College of Engineering, Thiruvananthapuram

Object Oriented Programming Lab

Anirudh A. V.

S2 CSE R2, Roll No. 11

Department of Computer Science
& Engineering

February 21, 2022

1 Exception Handling

1.1 Aim

To Wwritea Java program that shows the usage of try, catch, throws, and finally.

1.2 Algorithm

1. Start
2. Create the first try block.
    a. Store the first command-line arguments in a string.
    b. Obtain the length of the string and store it in an integer.
    c. Store the result of the operation in which 100 is divided by the length of the string (first command-line argument).
    d. Print the result of the operation to the user.
3. Write the first catch block which catches ArithemeticException if any and displays the exception to the user.
4. Write the second catch block which catches ArrayIndexOutOfBoundsException if any and displays the exception to the user.
5. Declare the finally block which displays a message no matter the exception
6. Stop

1.3 Code

```
public class ExceptionHandling{
    public static void main(String[] args) throws ArithmeticException{
        try {
            int a = 0;
            int b = 6;
            System.out.println(b/a);
        } catch (Exception e) {
            System.out.println(e);
        }
        finally{

            System.out.println("Excecution Completed!!!");
        }
    }
}
```

1.4 Sample Output

```
java.lang.ArithmeticException: / by zero
Excecution Completed!!!

E:\Anirudh\Anirudh\CET\SEM 3\OOP_Lab\Java_cycle_4>
```

2 User Defined Exceptions

2.1 Aim

To Write a Java program that shows how to create a user-defined exception.

2.2 Algorithm

1. Start

2. Create a CustomException class that inherits from the Exception main class.

3. The constructor for the class accepts an argument that is stored in a class variable.

4. Override the toString() function inherited from the Throwable class.

    a. Return a string that describes an error to the user

5. Create the CustomException public class.

    a. Define a static method that throws a CustomException error.

    b. It checks for the value of the variable and displays a message or throws an error as per the given condition

    c. Declare the public static main class.

    d. It accepts a number less than 10 from the user.

    e. Within a try block, call the static method defined earlier that can throw the custom exception.

    f. Define a catch statement that can catch that custom exception and display an error message to the user.

6. Stop

2.3 Code

```
class InvalidAgeException extends Exception {

  InvalidAgeException(String str) {
    super(str);
  }
}

public class UserException {

  public static void validate(int age) throws InvalidAgeException {
    if (age < 18) {
      throw new InvalidAgeException("Age below minimum age for voting");
    } else {
      System.out.println("Please cast your vote");
    }
  }

  public static void main(String[] args) {
    try {
      validate(13);
    } catch (Exception e) {
      System.out.println("Caught the exception");
      System.out.println("Exception occured : " + e);
    }
  }
}
```

2.4 Sample Output

```
Caught the exception
Exception occured : InvalidAgeException: Age below minimum age for voting

e:\Anirudh\Anirudh\CET\SEM 3\OOP_Lab\Java_cycle_4>
```

3 Multi-threading

3.1 Aim

To Write a Java program that implements a multi-threaded program which has three threads. First thread generates a random integer every 1 second. If the value is even, second thread computes the square of the number and prints. If the value is odd the third thread will print the value of cube of the number.

3.2 Algorithm

1. Start
2. Create a class Square that implements Runnable.
    a. Define a thread, string and a integer variable.
    b. Accept the integer, and the thread name in the constructor.
    c. Store the integer and thread name in the class variables and intiliase the thread.
    d. Declare the run() function.
        i. Print the name of the thread and the square of the integer.
        ii. Sleep the thread for 1 second.
        iii. Catch any exception if it occurs.
3. Create a class Cube that also implements Runnable.
    a. Define a thread, string and a integer variable.
    b. Accept the integer, and the thread name in the constructor.
    c. Store the integer and thread name in the class variables and intiliase the thread.
    d. Declare the run() function.
        i. Print the name of the thread and the cube of the integer.
        ii. Sleep the thread for 1 second.
        iii. Catch any exception if it occurs.

