

# Exception Handling

2301260 Programming Techniques

ผศ. ศศิภา พันธุ์ดีธร ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

# Chapter outline

- What an exception is?
- Java exception hierarchy
- Types of exception
- Exception handling
- try statement
  - try block
  - catch block
  - finally block
- How to throw exceptions
  - throw statement
  - Methods that throw exceptions
- Exception propagation
- Define and use your own types of exceptions

# Exception handling

- When a program runs into a runtime error, the program terminates abnormally
- Normally, we want the program continues to run or terminate properly.
- So, we use exception handling to handle exceptions and errors that occur in our programs.

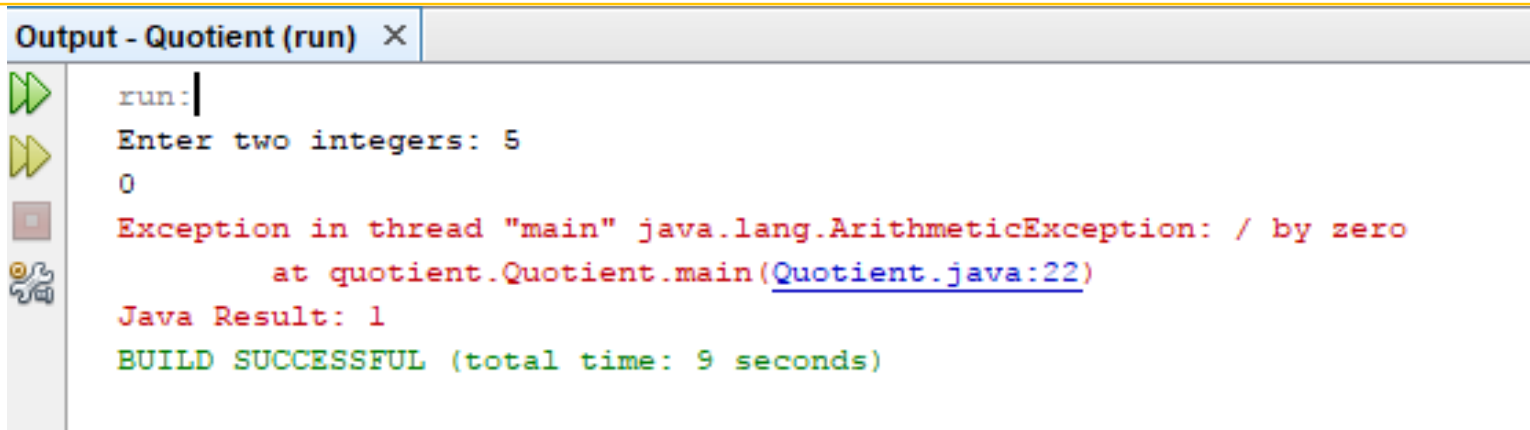
# No exception handling

```
import java.util.Scanner;

public class Quotient {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two integers
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();

        System.out.println(number1 + " / " + number2 + " is " + (number1 / number2));
    }
}
```



The screenshot shows a Java IDE's output window titled "Output - Quotient (run)". It displays the execution of the program. The user entered "5" for the first integer and "0" for the second. This resulted in an "Exception in thread 'main' java.lang.ArithmeticException: / by zero" at line 22 of the file "Quotient.java". The output also shows "Java Result: 1" and "BUILD SUCCESSFUL (total time: 9 seconds)".

```
Output - Quotient (run) X
run:
Enter two integers: 5
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at quotient.Quotient.main(Quotient.java:22)
Java Result: 1
BUILD SUCCESSFUL (total time: 9 seconds)
```

# No exception handling

- If we do not catch the exception, java interpreter will deal with the exception by
  1. Display the exception
  2. Display the execution stack to show where the exception occurs and how the exception is propagated
  3. Stop running the program
- What is exception????
  - Indication of problem during execution. For example, divide by zero, array index out of bound, negative array size, file not found, out of memory

# How to deal with the problem?

- Use if statement

```
import java.util.Scanner;

public class QuotientWithIf {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two integers
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();

        if (number2 != 0)
            System.out.println(number1 + " / " + number2 + " is " + (number1 / number2));
        else
            System.out.println("Divisor cannot be zero ");
    }
}
```

# How to deal with the problem?

- Use if statement in the method

```
import java.util.Scanner;


public class QuotientWithMethod {
    public static int quotient(int number1, int number2) {
        if (number2 == 0) {
            System.out.println("Divisor cannot be zero");
            System.exit(1);
        }

        return number1 / number2;
    }
}
```

# How to deal with the problem?

- Use exception handling

```
public class Quotient {  
    public static void main(String[] args) {  
        int i = 1;  
        int j = 0;  
        try {  
            System.out.println("Try block entered " + "i = " + i + " j = " + j);  
            System.out.println(i/j); // Divide by 0 - exception thrown  
            System.out.println("Ending try block");  
        }  
        catch(ArithmeticException e) { // Catch the exception  
            System.out.println("Arithmetic exception caught");  
        }  
        System.out.println("After try block");  
    }  
}
```





