an "enumeration defines a class type". Explain the meaning of this statement with appropriate code snippets.
	Answer:
	It is possible to specify Constructors, instance variables, methods and implement interfaces.
	Each enumeration constant has it's own copy of instance variables.
	enum Subjects{
	SSC(4), CGV(4), WTA(4), AJJ(3), MAD(2);
	private int credits;
	Subjects(int c){ credits = c; }
	int getCredits(){ return credits; }
	}
	}
	public class demo {
	public static void main(String[ ] args){
	System.out.println(Subjects.AJJ.getCredits());
	}
	}
	Output: 4
	
	In the code snippet above, a constructor is used to set a variable credits.
	A method getCredits is also written that returns the value of the instance variable.
	Each enumeration constant has it's own copy of credits.
		[05]	CO1	L2

(b)	Create an enumeration of BUTTONS with values, CREATE, UPDATE, DELETE. Receive a redirect url for each of the buttons and set a variable redirect to create.jsp, update.jsp and delete.jsp in the constructor of the enumeration respectively.
	Use switch case on an enumeration variable to output a custom notification message to user such as "CREATE button pressed, redirecting to <url>"
	
	`public class Main {`
	`enum BUTTONS{`
	`CREATE("create.jsp"), UPDATE("update.jsp"), DELETE("delete.jsp");`
	`String redirect;`
	
	`BUTTONS(String url){`
	`redirect=url;`
	`}`
	`String getURL() {`
	`return redirect;`
	`}`
	`}`
		[05]	CO1	L3

	`public static void main(String[] args) {`
	`BUTTONS bt = BUTTONS.CREATE;`
	`System.out.println(bt+" pressed. Redirecting to "+bt.getURL());`
	`switch(bt) {`
	`case CREATE:`
	`System.out.println("Creating...");`
	`break;`
	`case DELETE:`
	`System.out.println("Deleting...");`
	`break;`
	`case UPDATE:`
	`System.out.println("Updating...");`
	`break;`
	`}`
	`} }`
	Output:
	CREATE pressed. Redirecting to create.jsp
	Creating...			

2 (a)	What is the output of the following code?
	enum STATUS { OK, Accepted, BadRequest, Found, NotFound };
	System.out.println(STATUS.BadRequest.ordinal());
	System.out.println(STATUS.Found.compareTo(STATUS.NotFound));
	STATUS sok = STATUS.OK;
	STATUS sa = STATUS.Accepted;
	System.out.println(sok.equals(sa));
	2
	-1
	false
	
	Write code to print all the status codes using a for-each loop
	`for (STATUS s: STATUS.values()) {`
	`System.out.println(s);`
	`}`
	Output:
	OK
	Accepted
	BadRequest
	Found
	NotFound
		[04]	CO1	L3

(b)	Explain the methods ordinal(), compareTo() and equals().
	Ordinal is a value that indicates an enumeration constant's position in the list of constants.
	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.
	enum Subject {
	SSC, CGV, WTA, AJJ, MAD;
	}
	public class demo {
	public static void main(String[] args) {
		[06]	CO1	L2

	// ordinal
	System.out.println("Ordinal Values"); for (Subject s: Subject.values()){
	System.out.println(s+":"+s.ordinal());
	}
	Output: Ordinal Values SSC:0
	CGV:1 WTA:2 AJJ:3 MAD:4
	
	The ordinal value of two constants of the same enumeration can be compared 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.
	enum Subject { SSC, CGV, WTA, AJJ, MAD; } public class demo {
	public static void main(String[] args) {
	Subject sj, sj2, sj3;
	sj = Subject.SSC; sj2 = Subject.AJJ; sj3 = Subject.MAD;
	System.out.println("comparing SSC with AJJ: "+sj.compareTo(sj2)); if (sj.compareTo(sj2)<0)
	System.out.println(sj+" comes before "+sj2); else if (sj.compareTo(sj2)>0)
	System.out.println(sj+" comes after "+sj2); else
	System.out.println(sj+ "equals" + sj2);
	
	Output:
	comparing SSC with AJJ: -3 SSC comes before AJJ
	
	equals()
	We can compare for equality an enumeration constant with any other object by using equals( ), which overrides the equals( ) method defined by Object
	
	enum Subject { SSC, CGV, WTA, AJJ, MAD; } public class demo {
	public static void main(String[] args) {
	Subject sj, sj2, sj3;
	sj = Subject.SSC; sj2 = Subject.AJJ; sj3 = Subject.MAD;
	Subjects sj4 = Subjects.SSC; if (sj==sj4){
	System.out.println("sj and sj2 are the same");
	}
	Elective e = Elective.AJJ; boolean b = e.equals(sj2);
	System.out.println("Is Elective.AJJ and Subjects.AJJ same?"+b);
	boolean sjEsj4 = sj.equals(sj4);
	System.out.println("Is sj and sj4 same(using equals)?"+sjEsj4);
	