4. Create a class Generates that also implements Runnable.

    a. Define a thread and a string.

    b. Accept the thread name as an argument in the constructor.

    c. Store the thread name in the class variables and initialise the thread.

    d. Declare the run() function.

        i. In the run() function declare an object of Random().

        ii. Create a loop that loops ten times

            1. In each iteration, generate a random integer in the range of 0 to 10.

            2. If the generated number is divisible by 2, create an object of the Square class and thus create a new thread.

            3. Else, create an object of the Cube class and thus create a new thread.

            4. Sleep the thread for 1 second.

        iii. Catch any exception if it occurs.

5. Declare the main function and create an object of the Generate class.

6. Stop

3.3 Code

```
import java.util.Random;

class Square implements Runnable {
    Thread t;
    String thread;
    int num;

    Square(int num, String thr) {
        thread = thr;
        this.num = num;
        t = new Thread(this, thread);
        t.start();
    }

    public void run() {
        try {
            System.out.println(thread + ": Square is " + num * num);
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("Exception Occurred!! " + e);
        }
    }
}

class Cube implements Runnable {
    Thread t;
    String thread;
    int num;

    Cube(int num, String thr) {
        this.num = num;
        thread = thr;
        t = new Thread(this, thread);
        t.start();
    }

    public void run() {
        try {
            System.out.println(thread + ": Cube is " + num * num * num);
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("Exception Occurred!! " + e);
        }
    }
}

class Generate implements Runnable {
```

```
    Thread t;
    String thread;

    Generate(String thr) {
        thread = thr;
        t = new Thread(this, thread);
        t.start();
    }

    public void run() {
        Random rd = new Random();
        for (int i = 0; i < 10; i++) {
            try {
                int num = rd.nextInt(10);
                System.out.println(thread + ": Number: " + num);
                if (num % 2 == 0) {
                    Square sq = new Square(num, "Thread 2");
                } else {
                    Cube cb = new Cube(num, "Thread 3");
                }
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Exception Occurred!! " + e);
            }
        }
    }
}

public class Even_oddThread {
    public static void main(String args[]) {
        Generate g1 = new Generate("Thread 1");
    }
}
```

3.4 Sample Output

```
Thread 1: Number: 6
Thread 2: Square is 36
Thread 1: Number: 6
Thread 2: Square is 36
Thread 1: Number: 9
Thread 3: Cube is 729
Thread 1: Number: 3
Thread 3: Cube is 27
Thread 1: Number: 5
Thread 3: Cube is 125
Thread 1: Number: 2
Thread 2: Square is 4
Thread 1: Number: 8
Thread 2: Square is 64
Thread 1: Number: 8
Thread 2: Square is 64
Thread 1: Number: 5
Thread 3: Cube is 125
Thread 1: Number: 1
Thread 3: Cube is 1

e:\Anirudh\Anirudh\CET\SEM 3\OOP_Lab\Java_cycle_4>
```

4 Thread Synchronization

4.1 Aim

To write a Java program that shows thread synchronization.

4.2 Algorithm

1. Start

2. Create a class main.

    1. Declare a method inside it that prints the first square bracket and sleeps the thread.

    2. Catch Exceptions if any.

    3. Print the thread name and the final square bracket.

3. Create the class Synchronisation which implements Runnable

4. Declare a thread, a string, an Object of the Main class, and an integer flag.

5. Accept the thread name, integer value, and the object of Main as the constructor arguments.

    1. Assign the integer value to the class integer variable and the object to the class object variable.

    2. Initialise the thread

6. Declare the run() function which class the method of the Main class in the synchronized block.

7. In the main function, Create 3 class variables. This should start the threads and the synchronized messages are printed thereafter.