## **Output**

Try block entered i = 1 j = 0

Arithmetic exception caught

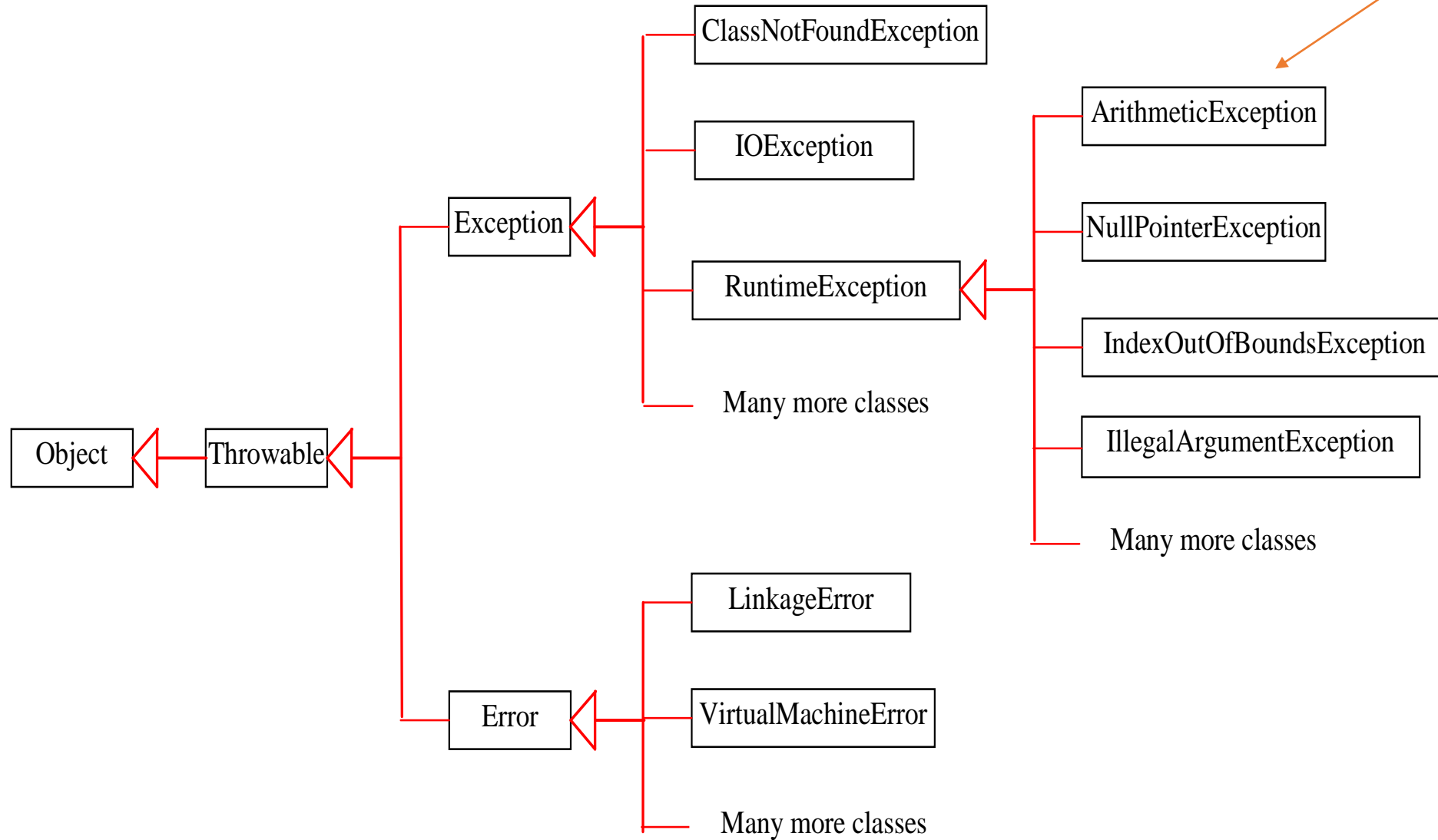
After try block

## Question?

หากแก้ไข i = 1 j = 1 ผลลัพธ์คืออะไร

ตัวอย่างนี้ จาวาเป็นผู้โยน **exception** ออกมาแล้ว **programmer** เป็น  
ผู้จัดการโดยการเขียน **try-catch** ดังไว้

# Exception Types



# Exception handling

- Enables programmers to create applications that can resolve (or handle) exceptions
- Handling exceptions allows a program to continue executing  
-> robust and fault-tolerant program
- The exception is said to be *thrown*
- The code receiving the exception object as a parameter is said to *catch* it.
- That is, the object identifying the exceptional circumstance is *thrown* as an argument to a specific piece of program code that has been written specifically to deal with (*catch*) that kind of problem

# Exception

- In java, exception is an **object** that is created and thrown when an abnormal situation arises in program
- Exception class is a subclass of Throwable class
- Throwable class has 2 constructors
  - A default constructor
  - A constructor that accepts an argument of String type. This string is used to describe the problem causing the exception.
- Objects of type Throwable contain 2 items of information about an exception
  - A message that we pass as an argument when creating an exception object
  - A record of the execution stack at the time the object created

