- Chapter 7

# Exception Handling

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

# Chapter 7:

# Exceptions Handling

# Objectives

- *Define an exception*
- *Explain exception handling*
- *Describe the* `try, catch`*, and* `finally` *blocks*
- *Examine multiple catch blocks*
- *Explore nested* `try / catch` *blocks*
- *Explain the use of* `throw` *and* `throws` *keywords*
- *Create user defined exceptions*
- *Learn usage of* `Assert` *statement*

# What is an exception?

- If an error is encountered while executing a program, an exception occurs.

example : 4 / 0 =


- When an exception occurs program terminates abruptly and control returns to operating system.
- Exception handling is performed to identify errors and trap them.

# Handling exceptions – General example

- Snippet of a pseudocode follows:

```
……
IF B IS ZERO GO TO ERROR
C = A/B
PRINT C
GO TO EXIT

ERROR:
DISPLAY "DIVISION BY ZERO"

EXIT:
END
```

Block that handles error

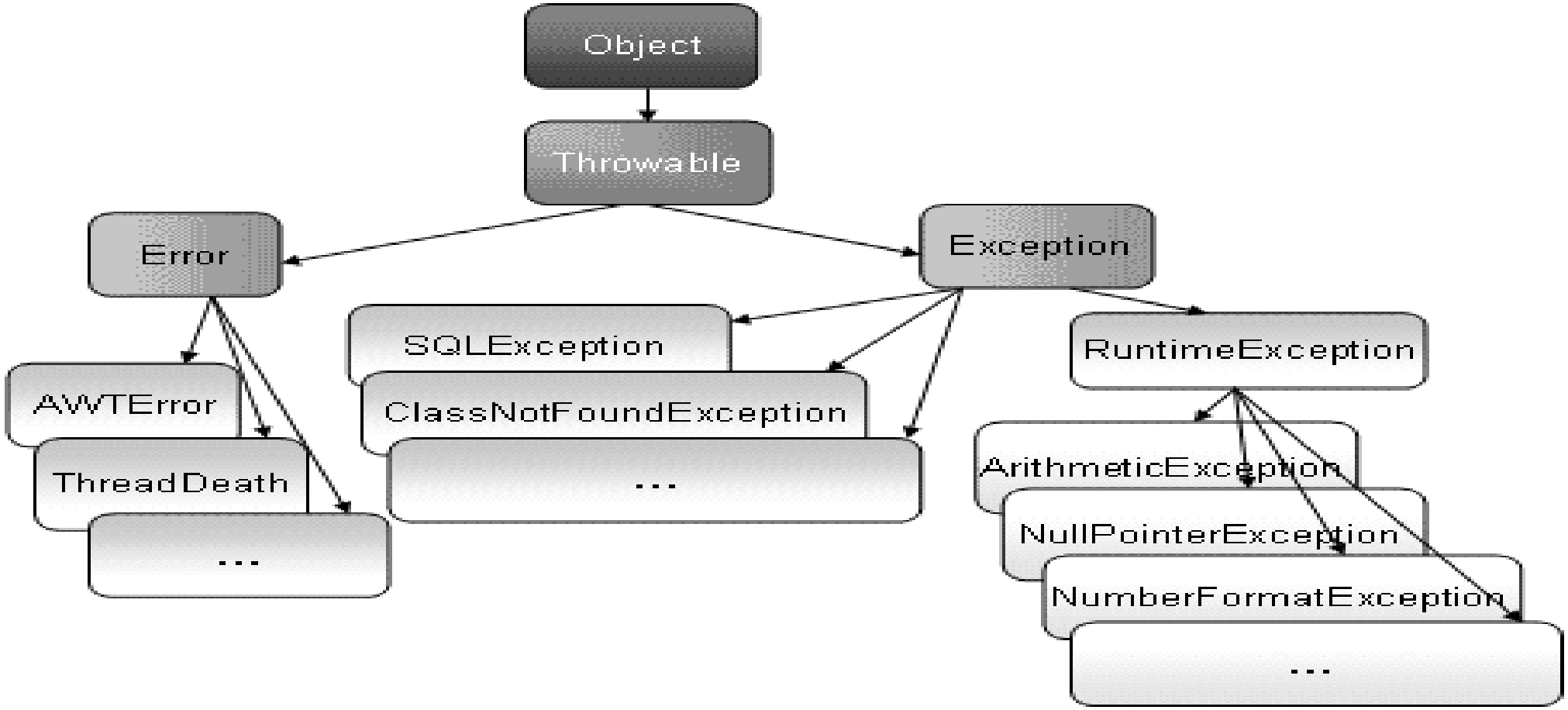
# Handling exceptions in Java

- Error handling in Java is performed with the help of an exception-handling model.
- Occurrence of an error causes an exception to be thrown, which is caught in a block.
- `Error` and `Exception` classes handle errors in Java.
- When an exception occurs, an instance representing that exception is created. The instance is then passed to the method that retrieves and processes the information.
- `Error` class exceptions are internal errors
  - e.g. – reading a file from a floppy without inserting it in the floppy drive

# Handling exceptions in Java Contd…

- Java exception handling is managed through five keywords: `try`, `catch`, `throw`, `throws`, and `finally`.
- Program statements, that have to be monitored for exceptions, are contained within a `try` block.
- By using `catch` keyword, the programmer can catch the exception and handle it in some rational manner.
