MIT WORLD PEACE UNIVERSITY

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

MULTITHREADING USING THREAD CLASS AND RUNNABLE INTERFACE IN JAVA

PRACTICAL REPORT
ASSIGNMENT 7

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

November 15, 2022

Contents

1 Aim and Objectives

Implementation and Understanding of Exception handling in Java and C++, and to learn and use the exception handling mechanisms with try and catch blocks.

2 Problem Statements

2.1 Problem 1 in Java

Write a program to create a multithreaded calculator that does addition, subtraction, multiplication, and division using separate threads. Additionally also handle '/ by zero' exception by the division method.

2.2 Problem 2 in Java

Print even and odd numbers in increasing order using two threads in Java

3 Theory

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : g++ and gcc on linux for C++, and javac, with JDK 18.0.2 for Java

5 Input

For Java

1. The Side of the Square
2. The Radius of the Circle
3. The Length and Breadth of the Rectangle.

6 Output

For Java

1. The Area of the Shapes
2. The Location of the Hill Stations
3. The Reason the Hill stations are Famous for.

7 Code

7.1 Java Implementation of Problem 1

```java
import java.lang.Thread;
import java.util.Scanner;

class Calculator extends Thread implements Runnable {
    public int a, b, what_to_do = 0;

    Calculator(int a, int b, int choice, String name) {
        this.a = a;
        this.b = b;
        this.what_to_do = choice;
        this.setName(name);
    }

    @Override
    public synchronized void start() {
        System.out.println("Starting the Thread");
        System.out.println("The Name of this Thread is: " + getName());
        super.start();
    }

    @Override
    public void run() {
        switch (what_to_do) {
            case 1:
                System.out.println(a + b);
                break;
            case 2:
                System.out.println(a - b);
                break;
            case 3:
                System.out.println(a * b);
                break;
            case 4:
                try {
                    System.out.println(a / b);
                } catch (ArithmeticException e) {
                    System.out.println("You cant Divide by Zero!");
                }
                break;
            default:
                break;
        }
    }
}

public class assignment_7_problem_1 {
    public static Calculator add, sub, mul, div;
    public static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int choice = 0;
        int a, b;
        System.out.println("Welcome To Thread Calculator of Assignment 7");
        while (choice != 5) {
            System.out.println("What would you like to do? ");
            System.out.println(
                    "1. Addition of 2 Numbers\n2. Subtraction of 2 Numbers\n3.
    Multiplication of 2 Numbers\n4. Division of 2 Numbers\n\n");
            choice = input.nextInt();
            if (choice == 5) {
                break;
            }
            System.out.println("Enter the 2 Numbers");
            a = input.nextInt();
            b = input.nextInt();
            switch (choice) {
                case 1:
                    System.out.println("You have chosen Addition!");
                    add = new Calculator(a, b, choice, "Adder");
                    try {
                        add.start();
                        add.join();
                    } catch (Exception e) {
                        System.out.println("Got some problem with making the
    thread!");
                        System.out.println(e);
                    }
                    break;
                case 2:
                    System.out.println("You have chosen Subtraction!");
                    sub = new Calculator(a, b, choice, "Subtractor");
                    try {
                        sub.start();
                        sub.join();
                    } catch (Exception e) {
                        System.out.println("Got some problem with making the
    thread!");
                        System.out.println(e);
                    }
                    break;
                case 3:
                    System.out.println("You have chosen Multiplication!");
                    mul = new Calculator(a, b, choice, "Multiplier");
                    try {
                        mul.start();
                        mul.join();
                    } catch (Exception e) {
                        System.out.println("Got some problem with making the
    thread!");
                        System.out.println(e);
                    }
                    break;
                case 4:
                    System.out.println("You have chosen Division!");
                    div = new Calculator(a, b, choice, "Divider");
                    try {
                        div.start();
                        div.join();
                    } catch (Exception e) {
                        System.out.println("Got some problem with making the
    thread!");
                        System.out.println(e);
                    }
```

```java
                        break;
                    case 5:
                        System.out.println("You have chosed to Exit!");
                    default:
                        break;
                }

            }
            System.exit(0);
        }
}
```

Listing 1: Full Time Employee.java

7.1.1 Java Output

```
Welcome To Thread Calculator of Assignment 7
What would you like to do?
1. Addition of 2 Numbers
2. Subtraction of 2 Numbers
3. Multiplication of 2 Numbers
4. Division of 2 Numbers


1
Enter the 2 Numbers
2
2
You have chosen Addition!
Starting the Thread
The Name of this Thread is: Adder
4
What would you like to do?
1. Addition of 2 Numbers
2. Subtraction of 2 Numbers
3. Multiplication of 2 Numbers
4. Division of 2 Numbers


4
Enter the 2 Numbers
5
0
You have chosen Division!
Starting the Thread
The Name of this Thread is: Divider
You cant Divide by Zero!
What would you like to do?
1. Addition of 2 Numbers
2. Subtraction of 2 Numbers
3. Multiplication of 2 Numbers
4. Division of 2 Numbers


5
```