# Execution stack

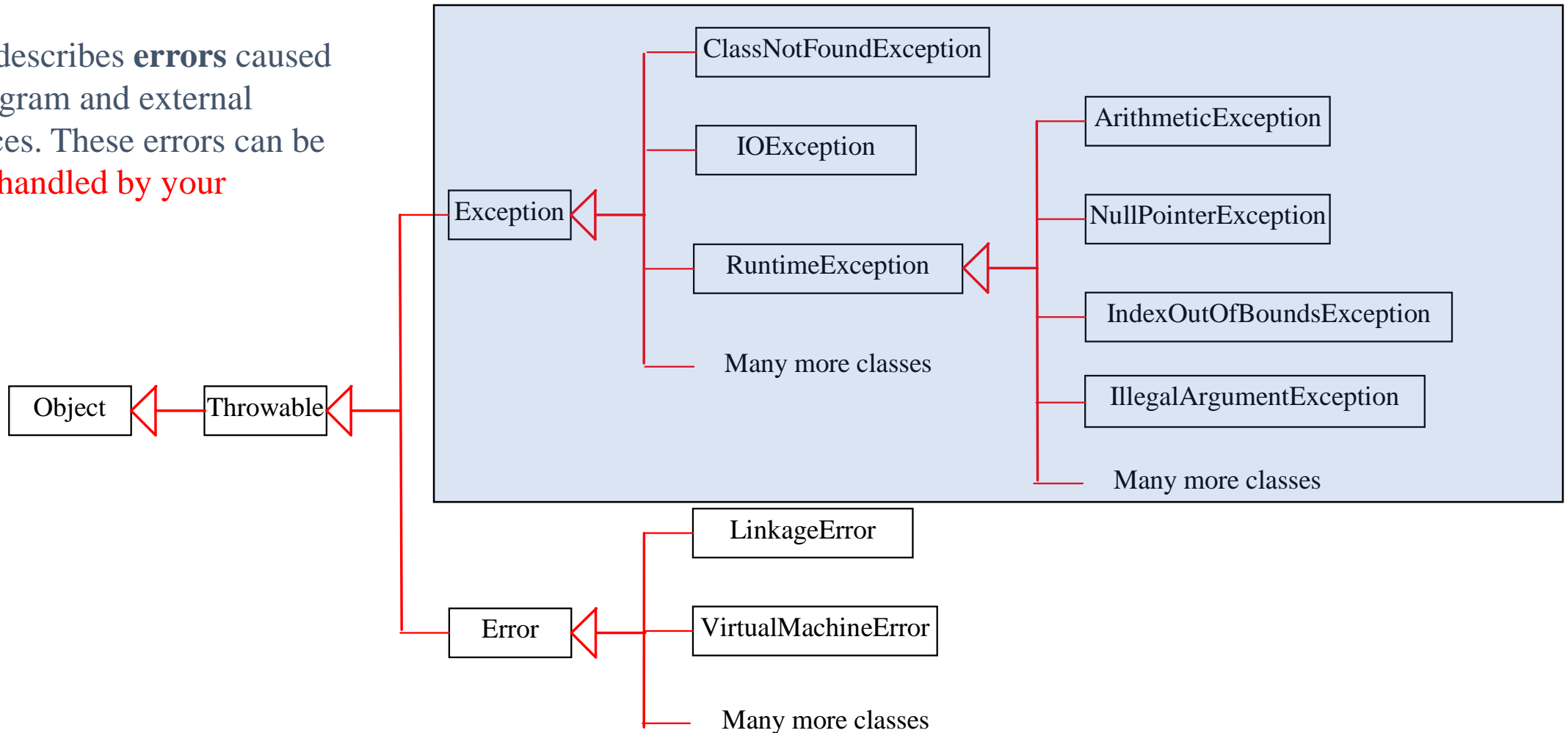
- Consists of
  - The line number in the source code where the exception originated
  - Followed by a trace of the method calls that immediately preceded the point at which the exception occurred
- The method calls are in sequence with the most recent method call appearing first