- To manually throw an exception, we use the keyword `throw`.
- `Throws` clause is used in a method to specify that this method will throw out an exception.
- In the `finally` block, we can specify the code that is absolutely necessary to be executed before a method returns.

# Hierarchy of Exception Classes



# Exception handling model

- Handled by five keywords – `try`, `catch`, `throw`, `throws`, and `finally`
- Two ways to handle exceptions in Java:
  - Statements that may throw exceptions are in the `try` block and exception handling statements in the `catch` block
  - A method may be declared in such a way that in case an exception occurs, its execution is abandoned

# try and catch block - Example

```
class ExceptionDemo
{
    public static void rgs[])
    {
        try
        {
            String
            System
        }
        catch(Exc
        {
            System
        }
    }
}
```

Output

```
C:\WINDOWS\system32\cmd.exe

E:\Java\JavaExamples>javac ExceptionDemo.java

E:\Java\JavaExamples>java ExceptionDemo
No arguments given!
```

# A sample program to demonstrate exceptions

```
public class StckExceptionDemo
{
    public static void main(String args[])
    {
        try
        {
            int num =
            System.o
        }
        catch(Excep
        {
            System.e
            e.printSta
        }
    }

    static int calculate( int no, int no1)
    {
        int num = no / no1;
        return num;
    }
}
```

Output

```
C:\WINDOWS\system32\cmd.exe

E:\Java\JavaExamples>javac StckExceptionDemo.java

E:\Java\JavaExamples>java StckExceptionDemo
Exception occurred  :java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException: / by zero
        at StckExceptionDemo.calculate(StckExceptionDemo.java:19)
        at StckExceptionDemo.main(StckExceptionDemo.java:7)
```

