

## WEEK 12

1. Write a program that calls a method that throws an exception of type `ArithmeticException` in a for loop at an undesirable situation (such as divide by zero or taking square root of negative number). Catch the exception and display appropriate message. (Example of Unchecked Exception).

```
Main.java x
1 package week12_1;
2
3 public class Main {
4     public static int divide(int dividend, int divisor) { 1 usage
5         return dividend / divisor;
6     }
7     public static void main(String[] args) {
8         for (int i = 3; i >= -3; i--) {
9
10            try {
11                int result = divide(100, i);
12                System.out.println("Result of 100 / " + i + " = " + result);
13            } catch (ArithmeticException e) {
14                System.out.println("Error: Cannot divide by zero. Skipping i = " + i);
15            }
16        }
17        System.out.println("Loop finished.");
18    }
19 }
```

```
Result of 100 / 3 = 33
Result of 100 / 2 = 50
Result of 100 / 1 = 100
Error: Cannot divide by zero. Skipping i = 0
Result of 100 / -1 = -100
Result of 100 / -2 = -50
Result of 100 / -3 = -33
Loop finished.
```

2. Write a program of your choice where a Checked Exception occurs at third function but handled at the first calling function. Use both ways of managing Checked Exception i.e. using try-catch block and throws keyword.

# WEEK 12

```
Main.java x
1 package week12_2;
2
3 import java.io.FileInputStream;
4 import java.io.IOException;
5
6 public class Main {
7     public void f1() { 1 usage
8         System.out.println("f1: Calling f2() (Inside try-catch block)");
9         try {
10             f2();
11         } catch (IOException e) {
12             System.out.println("f1: Exception was caught!");
13             System.out.println("f1: Error Details: " + e.getMessage());
14         }
15         System.out.println("f1: Complete Execution.");
16     }
17     public void f2() throws IOException { 1 usage
18         System.out.println(" f2: Calling f3()");
19         f3();
20         System.out.println(" f2: This Line Not Executed.");
21     }
22     public void f3() throws IOException { 1 usage
23         System.out.println(" f3: Trying to read a non-existent file");
24         FileInputStream fis = new FileInputStream("random.txt");
25         System.out.println(" f3: This Line Not Executed.");
26     }
27     public static void main(String[] args) {
28         Main demo = new Main();
29         demo.f1();
30     }
31 }
```

```
f1: Calling f2() (Inside try-catch block)
  f2: Calling f3()
    f3: Trying to read a non-existent file
f1: Exception was caught!
f1: Error Details: random.txt (The system cannot find the file specified)
f1: Complete Execution.
```

## WEEK 12

3. You are developing an online banking system where users can transfer money between accounts. If a user tries to withdraw more money than is available in their account, an `InsufficientFundsException` should be thrown.

```
Main.java x BankAccount.java
1 package week12_3;
2
3 public class Main {
4     public static void main(String[] args) {
5         BankAccount account = new BankAccount( initialBalance: 1000.0);
6         try {
7             account.withdraw( amount: 500.0);
8             account.withdraw( amount: 800.0);
9             System.out.println("This message will not be seen");
10        } catch (InsufficientFundsException e) {
11            System.out.println("Error Caught: " + e.getMessage());
12        }
13    }
14 }
15 }

Main.java x BankAccount.java x
1 package week12_3;
2
3 class InsufficientFundsException extends Exception { 3 usages
4     public InsufficientFundsException(String message) { 1 usage
5         super(message);
6     }
7 }
8 class BankAccount { 2 usages
9     private double balance; 6 usages
10
11     public BankAccount(double initialBalance) { 1 usage
12         this.balance = initialBalance;
13         System.out.println("Account created with balance: " + this.balance);
14     }
15
16     public void withdraw(double amount) throws InsufficientFundsException {
17         System.out.println("Attempting to withdraw: " + amount);
18
19         if (amount > this.balance) {
20             throw new InsufficientFundsException("Withdrawal failed. Balance:" + this.balance);
21         }
22         this.balance -= amount;
23         System.out.println("Success. New balance: " + this.balance);
24     }
25 }
```

```
Account created with balance: 1000.0
Attempting to withdraw: 500.0
Success. New balance: 500.0
Attempting to withdraw: 800.0
Error Caught: Withdrawal failed. Balance:500.0
```

## WEEK 12

4. Create a user-defined exception `InvalidAgeException` when the age of a person is below 18 years. Use this exception at appropriate place.

```
1 package week12_4;
2
3 class InvalidAgeException extends Exception { 4 usages
4     public InvalidAgeException(String message) { 1 usage
5         super(message);
6     }
7 }
8 public class Main {
9     public static void validateVoter(int age) throws InvalidAgeException { 2 usages
10         if (age < 18) {
11             throw new InvalidAgeException("You must be 18 or older to vote.");
12         }
13         System.out.println("Your vote is registered.");
14     }
15 public static void main(String[] args) {
16     try {
17         validateVoter( age: 25);
18     } catch (InvalidAgeException e) {
19         System.out.println("Error Caught: " + e.getMessage());
20     }
21     System.out.println();
22     try {
23         validateVoter( age: 15);
24     } catch (InvalidAgeException e) {
25         System.out.println("Error Caught: " + e.getMessage());
26     }
27 }
28 }
```

Your vote is registered.

Error Caught: You must be 18 or older to vote.