8. Stop

4.3 Code

```
class Main {
    void call(String thread) {
        System.out.printf("[");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("Exception Occurred!! " + e);
        }
        System.out.printf("%s", thread);
        System.out.println("]");
    }
}

class Synchronisation implements Runnable {
    String thread;
    Thread t;
    Main m1;
    int flag;
    Synchronisation(String thr, int flag, Main m) {
        thread = thr;
        m1 = m;
        this.flag = flag;
        t = new Thread(this, thread);
        t.start();
    }

    public void run() {
        // Main m1 = new Main();
        if (flag == 0) {
            m1.call(thread);
        } else {
            synchronized (m1) {
                m1.call(thread);
            }
        }
    }
}

public class Sync {
    public static void main(String args[]) {
        Main m = new Main();
        System.out.println("With thread Synchronisation...");
        Synchronisation s4 = new Synchronisation("Thread 1", 1, m);
        Synchronisation s5 = new Synchronisation("Thread 2", 1, m);
        Synchronisation s6 = new Synchronisation("Thread 3", 1, m);
    }
}
```

4.4 Sample Output

```
With thread Synchronisation...
[Thread 1]
[Thread 3]
[Thread 2]

e:\Anirudh\Anirudh\CET\SEM 3\OOP_Lab\Java_cycle_4>
```

5 Creating Two Threads

5.1 Aim

To write a Java program to create two threads: One for displaying all odd numbers between 1 and 100 and the second thread for displaying all even numbers between 1 and 100.

5.2 Algorithm

1. Start

2. Create a class main.

    A) Declare a method call inside it that prints the thread name and an integer as arguments

    B) Display the integer and the thread

3. Create the class Synchronisation which implements Runnable

4. Declare a thread, a string, an Object of the Main class, and an integer n.

5. Accept the thread name, integer value, and the object of Main as the constructor arguments.

    A) Assign the integer value to the class integer variable and the object to the class object variable.

    B) Initialise the thread

6. Declare the run() function which class the method of the Main class in the synchronized block.

7. In the main function, Create an object of the Main class.

8. Loop through numbers from 1 to 100.

9. If the number is divisible by 2, Create an object of the Synchronisation class and pass a thread name, an object of Main, and an integer as arguments. Else, Create an object of the Synchronization class and pass another thread name, the object of Main, and an integer as arguments.

10. Wait till the thread execution completes with the thread.join() method.

11. Catch exceptions if any.

12. Stop

5.3 Code

```
class Thread01 implements Runnable  {

    @Override
    public void run() {
        try {
            for (int i = 0; i <= 100; i++) {
                if (i%2==0){
                    System.out.println("Even nummber - "+i);
                }
                Thread.sleep(100);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

class Thread02 implements Runnable {

    @Override
    public void run() {
        try {

            for (int i = 0; i <= 100; i++) {
                if (i%2!=0){
                    System.out.println("Odd nummber - "+i);
                }
                Thread.sleep(100);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}

public class OddEven {
```

```
    public static void main(String[] args) {
        Thread01 obj1 = new Thread01();
        Thread t1 = new Thread(obj1);

        Thread02 obj2 = new Thread02();
        Thread t2 = new Thread(obj2);

        t1.start();
        t2.start();
    }
}
```

