DAY 6

/\*

Exception Handling

Whenever Exception occurs, it abruptly terminates the program.

We need to handle exceptions to avoid this.

Throwable

- Exception: IOException, SQLException, ClassNotFoundException, RuntimeException

- Error: StackOverflowError

Types of Exceptions:

1) Checked Exception - Occur at compile time.

2) Unchecked Exception - Occur at runtime.

3) Error - Beyond user's control.

RuntimeException(and all it's child classes) are Unchecked Exceptions.

IOException, SQLException, ClassNotFoundException are Checked Exceptions.

RuntimeException:

ArithmeticException

NumberFormatException

NullPointerException

IndexOutOfBoundsException - ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

Keywords in Exception Handling: try, catch, finally, throw, throws

\*/

/\*

import java.util.Scanner;

class TestException {

public static void main(String[] args) {

int a, b;

Scanner sc = new Scanner(System.in);

System.out.print("Enter value for a & b: ");

a = sc.nextInt();

b = sc.nextInt();

try {

System.out.println("Result: " + (a / b));

}

catch(ArithmeticException e) {

// System.out.println(e);

e.printStackTrace();

}

System.out.print("Our task is over.");

}

}

\*/

/\*

//Multiple Catch block

//one try{} can have multiple catch(){}

import java.util.Scanner;

class TestException {

public static void main(String[] args) {

int a, b;

String s = null;

Scanner sc = new Scanner(System.in);

System.out.print("Enter value for a & b: ");

a = sc.nextInt();

b = sc.nextInt();

try {

System.out.println("Result: " + (a / b));

System.out.println(s.length());

}

catch(ArithmeticException e) {

System.out.println(e);

// e.printStackTrace();

}

catch(NullPointerException e) {

System.out.println(e);

}

System.out.print("Our task is over.");

}

}

\*/

/\*

import java.util.Scanner;

class TestException {

public static void main(String[] args) {

int a, b;

// String s = null;

String s = "Hello";

int arr[] = {1, 2, 3, 4, 5};

Scanner sc = new Scanner(System.in);

System.out.print("Enter value for a & b: ");

a = sc.nextInt();

b = sc.nextInt();

try {

System.out.println("Result: " + (a / b));

System.out.println(s.length());

System.out.println(arr[8]);

}

catch(ArithmeticException e) {

System.out.println(e);

// e.printStackTrace();

}

catch(NullPointerException e) {

System.out.println(e);

}

catch(Exception e) {

System.out.println(e);

}

System.out.print("Our task is over.");

}

}

\*/

/\*

import java.util.Scanner;

class TestException {

public static void main(String[] args) {

int a, b;

String s = null;

// String s = "Hello";

int arr[] = {1, 2, 3, 4, 5};

Scanner sc = new Scanner(System.in);

System.out.print("Enter value for a & b: ");

a = sc.nextInt();

b = sc.nextInt();

try {

System.out.println("Result: " + (a / b));

System.out.println(s.length());

System.out.println(arr[8]);

}

catch(Exception e) {

System.out.println(e);

}

System.out.print("Our task is over.");

}

}

\*/

/\*

Contents of finally block is always executed no matter what. So, very important codes

that must be executed must be in finally block.

\*/

//finally block

/\*

import java.util.Scanner;

class TestException {

public static void main(String[] args) {

int a, b;

String s = null;

// String s = "Hello";

int arr[] = {1, 2, 3, 4, 5};

Scanner sc = new Scanner(System.in);

System.out.print("Enter value for a & b: ");

a = sc.nextInt();

b = sc.nextInt();

try {

System.out.println("Result: " + (a / b));

System.out.println(s.length());

System.out.println(arr[8]);

}

catch(Exception e) {

System.out.println(e);

}

finally {

System.out.println("Some very important codes that must be executed");

}

System.out.print("Our task is over.");

}

}

\*/

/\*

import java.util.Scanner;

class TestException {

public static void main(String[] args) {

int a, b;

String s = null;

// String s = "Hello";

int arr[] = {1, 2, 3, 4, 5};

Scanner sc = new Scanner(System.in);

System.out.print("Enter value for a & b: ");

a = sc.nextInt();

b = sc.nextInt();

try {

System.out.println("Result: " + (a / b));

System.out.println(s.length());

System.out.println(arr[8]);

}

finally {

System.out.println("Some very important codes that must be executed");

}

System.out.print("Our task is over.");

}

}

\*/

/\*

// final vs finally vs finalized

final: keyword (Non-access modifier)

finally: block

finalized: method

final class cannot be inherited: final class Test

final method cannot be overridden: final void show()

final variable cannot be changed: final int a = 50;

finalize(): It is invoked during garbage collection

System.gc() -> will invoke finalize()

\*/

//throw: used to generate an exception

/\*

import java.util.Scanner;

class Screening {

float mark;

void input() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter mark: ");

mark = sc.nextFloat();

}

}

class Result {

public static void main(String[] args) {

Screening s = new Screening();

s.input();

if(s.mark >= 35) {

System.out.println("Pass");

}

else {

throw new ArithmeticException("Fail. Reapper for exam.");

}

}

}

\*/

/\*

// Custom Exception (User Defined Exception)

import java.util.Scanner;

class ResultException extends RuntimeException {

ResultException(String error) {

super(error);

}

}

class Screening {

float mark;

void input() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter mark: ");

mark = sc.nextFloat();

}

}

class Result {

public static void main(String[] args) {

Screening s = new Screening();

s.input();

if(s.mark >= 35) {

System.out.println("Pass");

}

else {

throw new ResultException("Fail. Reapper for exam.");

}

}

}

\*/

/\*

Exception Propagation:

When a statement inside a function generates an exception and it is not handled in the

same function, then that exception propagates to the caller function.

// Unchecked Exceptions are propagated automatically

// To propagate Checked Exceptions, we have to declare them by using throws with

//the function signature

main() {

fun()

}

fun() {

try {

demo()

}

catch() {

}

}

demo() {

Statement; // if this statement generates exception

}

\*/

/\*

Scanner sc = new Scanner(System.in)

Why passing System.in while creating Scanner object?

No default constructor in Scanner class, so can't create

object by Scanner sc = new Scanner();

We are invoking constructor which needs InputStream class object.

"in" is an object of InputStream class. It is present as static member of System class.

So, we pass System.in as argument while creating Scanner object.

Hence, Scanner sc = new Scanner(System.in);

\*/

// Equality Operator ==

class TestEquality {

public static void main(String[] args) {

String s1 = new String("Hello");

String s2 = new String("Hello");

//String s1 = "Hello";

//String s2 = "Hello";

System.out.println(s1 == s2); //== compares the references and not content

System.out.println(s1.equals(s2)); //equals() compares content

}

}