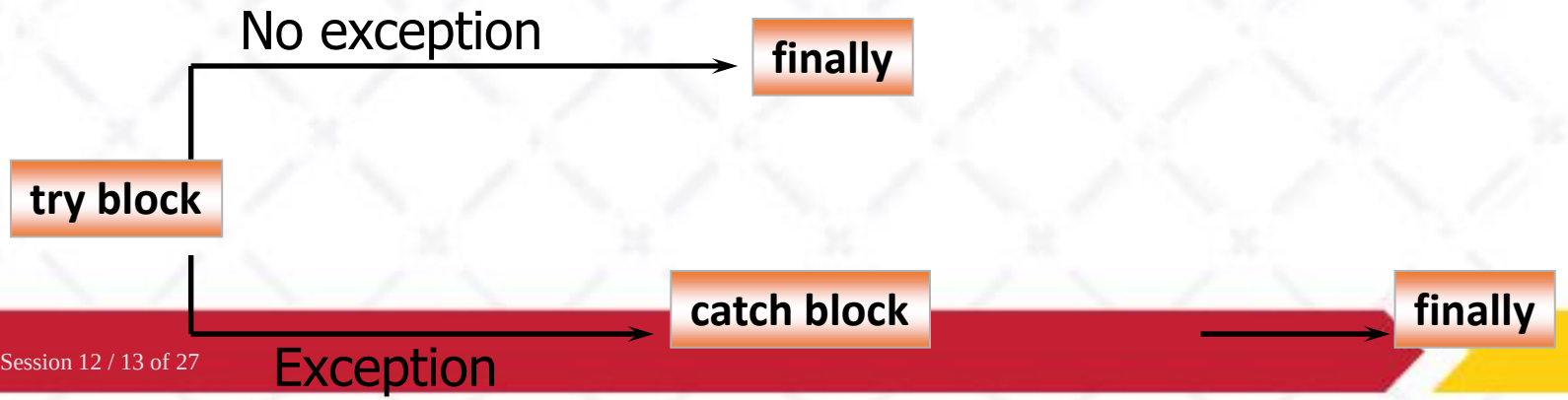
# Methods used in the example

- **`toString()`** retrieves a `String` representation of the information stored in an `Exception` object
- **`printStackTrace()`** is used to print out the cause of exception and also the line in the code which generated it

# finally block

- Ensures that all cleanup work is taken care of when an exception occurs

- Used in conjunction with a `try` block
- Guaranteed to run whether or not an exception occurs
- An exception if thrown will run the `finally` block even if there is no matching `catch` block

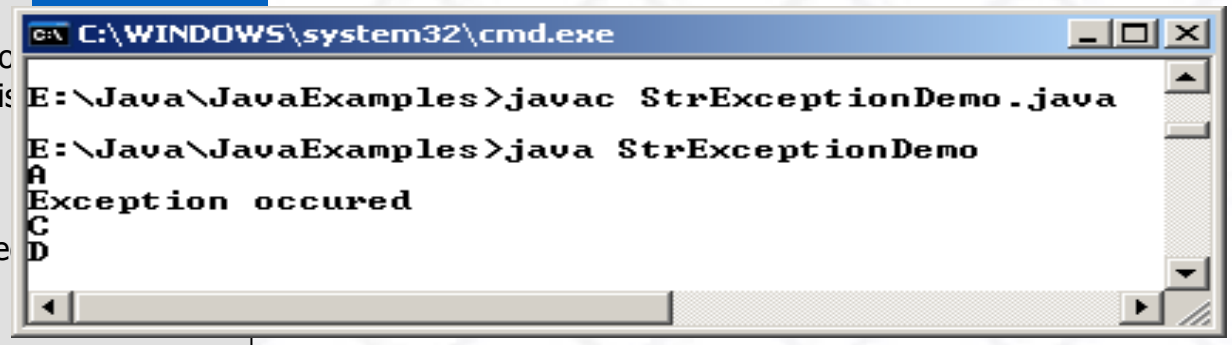


# finally block - Example

```
class StrExceptionDemo
{
    static String str;
    public static void main(String s[])
    {
        try
        {
                System.out.println("A");
                StrExceptionDemo.staticLengthmetho
                System.out.println("B"); //this code is
         }
            catch(NullPointerException ne)
         {
                System.out.println("Exception occure
                 System.out.println("C");
        }
         finally
        {
                System.out.println("D");
        }
    }

    static void staticLengthmethod()
    {
            System.out.println(str.length());
    }
}
```

Output



# Multiple catch blocks

- Single piece of code can generate more than one error.
- Hence, multiple catch blocks may have to be provided.
- Each `catch` statement has an order of appearance.