	Output:
	sj and sj2 are the same
	Is Elective.AJJ and Subjects.AJJ same?false
	Is sj and sj4 same(using equals)?true			

3 (a)	What is the output or error in the following code? (There are no syntax errors) Double x = 10.0; int i = x.intValue(); System.out.println(x+" "+i);	[02]	CO1	L3

	Output 10.0 10			

(b)	Explain and demonstrate autoboxing in expression evaluation.
	Answer:
	Expressions
	Within an expression, a numeric object is automatically unboxed
	Auto-unboxing also allows you to mix different types of numeric objects in an expression.
	Once the values are unboxed, the standard type promotions and conversions are applied.
	
	public abDemo(){
	Integer iob= 10; int I;
	Integer iob2 = 100; // autoboxing takes place
	iob2++;//auto-unboxing to int, after increment, gets autoboxed into object
	iob2 = iob2+(iob2/3);// iob gets unboxed, type promotion takes place on division by 3, then result gets autoboxed into Integer object
	i = iob2+(iob2/3);
	Double dob = 98.6; dob = dob+iob; // automatic type promotion takes place.
	
	Should programmers always use Type wrappers for all primitive types? Why or why not?
	Code will be technically correct. However, less efficient than code that uses the primitive double or int. This is because each autobox and auto-unbox adds overhead. Hence, it is advised to restrict usage of type wrapper only when object representation is required!
		[06]+ [02]	CO1	L2

4 (a)	Answer in one word. i) Annotation type definition is similar to an _____. ii) The default RetentionPolicy is _____. iii) An annotation that has no members is called a _____ annotation. iv) Default values for members are specified using the keyword _____. v) The meta-annotation _____ is used to specify retention policy for an annotation. Answer i) interface ii) CLASS iii) marker iv) default v) @Retention	[2.5M]	CO1	L1

(b)	Explain the following built-in annotations with code snippets and output:
	i) @Retention ii)@Override iii)Inherited
	@Retention
	@Retention(retention-policy)
	- Specifies a retention policy
	- retention-policy should be SOURCE, CLASS or RUNTIME
	- Default is CLASS.
	A retention policy determines at what point an annotation is discarded
	
	import java.lang.annotation.Retention;
	import java.lang.annotation.RetentionPolicy;
	public class annotationsDemo {
	@Retention(RetentionPolicy.RUNTIME) @interface myAnno{
	String str(); int val();
		[04]	CO1	L2

	}
	@myAnno(str="Annotation Example", val = 100)
	public void foo(){
	}
	public annotationsDemo(){
	Class c = this.getClass(); try{
	Method m = c.getMethod("foo");
	}catch(NoSuchMethodException e){
	System.out.println("Method Not Found");
	}
	}
	}
	public class demo {
	public static void main(String[] args) {
	annotationsDemo ad = new annotationsDemo();
	}			

5 (a)	What is the output of the following Code?
	import java.util.*;
	public class Main() {
	public static void main(String[] args) {
	ArrayList<String> al = new ArrayList<String>();
	al.add("ls"); al.add("mv"); al.add("cp"); al.add("cd");
	ArrayList<String> commands = new ArrayList<String>();
	commands.add("grep"); commands.add("find");
	System.out.println(al);
	System.out.println(commands);
	al.addAll(commands);
	System.out.println(al);
	System.out.println("Size of al: "+ al.size());
	System.out.println("Is cp present in al?
	"+al.contains("cp"));
	al.retainAll(commands);
	System.out.println(al);
	String [] arr = new String[5];
	arr =al.toArray(arr);
	System.out.println(arr[1] +" " + arr[3]);
	al.remove(1);
	System.out.println(al);
	al.add("ls"); al.add("mv"); al.add("cd");
	System.out.println(al.subList(1,3));
	}
	}
	Output:
	[ls, mv, cp, cd]
	[grep, find]
	[ls, mv, cp, cd, grep, find]
	Size of al: 6
	Is cp present in al? true
		[04]	CO2	L2