Listing 2: Output for Problem 1

7.2 Java Implementation of Problem 2

```java
import java.security.ProtectionDomain;
import java.util.Scanner;

import javax.swing.InputMap;

class printEven extends Thread implements Runnable {
    int limit;

    printEven(int limit) {
        this.limit = limit;
    }

    @Override
    public synchronized void start() {
        super.start();
        System.out.println("Printing Even Numbers");
    }

    @Override
    public void run() {
        for (int i = 0; i < limit; i++) {
            if (i % 2 == 0) {
                System.out.println(i);
            }
        }
    }
}

class printOdd extends Thread implements Runnable {
    int limit;

    printOdd(int limit) {
        this.limit = limit;
    }

    @Override
    public synchronized void start() {
        super.start();
        System.out.println("Printing Odd Numbers");
    }

    @Override
    public void run() {
        for (int i = 0; i < limit; i++) {
            if (i % 2 != 0) {
                System.out.println(i);
            }
        }
    }
}

public class assignment_7_problem_2 {
    static printEven pe;
    static printOdd po;
    static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int limit = 0;
        System.out.println("Enter To what limit Even or Odd numbers you want to
    See");
        limit = input.nextInt();
        pe = new printEven(limit);
        po = new printOdd(limit);
        try {
            pe.start();
            pe.join();
            po.start();
            po.join();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Listing 3: HillStation

7.2.1 Java Output

```
Enter To what limit Even or Odd numbers you want to See
10
Printing Even Numbers
0
2
4
6
8
Printing Odd Numbers
1
3
5
7
9
```

Listing 4: Output for Problem 2

8 Conclusion

Thus, learned to use polymorphism and implemented solution of the given problem statement using C++ and Java.

9 FAQs

1. **Why do we use Exception Handling mechanism?**
   Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

   (a) It helps you avoid the program from crashing
   (b) It helps you set the program control in a more detailed manner
   (c) It helps you manually stop the program safely according to your wish.

```
// Psudo code for exception handling
int a;
try
{
cin>> a;
cout<<33/a; // if a is 0, Exception is thrown, program crashes.
}
catch DivisionByZeroException as e
{
cout<<"cant divide by zero;
}
```

2. **Is it possible to use multiple catch for single throw? Explain?**
   Yes, it is possible to use multiple catch statements for a single throw statement in both C++ and Java. C++ try and catch block works in a specific way, where the throw keyword throws an exception with an integer, or an exception. You can then write multiple catch statements just below the try block.
   In Java, you can throw instances of the Exception class, or throw any of the pre defined exceptions in java.

   Example

```
int a;
try
{
  cin>> a;
  if(a == 0)
  {
    throw a; // int is being thrown.
  }
  else{
    cout<<33/a;
    throw 'a'; // character is being thrown.
  }
}
catch (int a)
{
  cout<<"cant divide by zero;
}
catch (char a)
```

```
    {
      cout<<"A is not zero";
    }

```

```
    try {
            withdrawal_amt = input.nextInt();
            if (withdrawal_amt > amount) {
                throw new Exception("Withdrawal amount more than amount. ");
        new_amount = withdrawal_amt / amount;
            } else {
                amount -= withdrawal_amt;
            }
        } catch (Exception e) {
            System.out.println("Withdrawal Amount more than amount in bank. ")
    ;
            return 0;
        } catch (DivisionByZeroException d)
    {
      System.out.println(d);
    }

```

3. **What is Exception Specification?**
   An exception specification is a contract between the function and the rest of the program. It is a guarantee that the function will not throw any exception not listed in its exception specification.

```
    void f1(void) throw(int) {
      printf_s("About to throw 1\n");
      if (1)
         throw 1;
     }

```

```
    void function_name() throw(Exception)
    {
      if (error)
      {
      throw Exception("Error");
      }
    }

```

4. **What is Re-throwing Exception?**
   If a catch block cannot handle the exception that it was designed to handle, then you can rethrow that exception from that catch block. It causes the original exception to be rethrown.

   Because the exception has already been caught at the scope in which the rethrow expression occurs, it is rethrown out to the next dynamically enclosing try block. Therefore, it cannot be handled by catch blocks at the scope in which the rethrow expression occurred. Any catch blocks for the dynamically enclosing try block have an opportunity to catch the exception.

```
    void f() {
      try {
        cout << "In try block of f()" << endl;
        cout << "Throwing exception of type E1" << endl;
        E1 myException;
        throw myException;
      }
      catch (E2& e) {
        cout << "In handler of f(), catch (E2& e)" << endl;
        cout << "Exception: " << e.message << endl;
        throw;
      }
      catch (E1& e) {
        cout << "In handler of f(), catch (E1& e)" << endl;
        cout << "Exception: " << e.message << endl;
        throw;
      }
    }


```

5. **Explain use of finally keyword in java.**
   Java finally block is a block used to execute important code such as closing the connection, etc.

   Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

   The finally block follows the try-catch block.

```
  class TestFinallyBlock {
    public static void main(String args[]){
    try{
    //below code do not throw any exception
     int data=25/5;
     System.out.println(data);
    }
    //catch won't be executed
    catch(NullPointerException e){
    System.out.println(e);
    }
    //executed regardless of exception occurred or not
     finally {
    System.out.println("finally block is always executed");
    }
```