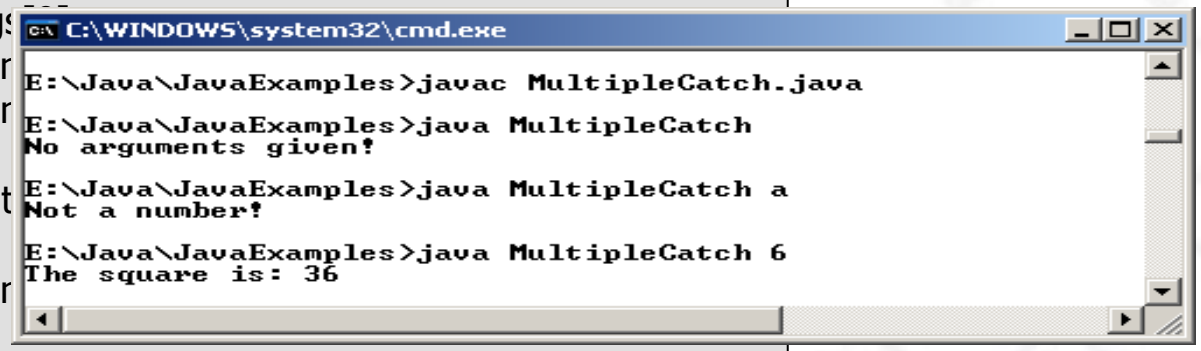
```
..........
try{
}
catch(ArrayIndexOutOfBoundsException e) {
}
catch(Exception e) {
}
..........
```

- `ArrayIndexOutOfBoundsException` being a subclass of `Exception` class is handled first.

# Example

```
class MultipleCatch
{
    public static void main(String args[])
    {
        try
        {
            String num = arg
            int numValue = Ir
            System.out.printlr
        }
        catch(ArrayIndexOut
        {
            System.out.printlr
        }
        catch(NumberFormatException nb)
        {
            System.out.println("Not a number!");
        }
    }
}
```

Output



# Nested try - catch blocks

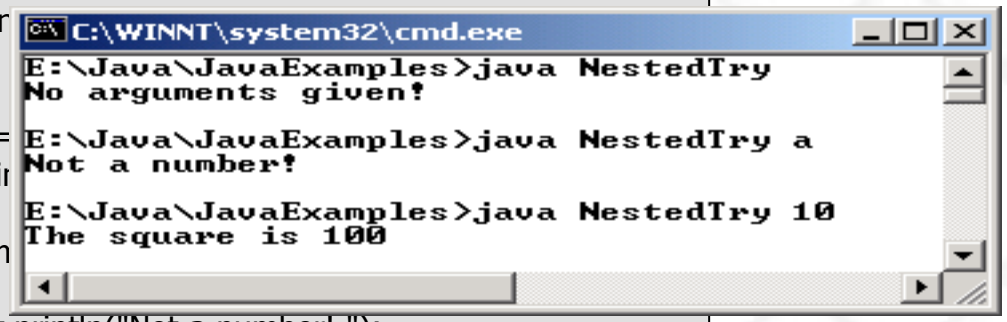
- At times, a part of one block may cause an error and the entire block may cause another error.

- In such a case, exception handlers have to be nested.
- When nested `try` blocks are used, inner `try` block is executed first.
- If no matching `catch` block is encountered, `catch` blocks of outer try blocks are inspected until all nested `try` statements are executed.

# Example

```
class NestedTry
{
    public static void main(String args[])
    {
        try
        {
            int num = args.ler
            try
            {
                int numValue =
                System.out.prir
            }
            catch(NumberForn
            {
                System.out.println("Not a number! ");
            }
        }
        catch(ArrayIndexOutOfBoundsException ne)
        {
            System.out.println("No arguments given! ");
        }
    }
}
```

Output



# Using throw & throws

- Exceptions are thrown using `throw` keyword.

```
try
{
        if(flag<0)
        {
                throw new NullPointerException();
        }
}
```

- A single method may throw more than one exception.

