2021 Scheme

1	N	T							

NITTE EDUCATION TRUST

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

An Autonomous Institution Approved by UGC/AICTE/Govt. of Karnataka
Accredited by NBA (Tier – I) and NAAC 'A+' Grade
Affiliated to Visveswaraya Technological University, Belagavi
Post Box No. 6429, Yelahanka, Bengaluru – 560 064, Karnataka, India

KNOWLEDGE • CHARACTER • UNITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MSE 2 SCHEME AND SOLUTION

Course Title with code	OOP with JAVA(21CS35)	**Maximum Marks**	**30 Marks**
Date and Time	3/2/2023, 9.30-10.30am	**No. of Hours**	1.0
Course Instructor(s)	Dr. Vijaya Shetty S/ Mrs. Shruthi Shetty J		

1. Answer any **two full questions.**
2. Any missing data may assume suitably.

Q. No	Question	MAX MARKS
1. a	**Scheme**: Any 2 points--1×2=2m **Solution**:	2

THROW	THROWS
A throw is used to throw an exception explicitly within a method or to rethrow the exception caught in the method.	A throws is used to declare one or more exceptions that are thrown out of a method.
Can throw a single exception using throw.	Multiple can be thrown using Throws.
Throw keyword is used inside the method.	Throws keyword is used in method Signature.
Only unchecked exceptions can be propagated using throw keyword.	Checked exception can be propagated using throws keyword.
Throw keyword is followed by the instance variable	Throws keyword is followed by the exception class

1. b	**Scheme:** **Command line Argument -1m** NullPointerException—2.5m NumberFormatException—2.5m	6m

```
package exam;
public class exception {
        public static void main(String[] args)
        {
                int a = args.length;
                System.out.println("Number of arguments: "+a);
                String S = "nmit";
                if(a==0)
                  S = null;
                else
                  S="333.333";
                try
                {
```

	if (S.equals("NMIT"))
	System.***out***.println("NMIT");
	int a1 = Integer.*parseInt*(S);
	}
	catch(NullPointerException e)
	{
	System.***out***.print("Null Pointer Exception Caught ----"+e);
	}
	catch(NumberFormatException ex){
	System.***err***.println("Invalid string in argumment--- "+ex);
	}
	}
	}
	Output
	Number of arguments: 0
	Null Pointer Exception Caught ----java.lang.NullPointerException: Cannot invoke "String.equals(Object)" because "S" is null
	
	Number of arguments: 1
	Invalid string in argumment--- java.lang.NumberFormatException: For input string: "333.333"	

1. c	Scheme: oddThread—2.5m
	evenThread—2.5m
	main() with call to join—2m
	Solution:
	
	//Using join() to wait for threads to finish.
	class oddThread implements Runnable {
	String name; // name of thread
	int sum=0;
	Thread t;
	oddThread (String threadname) {
	name = threadname;
	t = new Thread(this, name);
	System.out.println("Odd Thread: " + t);
	t.start(); // Start the thread
	}
	//This is the entry point for thread.
	public void run() {
	for(int i = 1; i <=10; i=i+2) {
	System.out.println(name + ": " + i);
	sum+=i;
	}
	System.out.println("Odd sum: " + sum);
	System.out.println(name + " exiting.");
	}
	}
	class evenThread implements Runnable {
	String name; // name of thread
	int sum=0;
	Thread t;
	evenThread (String threadname) {
		7m

```
name = threadname;
t = new Thread(this, name);
System.out.println("Even thread: " + t);
t.start(); // Start the thread
}
//This is the entry point for thread.
public void run() {
for(int i = 0; i <=10; i=i+2) {
System.out.println(name + ": " + i);
sum+=i;
}
System.out.println("Even sum: " + sum);
System.out.println(name + " exiting.");
}
}
class OddEvenThread{
public static void main(String args[]) {
        int sum=0;
oddThread ob1 = new oddThread ("Odd Thread");
evenThread ob2 = new evenThread ("Even  Thread");
try {
ob1.t.join();
ob2.t.join();
} catch (InterruptedException e) {
System.out.println("Main thread Interrupted");
}
System.out.println("odd-even sum="+ (ob1.sum+ob2.sum));
}
}
```

2.a. Scheme: Any 5 methods 0f Collection Interface with description--1×5=5m — 5m

Method	Description
boolean add(E *obj*)	Adds *obj* to the invoking collection. 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.
boolean addAll(Collection<? extends E> *c*)	Adds all the elements of *c* to the invoking collection. Returns **true** if the collection changed (i.e., the elements were added). Otherwise, returns **false**.
void clear( )	Removes all elements from the invoking collection.
boolean contains(Object *obj*)	Returns **true** if *obj* is an element of the invoking collection. Otherwise, returns **false**.
boolean containsAll(Collection<?> *c*)	Returns **true** if the invoking collection contains all elements of *c*. Otherwise, returns **false**.
boolean equals(Object *obj*)	Returns **true** if the invoking collection and *obj* are equal. Otherwise, returns **false**.
boolean isEmpty( )	Returns **true** if the invoking collection is empty. Otherwise, returns **false**.
Iterator<E> iterator( )	Returns an iterator for the invoking collection.
boolean remove(Object *obj*)	Removes one instance of *obj* from the invoking collection. Returns **true** if the element was removed. Otherwise, returns **false**.
boolean removeAll(Collection<?> *c*)	Removes all elements of *c* from the invoking collection. Returns **true** if the collection changed (i.e., elements were removed). Otherwise, returns **false**.
default boolean removeIf(Predicate<? super E> *predicate*)	Removes from the invoking collection those elements that satisfy the condition specified by *predicate*. (Added by JDK 8.)