# Some methods of Throwable class

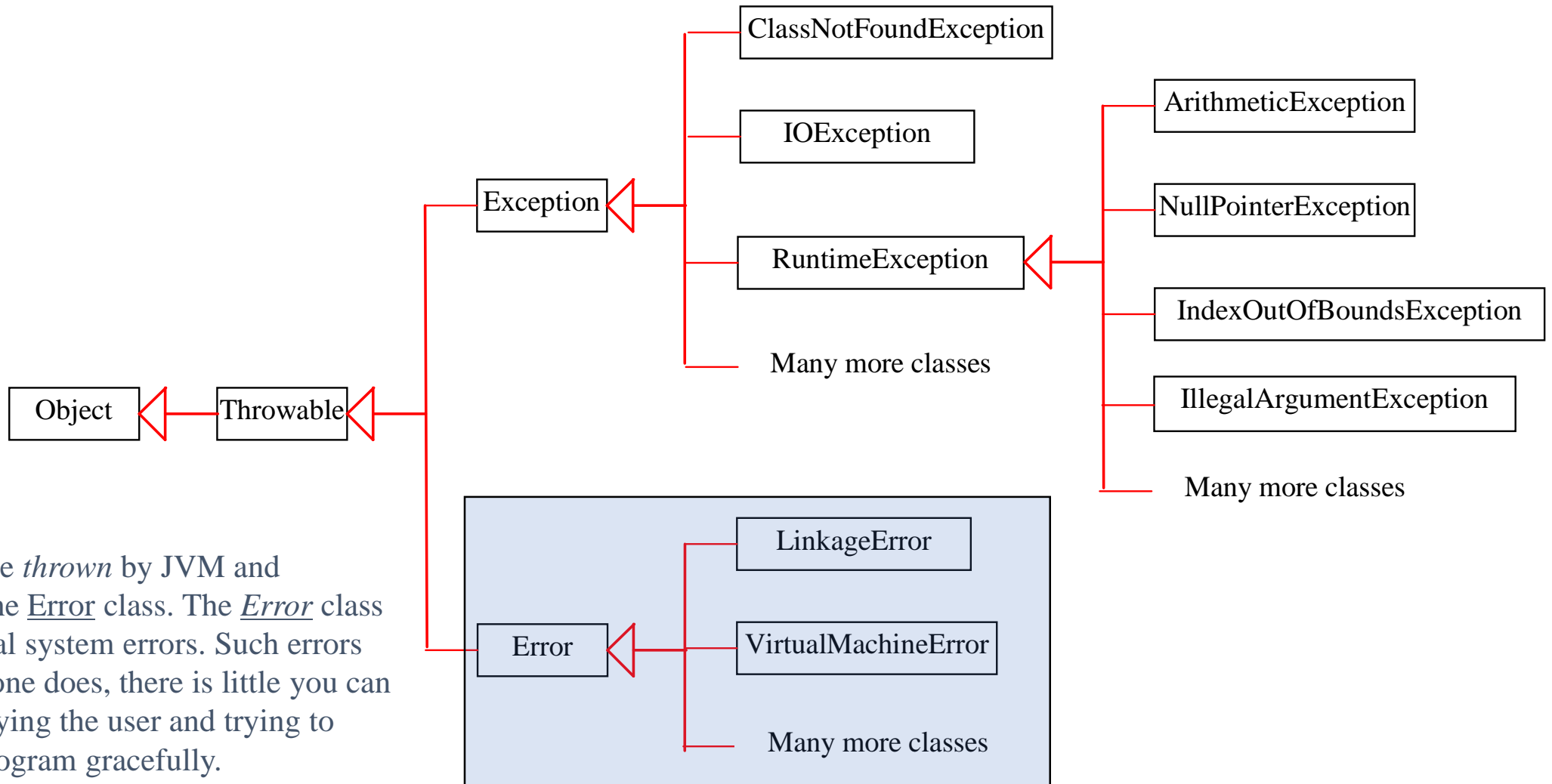
Method	Description
toString()	Return the package of exception followed by the message that describes the exception
getMessage()	Return the message that describes the exception
printStackTrace()	Output the message and the stack trace to the screen

# Exceptions

**Exception** describes **errors** caused by your program and external circumstances. These errors can be **caught and handled by your program**.



# System Errors





# Difference between error & exception

**1. Errors** indicate serious problems and abnormal conditions that most applications should not try to handle.

- Error defines problems that are not expected to be caught under normal circumstances by program. For example memory error, hardware error, JVM error etc.

**2. Exceptions** are conditions within the code.

- A developer can handle such conditions and take necessary corrective actions. For examples – DivideByZero exception, NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException

# Types of exceptions

- **Checked exception:**

- The compiler forces the programmer to check and deal with the exceptions
- Due to external circumstances that the programmer cannot prevent
- Majority occur when dealing with input and output
- Extend class ***IOException = I/O operation*** ไม่สามารถทำงานได้อย่างสมบูรณ์  
***EOFException = พบรหัส end of file*** ก่อน  
***FileNotFoundException = ไม่พบชื่อไฟล์ที่ระบุ***

# Types of exceptions

- ***Unchecked exception:***

- They are the programmer's or system's fault
- Extend class ***RuntimeException*** or ***Error***

- Examples of runtime exceptions:

**NumberFormatException** การกำหนดรูปแบบของตัวเลขไม่ถูก

**IllegalArgumentException** การส่งค่าให้เมทอดไม่ตรงข้อกำหนด

**NegativeArraySizeException** การสร้างอาร์เรย์ขนาดติดลบ

**IndexOutOfBoundsException** การอ้างอิงดัชนีที่ไม่อยู่ในช่วง

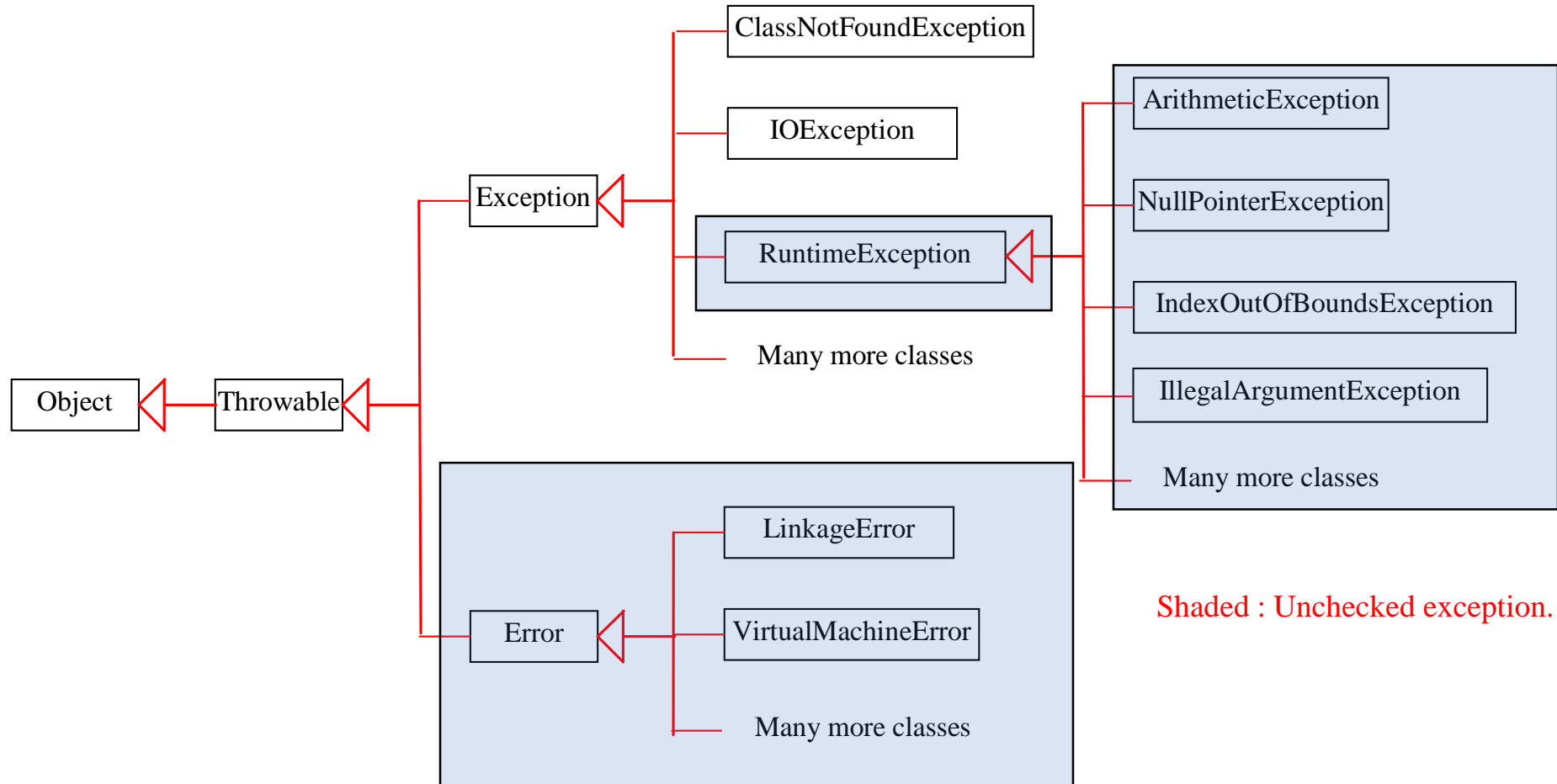
- Example of error:

**OutOfMemoryError** หน่วยความจำถูกใช้จนไม่พอให้ทำงานได้

**IOException** อุปกรณ์รับส่งข้อมูลผิดพลาด

**NoClassDefFoundError** ไม่พบคลาสที่ต้องการ

# Unchecked Exceptions



# Exception handling, two choices:

1. Handle the exception
2. Not handle the exception

If it is a **checked** exception, **must declare** the exception : Tell compiler that you want method to be terminated when the exception occurs (declare with throws clause)

```
public void read(String filename) throws FileNotFoundException {  
    FileReader reader = new FileReader(filename);  
    Scanner in = new Scanner(reader);  
    ...  
}
```

- For multiple exceptions:

```
public void read(String filename) throws IOException, ClassNotFoundException
```

- Keep in mind inheritance hierarchy: If method can throw an IOException and FileNotFoundException, **only use IOException**

# try statement

```
try {  
    ...           // มี try block ได้แค่ 1 block เท่านั้น  
}  
  
catch ( ... ) {  
    ...           // ไม่มีเลย หรือมีได้ 1 หรือ มากกว่า 1 block ก็ได้  
}  
  
finally {  
    ...           // ไม่มีเลย หรือมีได้เพียง 1 block เท่านั้น  
}
```

แต่ try แล้วต้องมี catch หรือ try แล้วต้องมี finally

สรุป นอกจาก **try-catch-finally** ครบชุด

อาจเป็น **try-catch** หรือ **try-finally** ก็ได้

## จัดการ (try-catch)

```
import java.util.Scanner;
public class Quotient {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        try {
            // Prompt the user to enter two integers
            System.out.print("Enter two integers: ");
            int number1 = input.nextInt();
            int number2 = input.nextInt();
            System.out.println(number1 + " / " + number2 + " is " + (number1 / number2));
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

```
run:
Enter two integers: 5
0
java.lang.ArithmeticException: / by zero
BUILD SUCCESSFUL (total time: 8 seconds)
```