5.4 Sample Output

```
PS E:\Anirudh\Anirudh\CET\SEM 3\OOP_Lab\Java_cycle_4> javac OddEven.java
PS E:\Anirudh\Anirudh\CET\SEM 3\OOP_Lab\Java_cycle_4> java OddEven
Even nummber - 0
Odd nummber - 1
Even nummber - 2
Odd nummber - 3
Even nummber - 4
Odd nummber - 5
Even nummber - 6
Odd nummber - 7
Even nummber - 8
Odd nummber - 9
Even nummber - 10
Odd nummber - 11
Even nummber - 12
Odd nummber - 13
Even nummber - 14
Odd nummber - 15
Even nummber - 16
Odd nummber - 17
Even nummber - 18
Odd nummber - 19
Even nummber - 20
Odd nummber - 21
Even nummber - 22
Odd nummber - 23
Even nummber - 24
Odd nummber - 25
Even nummber - 26
Odd nummber - 27
Even nummber - 28
Odd nummber - 29
Even nummber - 30
Odd nummber - 31
Even nummber - 32
Odd nummber - 33
Even nummber - 34
Odd nummber - 35
Even nummber - 36
Odd nummber - 37
Even nummber - 38
Odd nummber - 39
Even nummber - 40
Odd nummber - 41
Even nummber - 42
```

```
Odd nummber - 57
Even nummber - 58
Odd nummber - 59
Even nummber - 60
Odd nummber - 61
Even nummber - 62
Odd nummber - 63
Even nummber - 64
Odd nummber - 65
Even nummber - 66
Odd nummber - 67
Even nummber - 68
Odd nummber - 69
Even nummber - 70
Odd nummber - 71
Even nummber - 72
Odd nummber - 73
Even nummber - 74
Odd nummber - 75
Even nummber - 76
Odd nummber - 77
Even nummber - 78
Odd nummber - 79
Even nummber - 80
Odd nummber - 81
Even nummber - 82
Odd nummber - 83
Even nummber - 84
Odd nummber - 85
Even nummber - 86
Odd nummber - 87
Even nummber - 88
Odd nummber - 89
Even nummber - 90
Odd nummber - 91
Even nummber - 92
Odd nummber - 93
Even nummber - 94
Odd nummber - 95
Even nummber - 96
Odd nummber - 97
Even nummber - 98
Odd nummber - 99
Even nummber - 100
PS E:\Anirudh\Anirudh\CET\SEM 3\OOP Lab\Java cycle 4>
```

6 Thread Priority

6.1 Aim

To Write a Java program that shows thread priorities.

6.2 Algorithm

1. Start

2. Create a class main.

3. Declare a method call inside it that accepts a thread name, a message, and an integer as arguments

4. Display the thread, message, and priority.

5. Create the class Synchronisation which implements Runnable

6. Declare a thread, a string, an Object of the Main class, and an integer n.

7. Accept the thread name, integer value, and the object of Main as the constructor arguments.

8. Assign the integer value to the class integer variable and the object to the class object variable and set the priority of the thread as the value of the integer argument.

9. Initialise the thread

10. Declare the run() function.

    1. Call the call method of the Main class object twice. Pass the thread name, a string message, and the priority value as the arguments each time.

    2. Sleep the thread for one second.

    3. Catch Exceptions if any.

    4. Call the call method of the Main class object once more. Pass the thread name, a string message, and the priority value as the arguments each time.

11. In the main function, Create 3 class variables. This should start the threads and the synchronized messages are printed thereafter.

12. Stop

6.3Code

```
class T1 implements Runnable {

    @Override
    public void run() {
        try {
            for (int i = 0; i <= 10; i++) {
                Thread.sleep(1000);
                if (i%2==0){
                    System.out.println("Even nummber - "+i);
                }

            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

class T2 implements Runnable {

    @Override
    public void run() {
        try {

            for (int i = 0; i <= 10; i++) {
                Thread.sleep(1000);
                if (i % 2 != 0) {
                    System.out.println("Odd nummber - " + i);
                }

            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}

public class Priority {
```

```
    public static void main(String[] args) {
        // T1 obj1 = new T1();
        Thread t1 = new Thread(new T1());

        // T2 obj2 = new T2();
        Thread t2 = new Thread(new T2());

        t1.start();

        System.out.println("Thread 1 priority : " + t1.getPriority());
        t1.setPriority(1);
        System.out.println("Thread 1 new priority : " + t1.getPriority());
        t2.start();
        System.out.println("Thread 2 priority : " + t2.getPriority());
        t2.setPriority(3);
        System.out.println("Thread 2 new priority : " + t2.getPriority());
    }
}
```

6.4 Sample Output

```
Thread 1 priority : 5
Thread 1 new priority : 1
Thread 2 priority : 5
Thread 2 new priority : 3
Even nummber - 0
Odd nummber - 1
Even nummber - 2
Odd nummber - 3
Even nummber - 4
Odd nummber - 5
Even nummber - 6
Odd nummber - 7
Even nummber - 8
Odd nummber - 9
Even nummber - 10

e:\Anirudh\Anirudh\CET\SEM 3\OOP_Lab\Java_cycle_4>
```