	[grep, find] find null [grep] [ls, mv]			
(b)	Explain any 5 methods used in part (a) of the Collection and List interface in detail with syntax and behavior. boolean **add**(**E** obj) -adds obj of type E to collection -Returns true if added successfully. -Return false if collection does has restrictions like not allowing duplicates. boolean addAll(**Collection**<? extends **E**> c) Adds entire collection boolean **contains**(**Object** o) - check if collection contains ***o*** boolean **retainAll**(Collection<?> c) - removes all elements except those in ***c*** T> T[ ] **toArray**(T[ ] *array*) - if size of *array* **equals** number of elements - items are returned in array. - If size of array is **less** New array of necessary size is allocated and returned. - If size of array is **greater** - Excess elements are set to nul. ArrayStoreException : if any element is not a subtype of array. int **size**() - returns size of elements in an array. Belongs to List interface **List**<**E**>**subList**(int fromIndex, int toIndex) – returns a view of a list fromIndex(inclusive), toIndex(exclusive). default void **sort**(**Comparator**<? super **E**> c) **E remove**(int index) – removes element at index.	[06]	CO2	L2

6 (a)	What is the output of the following Code? Explain any 3 methods pertaining to the List, Queue or Deque interface from the following code in detail.
	
	import java.util.*;
	public class Main() {
	public static void main(String[] args) {
	`LinkedList<Integer> lst = new LinkedList<Integer>();`
	`lst.add(45); lst.add(55); lst.add(2); lst.add(105);`
	`System.out.println(lst);`
	`System.out.println(lst.remove());`
	`System.out.println(lst.peek());`
	`lst.addLast(66);`
	`lst.addFirst(11);`
	`System.out.println(lst);`
	`System.out.println(lst.pop());`
		[6]	CO2	L2

```
        lst.clear();
         System.out.println(lst.poll());
         }
 }

 Output:
[45, 55, 2, 105]
45
55
[11, 55, 2, 105, 66]
11
 Null
```

Queue Interface

Boolean **add**(**E** e)

Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions,
returning true upon success and throwing an **IllegalStateException** if no space is currently available.

poll()

Retrieves and removes the head of this queue, or returns null if this queue is empty.

E peek() Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

E remove()

Retrieves and removes the head of this queue.

Deque Interface

void **addFirst**(**E** obj), void **addLast**(**E** obj)

Adds *obj* to the head (addFirst) or
tail(addLast) of the deque.

E pop()

Pops an element from the stack represented by this
deque.

Collection interface

void clear()

- Removes all items in a collection

(b)	Create a class called Subject with private variables, name, code and credits. Create 3 objects of Subject. Populate with appropriate name, code and credits. Override the toString() method to display name, code and credits. Write appropriate get/set methods that are required. Create a LinkedList of type Subject. Add the 3 objects created to the LinkedList collection. Loop through and display the details of the Subjects.	[4]	CO2	L3

Calculate and display total credits.

```
import java.util.*;
class Subject{
	private String name;
	private String code;
	private int credits;

	public Subject(String n, String c, int cr) {
		name=n;code=c; credits=cr;
	}
	int getCredits() {
		return credits;
	}
	@Override
	public String toString() {
		return name+"-"+code+"-"+credits;
	}
}
public class q6Subject {

		public q6Subject() {
			Subject s1,s2,s3;
			s1= new Subject("AJJ","C103",4); s2=new
Subject("OS","C154",4);
			s3 = new Subject("CGV","C102", 3);
			LinkedList<Subject> al = new LinkedList<Subject>();
			al.add(s1); al.add(s2); al.add(s3);
			int totalCredits=0;
			for(Subject s: al) {
				totalCredits+=s.getCredits();
				System.out.println(s);
			}
			System.out.println("Total Credits "+totalCredits);

				}
}
```

```
AJJ-C103-4
OS-C154-4
CGV-C102-3
Total Credits 11
```

CI CCI HoD

CO PO Mapping

CO-PO and CO-PSO Mapping																	
Course Outcomes	**Blooms Level**	**Modules covered**	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**	**PSO4**

CO1	Interpret the need for advanced JAVA concepts like Enumerations, Wrapper Classes and Annotation.	**L1,L 2,L3**	**1**	2	2	2	0	3	0	0	0	3	0	0	0	3	0	0	2
CO2	Explain Collections Interface and Framework in development of modular applications.	**L1,L 2,L3**	**2**	2	2	2	0	3	0	0	0	3	0	0	0	3	0	0	2
CO3	Use In-Built String Handling Functions for development of Java Programs.	**L1,L 2,L3**	**3**	2	2	2	0	3	0	0	0	3	0	0	0	3	0	0	2
CO4	Describe how servlet fit into Java – Based web application architecture	**L1,L 2,L3**	**4**	3	2	2	0	3	0	0	0	3	0	0	0	3	0	0	3
CO5	Illustrate database access and details for managing information using the JDBC API	**L1,L 2,L3**	**5**	3	3	2	0	3	0	0	0	3	0	0	0	3	0	0	3

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium
PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				