```
import java.util.Scanner;
```

```
public class Quotient {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        try {
```

```
            // Prompt the user to enter two integers
```

```
            System.out.print("Enter two integers: ");
```

```
            int number1 = input.nextInt();
```

```
            int number2 = input.nextInt();
```

```
            System.out.println(number1 + " / " + number2 + " is " + (number1 / number2));
```

```
        }
```

```
        finally {
```

```
            System.out.println("Hello");
```

```
        }
```

```
    }
```

```
}
```

ไม่จัดการ (มี try ไม่มี catch)

run:

Enter two integers: 5

0

Hello

Exception in thread "main" java.lang.ArithmeticException: / by zero  
at quotient.Quotient.main(Quotient.java:22)

Java Result: 1

BUILD SUCCESSFUL (total time: 9 seconds)



```
import java.util.Scanner;
```

```
public class Quotient {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        try {
```

```
            // Prompt the user to enter two integers
```

```
            System.out.print("Enter two integers: ");
```

```
            int number1 = input.nextInt();
```

```
            int number2 = input.nextInt();
```

```
            System.out.println(number1 + " / " + number2 + " is " + (number1 / number2));
```

```
        }
```

```
        catch (Exception e) {
```

```
            System.out.println(e);
```

```
        }
```

```
    finally {
```

```
        System.out.println("Hello");
```

```
    }
```

```
}
```

```
}
```

จัดการ (try-catch)

run:

Enter two integers: 5

0

java.lang.ArithmeticException: / by zero

Hello

**BUILD SUCCESSFUL** (total time: 22 seconds).

# try statement

- Code that could generate errors put in `try` block
- Code for error handling enclosed in a `catch` block (with an exception type specified as a parameter)
- The `finally` block **always executes**
- Termination model of exception handling
  - The block in which the exception occurs **expires**

Execution  
starts →

```
try {  
    //code that can throw exceptions  
}
```

```
catch (Exception1 e) {  
    //code to process exception  
}
```

```
catch (Exception2 e) {  
    //code to process exception  
}
```

```
finally {  
    //code to execute after try block  
}
```

```
//other statements
```

Normal execution

Execution  
starts →

```
try {  
    //code that can throw exceptions  
}
```

```
catch (Exception1 e) {  
    //code to process exception  
}
```

```
catch (Exception2 e) {  
    //code to process exception  
}
```

```
finally {  
    //code to execute after try block  
}
```

```
//other statements
```

Exception occurs

```

class Nest{
    public static void main(String args[]){
        //Parent try block
        try{
            //Child try block1
            try{
                //try-catch block inside another try block
                System.out.println("Inside block1");
                int b =45/0;
                System.out.println(b);
            }
            catch(ArithmeticException e1){
                //Exception Message
                System.out.println("Exception: e1");
            }
            //Child try block2
            try{
                //try-catch block inside another try block
                System.out.println("Inside block2");
                int b =45/0;
                System.out.println(b);
            }
            catch(ArrayIndexOutOfBoundsException e2){
                //Exception Message
                System.out.println("Exception: e2");
            }
            System.out.println("Just other statement");
        }
        catch(ArithmeticException e3){ //Catch of Main(parent) try block
            //Exception Message
            System.out.println("Arithmetic Exception");
            System.out.println("Inside parent try catch block");
        }
        catch(ArrayIndexOutOfBoundsException e4){
            System.out.println("ArrayIndexOutOfBoundsException");
            System.out.println("Inside parent try catch block");
        }
        catch(Exception e5){
            System.out.println("Exception");
            System.out.println("Inside parent try catch block");
        }
        System.out.println("Next statement..");
    }
}

```

**Flow of try catch block**  
 program can also  
 contain **nested**  
**try-catch-finally blocks.**

```

----jGRASP exec: java Nest

Inside block1
Exception: e1
Inside block2
Arithmetic Exception
Inside parent try catch block
Next statement..

----jGRASP: operation complete.

```

# Declaring Exceptions

- Every method must state the types of checked exceptions it might throw. This is known as *declaring exceptions*.

```
public void myMethod() throws IOException
```

```
public void myMethod() throws IOException, OtherException
```

(declare)