# Using throws

```
public class Example
{
    public void exceptionExample()
    {
        try
        {
            // statements
            check();
        }
    catch(Exception e)
    {
    //statements
    }
    }
// multiple exceptions separated by a comma
    void check() throws NullPointerException, NegativeArraySizeException
    {
        if (flag < 0)
            throw new NullPointerException();
        if(arrsize < 0)
            throw new NegativeArraySizeException();
  }
}
```

- check() method includes the `throws` keyword
- Such methods should be invoked within a `try / catch` blocks

# User defined exceptions

- Built-in exceptions may not always be enough to trap all the errors.
- Hence there is a need for user defined exception class.
- Should be a subclass of the `Exception` class
- The new exception type can be caught separately from other subclasses of `Throwable`.
- User defined exception classes that are created will have all methods of `Throwable` class.

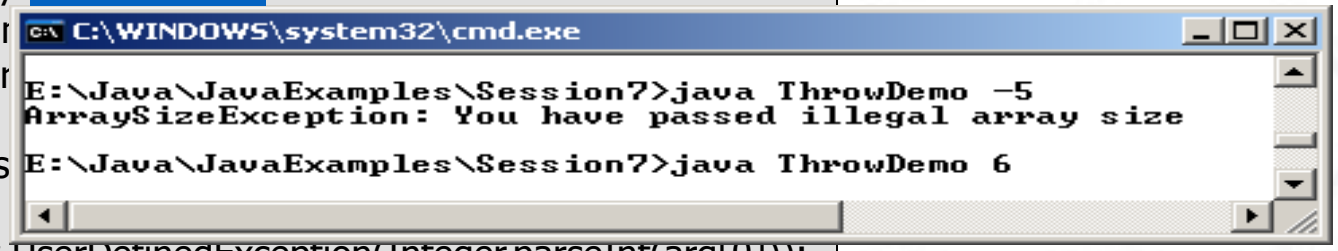
# Example

```
class ArraySizeException extends NegativeArraySizeException
{
    ArraySi
    {
        supe
    }
}
class User
{
    int size
    UserDe
    {
        size
        try
        {
            checkSize();
        }
        catch(ArraySizeException e)
{
            System.out.println(e);
        }
    }
```

```
void checkSize() throws ArraySizeException
    {
        if(size < 0)
            throw new ArraySizeException();

        array
        for(i
            ar
    }
    public s
    {
        new UserDefinedException(Integer.parseInt(arg[0]));
    }
}
```

Output



# Summary

- Whenever an error is encountered while executing a program, an `Exception` occurs.
- An `Exception` occurs at run time in a code sequence.
- Every exception that is thrown must be caught, or the application terminates abruptly.
- Exception handling allows combining error processing in one place.
- Java uses the `try` and `catch` block to handle exceptions.

# Summary Contd…

- The statements in the `try` block throw exceptions and the `catch` block handles them.
- Multiple `catch` blocks can be used together to process various exception types separately.
- The `throws` keyword is used to list the exception that a method can throw.
- The `throw` keyword is used to indicate that exception has occurred.
- The statements in the `finally` block are executed irrespective of whether an exception occurs or not.

# Programming with assertions

- Assertions enables the programmer to test assumptions about the program.
- Ways of writing assertion statements are:
  - `assert` expression;
  - `assert` *expression1:expression2;*
- The expression in the first form will have a `boolean` value and evaluate whether the expression is true and if it is false it will throw an `AssertionError`.
- The expression1 in the second form will be a boolean expression and expression2 can have any value, except that it cannot invoke a `void` method.
- If expression1 evaluates to false it will throw an `AssertionError`. The value in expression2 will be passed to `AssertionError` constructor and the value will be used to display the error message.

# Programming with assertions Contd…

- Situations for assertion statements
  - Internal invariants: assertion can be used wherever it asserts an invariant
  - Control Flow Invariants: an assertion can be placed at any location where control cannot be reached
  - Preconditions,post conditions and class invariants
- Command for compiling files that uses assertion statements is:
  - `javac -source 1.4 filename`