2.b. Scheme: auto-boxing and auto-unboxing of char and Character types for the switch statement ---2.5m
simple calculator operations—2.5m — 5m

```
package exam;
import java.util.Scanner;
public class AutoPack{
        public static void main(String[] args) {
          // Autoboxint and unboxing for char and Character types in the switch
statement
          Character objC ;
      double a,b;
      Scanner input = new Scanner(System.in);
      System.out.println("Input 2 numbers:");
      a=input.nextDouble();
      b=input.nextDouble();
      System.out.println("Input operator character(+  -  /  *):");
      objC=input.next().charAt(0);
          // here is the auto-unboxing of objC into char type
          switch (objC) {
            case '+':
              System.out.println("Sum= "+ (a+b));
              break;
            case '-':
              System.out.println("a-b= " +(a-b));
              break;
            case '*':
              System.out.println("a*b= "+(a*b));
              break;
            case '/':
              System.out.println("a/b= "+(a/b));
              break;
            default:
              System.out.println("Undefined operation");
          }
        }
      }
```

OUTPUT

```
Input 2 numbers:
2
3
Input operator character(+  -  /  *):
+
Sum= 5.0
```

2.c. Scheme: creating ArrayList—3m
Displaying ArrayList using foreach—2m — 5m

```
import java.util.*;
class ArrayListTraversal{
public static void main(String args[]){
 ArrayList<String> list=new ArrayList<String>();//Creating arraylist
 list.add("Java");//Adding object in arraylist
 list.add("Python");
 list.add("C++");
 list.add("C");
```

	System.out.println("Traversing list through forEach() method:");
	for(String lan:list)
	System.out.println(lan);
	}
	}	

3.a.	Scheme: Explanation about custom exception types—1m Constructors of Exception class –2m toString() method—2m Exception types designed to handle situations specific to our applications are custom exception types. Define a subclass of **Exception** (which is, of course, a subclass of **Throwable**). The **Exception** class does not define any methods of its own. It does, of course, inherit those methods provided by **Throwable**. Thus, all exceptions, including those that you create, have the methods defined by **Throwable** available to them. You may also wish to override one or more of these methods in exception classes that you create. **Exception** defines 2 constructors that can be used for custom exceptions.: **Exception( )** **Exception(String *msg*)** The first form creates an exception that has no description. The second form lets you specify a description of the exception. Although specifying a description when an exception is created is often useful, sometimes it is better to override **toString( )**. Here's why: The version of **toString( )** defined by **Throwable** (and inherited by **Exception**) first displays the name of the exception followed by a colon, which is then followed by your description. By overriding **toString( )**, you can prevent the exception name and colon from being displayed. This makes for a cleaner output, which is desirable in some cases. The following example declares a new subclass of **Exception** and then uses that subclass to signal an error condition in a method. It overrides the **toString( )** method, allowing a carefully tailored description of the exception to be displayed.	5m

3.b.	Scheme: Creating 2 lists—3m
	Merging and displaying—2m
	/* To merge two Linked lists*/
	import java.util.*;
	public class mergeList{
	public static void main(String args[]){
	LinkedList<String> listl=new LinkedList<String>();
	System.out.println("Initial list of elements: "+listl);
	listl.add("XXX");
	listl.add("YYY");
	listl.add("ZZZ");
	System.out.println("List 1: "+listl);
	LinkedList<String> list2=new LinkedList<String>();
	list2.add("AAA");
	list2.add("BBB");
	System.out.println("List 2: "+list2);
	//Adding second list elements to the first list
	ll.addAll(l2);
	System.out.println("After merging List 1: "+listl);
		5m

	} }	

3.c.	Scheme: Creating Hashset: 3m
	Traversing using an Iteraor—2m
	package exam;
	/* HashSet demo*/
	import java.util.*;
	class HashSet1{
	public static void main(String args[]){
	//Creating HashSet and adding elements
	HashSet<String> set=new HashSet();
	set.add("One");
	set.add("Two");
	set.add("Three");
	set.add("Four");
	set.add("Five");
	set.add("Three");
	Iterator<String> i=set.iterator();
	while(i.hasNext())
	{
	System.out.println(i.next());
	}
	}
	}
	Output
	Five
	One
	Four
	Two
	Three
		5m

Faculty Signature	Course Co-Ordinator/Mentor Signature	HoD Signature