```
public class ReadData {  
    public static void main(String[] args) throws Exception  
    { // Create a File instance  
        java.io.File file = new java.io.File("scores.txt");  
        // Create a Scanner for the file Scanner  
        input = new Scanner(file);  
        // Read data from a file  
        while (input.hasNext()) {  
            String firstName = input.next();  
            String mi = input.next();  
            String lastName = input.next();  
            int score = input.nextInt();  
            System.out.println( firstName + " " + mi + " " +  
lastName + " " + score);  
        }  
        // Close the file  
        input.close();  
    }  
}
```

# สรุป

- **Exception** คือข้อผิดพลาดที่อาจเกิดขึ้นในโปรแกรม แบ่งเป็น
  - **checked exc. compiler** จะตรวจตั้งแต่ **compile program** ว่าเราเขียนโค้ดจัดการหรือยัง (**handle**) หรือเราบอกหรือยังว่าเมื่อบุคคลนั้นอาจโยน **exception** ออกมาได้ (**declare**) ถ้ายังจะ **compile** ไม่ผ่าน
  - **unchecked exc.** ความผิดพลาดที่เราไม่ต้องจัดการก็ได้ **compiler** ไม่สนใจ
- การจัดการกับความผิดพลาด (จากตัวอย่างที่แสดง)
  - จัดการเอง (**catch**) โปรแกรมก็จะจบสวยงามตามที่เราสั่ง
  - ไม่จัดการเอง (ไม่ **catch**) โปรแกรมก็จะจบด้วยการที่ **system** จัดการให้ ซึ่งถ้าเป็น **checked exception** จะต้องเขียน **declare** เอาไว้ให้ด้วยที่หัวเมธอดของโค้ดที่มีโอกาสจะเกิด **exception** ด้วย
- ที่กำลังจะเรียนต่อไปคือ ไม่จัดการเอง แต่โยนต่อให้คนอื่นจัดการให้ ด้วยคำสั่ง  
**throw new exception-object**



# Throwing exceptions

- Throw by system
- Throw by programmer
  - Programmer can throw exception in program by
    1. Write throw statement in try block
    2. Write throw statement in method
- Exception objects are from Java exception classes or user-defined exception classes

# throw by programmer

- To throw exceptions, we use throw statement  
*throw new ExceptionClass();*

throw new ArithmeticException();

Do not add any exception description

OR

*throw new ExceptionClass("-----");*

throw new ArithmeticException("I code my  
program to throw Divide by zero");

Adding an exception description → **should do**

# Example of throw in try block

```
import java.util.Scanner;

public class Quotient {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        try {

            System.out.print("Enter two integers: ");           // Prompt the user to enter two integers

            int number1 = input.nextInt();

            int number2 = input.nextInt();

            if (number2 == 0)

                throw new ArithmeticException("I code my program to throw Divide by Zero");

            System.out.println(number1 + " / " + number2 + " is " + (number1 / number2));

        }

        catch (ArithmeticException e) {

            System.out.println(e);

        }

    }

}
```

```
run:
Enter two integers: 5
0
java.lang.ArithmeticException: I code my program to throw Divide by Zero
BUILD SUCCESSFUL (total time: 3 seconds)
```

# Example of throw in method

```
import java.util.Scanner;

public class Quotient {

    public static int quotient_met( int numerator, int denominator ) throws ArithmeticException { //หรือไม่มีก็ได้ เพราะไม่เป็น checked exc.

        if (denominator == 0)

            throw new ArithmeticException("I check it to throw error");

        return numerator / denominator;

    }

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        try {

            System.out.print("Enter two integers: "); // Prompt the user to enter two integers

            int number1 = input.nextInt();

            int number2 = input.nextInt();

            System.out.println(number1 + " / " + number2 + " is " + quotient_met(number1, number2));

        }

        catch (ArithmeticException e) {

            System.out.println(e);

        }

    }

}
```

```
import java.util.Scanner;

public class Quotient {

    public static int quotient_met( int numerator, int denominator ) {    throws ArithmeticException //หรือไม่มีก็ได้ เพราะไม่เป็น checked exc.

        if (denominator == 0)

            throw new ArithmeticException("I check it to throw error");

        return numerator / denominator;

    }

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        boolean continueLoop = true;

        do {

            try {

                System.out.print("Enter two integers: ");          // Prompt the user to enter two integers

                int number1 = input.nextInt();

                int number2 = input.nextInt();

                System.out.println(number1 + " / " + number2 + " is " + quotient_met(number1, number2));

                continueLoop = false;

            }

            catch (ArithmeticException e) {

                System.out.println(e);

            }

        } while (continueLoop);

    }

}
```

Add do-while loop

```
run:
Enter two integers: 5
0
java.lang.ArithmeticException: I check it to throw error
Enter two integers: 2
1
2 / 1 is 2
BUILD SUCCESSFUL (total time: 15 seconds)
```

```

public class CircleWithException {
    /** The radius of the circle */
    private double radius;
    /** The number of the objects created */
    private static int numberOfObjects = 0;
    /** Construct a circle with radius 1 */
    public CircleWithException() {
        this(1.0);
    }
    /** Construct a circle with a specified radius */
    public CircleWithException(double newRadius) {
        setRadius(newRadius);
        numberOfObjects++;
    }
    /** Return radius */
    public double getRadius() {
        return radius;
    }
    /** Set a new radius */
    public void setRadius(double newRadius)
        throws IllegalArgumentException {
        if (newRadius >= 0)
            radius = newRadius;
        else
            throw new IllegalArgumentException(
                "Radius cannot be negative");
    }
    /** Return numberOfObjects */
    public static int getNumberOfObjects() {
        return numberOfObjects;
    }
    /** Return the area of this circle */
    public double findArea() {
        return radius * radius * 3.14159;
    }
}

```

```

public class TestCircleWithException {
    public static void main(String[] args) {
        try {
            CircleWithException c1 = new CircleWithException(5);
            CircleWithException c2 = new CircleWithException(-5);
            CircleWithException c3 = new CircleWithException(0);
        }
        catch (IllegalArgumentException ex) {
            System.out.println(ex);
        }

