ASSIGNMENT -5

Exception and Multithreading

a) Write a Java program to handle each of the following exception during execution. Write a suitable finally block for the selected exceptions wherever found appropriate.

i) InterruptedException

ii) ArrayIndexOutOfBoundsException

iii) IllegalArgumentException

iv) IndexOutOfBoundsException

v) NegativeArraySizeException

vi) NullPointerException

vii) IllegalAccessException

CODE :-

```
import java.util.Scanner;
public class ExceptionDemo {
   static void handleInterrupted() {
      try {
         System.out.println("Thread going to sleep for 2 seconds");
         Thread.sleep(2000);
      }
      catch (InterruptedException e) {
         System.out.println("Error: Thread was interrupted");
      }
      finally {
         System.out.println("Thread operation complete\n");
      }
   }
```

```
static void handleArrayIndexOutOfBounds() {
    int[] arr = {10, 20, 30};
    try {
        System.out.println("Accessing array element at index 3");
        System.out.println("Value at index 3: " + arr[3]);
    }
    catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Error: Cannot access element outside array bounds");
    }
    finally {
        System.out.println("Array elements are:");
        for(int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println("\n");
    }
}

static void handleIllegalArgument(int marks) {
    try {
        if(marks < 0 || marks > 100) {
            throw new IllegalArgumentException();
        }
        System.out.println("Valid marks: " + marks);
    }
    catch (IllegalArgumentException e) {
        System.out.println("Error: Marks should be between 0 and 100");
    }
```

```
    finally {
        System.out.println("Marks validation complete\n");
    }
}

static void handleIndexOutOfBounds() {
    String[] names = {"Ram", "Shyam"};
    try {
        System.out.println("Accessing name at index 2");
        System.out.println("Name at index 2: " + names[2]);
    }
    catch (IndexOutOfBoundsException e) {
        System.out.println("Error: Index is out of bounds");
    }
    finally {
        System.out.println("Available names:");
        for(String n : names) {
            System.out.print(n + " ");
        }
        System.out.println("\n");
    }
}

static void handleNegativeArraySize() {
    try {
        System.out.println("Creating array of size -3");
        System.out.println("Attempting to create the array...");
        int[] arr = new int[-3];
    }
```

```
    catch (NegativeArraySizeException e) {
        System.out.println("Error: Array size cannot be negative");
    }
    finally {
        System.out.println("Array size must be positive\n");
    }
}

static void handleNullPointer() {
    String str = null;
    try {
        System.out.println("Getting length of null string");
        System.out.println("String length: " + str.length());
    }
    catch (NullPointerException e) {
        System.out.println("Error: Cannot perform operations on null object");
    }
    finally {
        System.out.println("String operation complete\n");
    }
}

static void handleIllegalAccess(String name) {
    try {
        if(name == null || name.isEmpty()) {
            throw new IllegalArgumentException();
        }
        System.out.println("Valid name: " + name);
    }
```

```
    catch (IllegalArgumentException e) {
        System.out.println("Error: Name cannot be null or empty");
    }
    finally {
        System.out.println("Name validation complete\n");
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int choice;
    while(true){
        System.out.println("Exception Handling Menu:");
        System.out.println("1. InterruptedException");
        System.out.println("2. ArrayIndexOutOfBoundsException");
        System.out.println("3. IllegalArgumentException");
        System.out.println("4. IndexOutOfBoundsException");
        System.out.println("5. NegativeArraySizeException");
        System.out.println("6. NullPointerException");
        System.out.println("7. IllegalAccess Check");
        System.out.println("8. Exit");
        System.out.print("Enter your choice: ");
        choice = sc.nextInt();
        switch(choice) {
            case 1:
                handleInterrupted();
                break;
            case 2:
                handleArrayIndexOutOfBounds();
```

```
            break;
        case 3:
            System.out.print("Enter marks (0-100): ");
            int marks = sc.nextInt();
            handleIllegalArgument(marks);
            break;
        case 4:
            handleIndexOutOfBounds();
            break;
        case 5:
            handleNegativeArraySize();
            break;
        case 6:
            handleNullPointer();
            break;
        case 7:
            System.out.print("Enter name: ");
            sc.nextLine();
            String name = sc.nextLine();
            handleIllegalAccess(name);
            break;
        case 8:
            System.out.println("Exiting program...");
            sc.close();
            return;
        default:
            System.out.println("Invalid choice! Please try again.\n");
    }
}
```

```
    }
}
```

---

b) An user defined exception class : UserVerySmallNumException. This exception is thrown before performing a/b, on finding that a/b is very small less than 0.00001. Override toString method to display appropriate exception error message.

