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 Unhecked Exception).

Main.java

```
package week12_1;

public class Main {
    public static int divide(int dividend, int divisor) {  1 usage
        return dividend / divisor;
    }
    public static void main(String[] args) {
        for (int i = 3; i >= -3; i--) {

            try {
                int result = divide( dividend: 100, i);
                System.out.println("Result of 100 / " + i + " = " + result);
            } catch (ArithmeticException e) {
                System.out.println("Error: Cannot divide by zero. Skipping i = " + i);
            }
        }
        System.out.println("Loop finished.");
    }
}
```

```
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

```java
package week12_2;

import java.io.FileInputStream;
import java.io.IOException;

public class Main {
    public void f1() {  1 usage
        System.out.println("f1: Calling f2() (Inside try-catch block)");
        try {
            f2();
        } catch (IOException e) {
            System.out.println("f1: Exception was caught!");
            System.out.println("f1: Error Details: " + e.getMessage());
        }
        System.out.println("f1: Complete Execution.");
    }
    public void f2() throws IOException {  1 usage
        System.out.println("  f2: Calling f3()");
        f3();
        System.out.println("  f2: This Line Not Executed.");
    }
    public void f3() throws IOException {  1 usage
        System.out.println("    f3: Trying to read a non-existent file");
        FileInputStream fis = new FileInputStream( name: "random.txt");
        System.out.println("    f3: This Line Not Executed.");
    }
    public static void main(String[] args) {
        Main demo = new Main();
        demo.f1();
    }
}
```

```
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

```java
package week12_3;

public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount( initialBalance: 1000.0);
        try {
            account.withdraw( amount: 500.0);
            account.withdraw( amount: 800.0);
            System.out.println("This message will not be seen");

        } catch (InsufficientFundsException e) {
            System.out.println("Error Caught: " + e.getMessage());
        }
    }
}
```

BankAccount.java

```java
package week12_3;

class InsufficientFundsException extends Exception {  3 usages
    public InsufficientFundsException(String message) {  1 usage
        super(message);
    }
}
class BankAccount {  2 usages
    private double balance;  6 usages

    public BankAccount(double initialBalance) {  1 usage
        this.balance = initialBalance;
        System.out.println("Account created with balance: " + this.balance);
    }

    public void withdraw(double amount) throws InsufficientFundsException {
        System.out.println("Attempting to withdraw: " + amount);

        if (amount > this.balance) {
            throw new InsufficientFundsException("Withdrawal failed. Balance:" + this.balance);
        }
        this.balance -= amount;
        System.out.println("Success. New balance: " + this.balance);
    }
}
```

```
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.

Main.java

```
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 usag
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.
```