        System.out.println("Number of objects created: " +
            CircleWithException.getNumberOfObjects());
    }
}

```

run:

java.lang.IllegalArgumentException: Radius cannot be negative

Number of objects created: 1

**BUILD SUCCESSFUL (total time: 15 seconds)**

# When to Throw Exceptions

- An exception occurs in a method.
- **If you want the exception to be processed by its caller,**
  - Then you should create an exception object and **throw** it.
  - If you can handle the exception in the method where it occurs, **there is no need to throw it.**

# When to Use Exceptions

When should you use the try-catch block in the code? You should use it to deal with unexpected error conditions. Do not use it to deal with simple, expected situations. For example, the following code

```
try {  
    System.out.println(refVar.toString());  
}  
  
catch (NullPointerException ex) {  
    System.out.println("refVar is null");  
}
```



# When to Use Exceptions

is better to be replaced by

```
if (refVar != null)
    System.out.println(refVar.toString());
else
    System.out.println("refVar is null");
```

# Using several catch blocks

- If you put  
    `catch (Exception e) { ... }`  
    at the beginning of catch blocks

- What will happen??

```
    catch (Exception e)    {  
        System.out.println ("exception occurs!!!");  
    }  
    catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println ("out of array range!!!");  
    }  
    catch (ArithmeticException e) {  
        System.out.println ("cannot div by zero!!!");  
    }
```

# Compilation error

----- javac -----

HandlingArray.java:26: exception  
java.lang.ArrayIndexOutOfBoundsException has already been  
caught

catch (ArrayIndexOutOfBoundsException e)  
^

HandlingArray.java:29: exception  
java.lang.ArithmeticException has already been caught

catch (ArithmeticException e)  
^

2 errors

# multi-catch

- Handling multiple exceptions in one catch if the bodies of several catch blocks are identical

```
catch (Type1 | Type2 | Type3 e) {  
    statement;  
    ...  
}
```

```
catch (ArrayIndexOutOfBoundsException | ArithmeticException e) {  
    System.out.println ("Exception occurs!!!");  
}
```

# Exception propagation

- If an exception occurs at a statement but we do not catch it, it will propagate to the outer block
- If there is no exception handling, it will continue to propagate to the method
- If there is still no exception handling, it will propagate to the calling method, ... until main method... and finally, java interpreter will handle it by reporting the exception and stopping the program

# User-defined exception

- Two reasons to define a new exception class
  - Want to add more information when the standard exception occurs
  - The new exception is different from the standard exceptions
- The new exception class must extends
  - Exception or its subclass for checked exception
  - RuntimeException or its subclass for unchecked exception

# Designing Your Own Exception Types

- ```
if (amount > balance)
{
    throw new InsufficientFundsException(
        "withdrawal of " + amount + " exceeds balance of "
        + balance);
}
```

```
public class InsufficientFundsException extends RuntimeException
{
    public InsufficientFundsException() {}

    public InsufficientFundsException(String message)
    {
        super(message);
    }
}
```

# Exercise

```
public class BankAccount {
    private double balance;
    private String owner;
    public BankAccount() {
        balance = 0;
    }
    public BankAccount(String name,double initialBalance) {
        balance = initialBalance;
        owner = name;
    }
    public void deposit(double amount) {
        balance = balance + amount;
    }
    public void withdraw(double amount) {
        balance = balance - amount;
    }
    public double getBalance() {
        return balance;
    }
    public String getOwner() {
        return owner;
    }
}
```

กำหนดคลาส **BankAccount** และ **BankAccountTester** ซึ่งมี main method ด้านล่าง  
ให้เพิ่มคำสั่ง **try statement** และ **throw IllegalArgumentException statement**  
ในคลาส **BankAccount** เพื่อจัดการกับข้อผิดพลาดที่อาจเกิดในคลาส **BankAccount**  
ตามกรณีด้านล่างนี้

1. สร้าง **account** ด้วยค่าเงินเริ่มต้นติดลบ
2. ฝากเงินด้วยค่าติดลบ
3. ถอนเงินเกินกว่าเงินที่มีอยู่

ในการตรวจจับข้อผิดพลาด เมื่อพบข้อผิดพลาด ให้แสดงข้อความระบุข้อผิดพลาดออกทางจอภาพดังตัวอย่าง  
ของผลลัพธ์โปรแกรมที่แสดงต่อไปนี้

**Initial Balance of -10.0 cannot be negative.**

**Deposit of -500.0 cannot be negative.**

**Withdrawal of 300.0 exceeds balance of 100.0**

**B has 100.0**

```
public static void main(String[] args) {
    BankAccount a=new BankAccount("A",-10);
    BankAccount b=new BankAccount("B",100);
    b.deposit(-500);
    b.withdraw(300);
    System.out.println(b.getOwner()+" has "+b.getBalance());
}
```



# References

- Deitel, H.M., and Deitel, P.J., *Java How to Program*, ninth edition, Prentice Hall, 2012.
- Horstmann, C., *Big Java*, John Wiley & Sons, 2009.
- Horton, I., *Ivor Horton's Beginning Java™ 2, JDK™ 5 Edition*, Wiley Publishing, Inc., 2005.
- Liang, Y. D., *Introduction to Java Programming*, tenth edition, Pearson Education Inc, 2015.