CODE :-

```
class UseVerySmallNumException extends Exception{
    private final double val;
    UseVerySmallNumException(double v){
        val = v;
    }
    @Override
    public String toString(){
        return "The Quotient " + val + " is too small";
    }
}
public class CustomException {
    static double divide(double a , double b) throws UseVerySmallNumException,
ArithmeticException{
        if(b == 0){
            throw new ArithmeticException();
        }
        double ans = a / b;
        if(ans <= 0.000001){
            throw new UseVerySmallNumException(ans);
        }
        System.out.println("Normal flow");
        return ans;
```

```
    }

    public static void main(String[] args) {
        try {
            for(int i = 0 ; i < 7 ; i ++){
                double dividend = Math.pow(10 , i);
                System.out.println("Result : "  + divide(1 , dividend));
            }
        }
        catch (UseVerySmallNumException e) {
            System.out.println("Caught : " + e);
        }
    }
}
```

---

c) Demonstrate the Shortest Job First Process Scheduling Algorithm using java multithreading. Here, assume that job size is decided by the numbers elements (n) a method is going to print from 1 to n. For example, a method called to print elements from 1 to 5 has higher priority than a method called to print from 1 to 10.

CODE :-

```
import java.util.Scanner;
public class Shortest_Job_First {
    static class Job implements Runnable {
        int id;
        int n;
        Job(int id, int n) {
            this.id = id;
            this.n = n;
        }
```

```
    @Override
    public void run() {
        for (int i = 1; i <= n; i++) {
            System.out.print("J" + id + ":" + i + (i == n ? "" : " "));
            try {
                Thread.sleep(20);
            }
            catch (InterruptedException ignored) {}
        }
        System.out.println();
    }
}

public static void main(String[] args) throws InterruptedException {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter number of jobs: ");
    int m = sc.nextInt();
    Job[] jobs = new Job[m];
    for (int i = 1; i <= m; i++) {
        System.out.print("Enter size for job " + i + ": ");
        int n = sc.nextInt();
        jobs[i - 1] = new Job(i, n);
    }
    for (int i = 0; i < m - 1; i++) {
        int min = i;
        for (int j = i + 1; j < m; j++) {
            if (jobs[j].n < jobs[min].n) {
                min = j;
            }
```

```
            }
            if (min != i) {
                Job tmp = jobs[i];
                jobs[i] = jobs[min];
                jobs[min] = tmp;
            }
        }
        for (int i = 0; i < m; i++) {
            Thread t = new Thread(jobs[i]);
            t.start();
            t.join();
        }
        sc.close();
    }
}
```

---

d) Demonstrate the use of synchronized block for a Inventory class where one and only one thread can update the value of a stock item at a any instance of time

CODE :-

```
class Inventory {
    int val = 100;
    synchronized void update(int x) {
        val += x;
        System.out.println(Thread.currentThread().getName() + ": " + val);
    }
    int get() {
        return val;
    }
```

```
}
class Worker implements Runnable {
    Inventory inv;
    int x;
    Worker(Inventory inv, int x) {
        this.inv = inv;
        this.x = x;
    }
    @Override
    public void run() {
        for(int i = 0; i < 5; i++) {
            inv.update(x);
            try {
                Thread.sleep(100);
            }
            catch(InterruptedException e) {
                System.out.println("Error");
            }
        }
    }
}
public class Synchronized {
    public static void main(String[] args) throws InterruptedException {
        Inventory inv = new Inventory();
        Thread t1 = new Thread(new Worker(inv, 10), "T1");
        Thread t2 = new Thread(new Worker(inv, -10), "T2");
        System.out.println("Start: " + inv.get());
        t1.start();
        t2.start();
```

```
        t1.join();
        t2.join();
        System.out.println("End: " + inv.get());
    }
}
```
