Advanced Programming

Exception Handling

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

Example

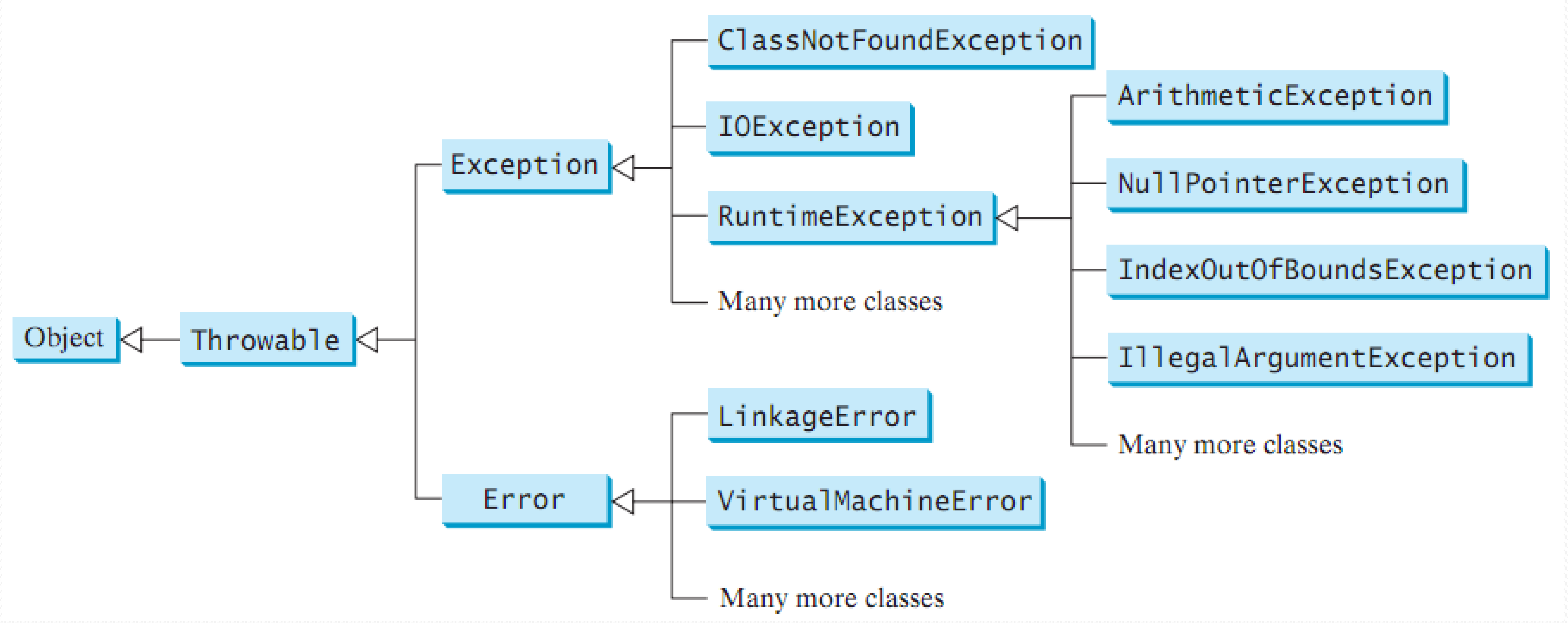
```
public class Quotient {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);

      // 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));
   }
}
```

```
public class QuotientWithException {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("Enter two integers: ");
      int number1 = input.nextInt();
      int number2 = input.nextInt();
      try {
            if (number2 == 0)
                throw new ArithmeticException("Divisor cannot
                         be zero");
            System.out.println(number1 + " / " + number2 +
                  "is " + (number1 / number2));
      } catch (ArithmeticException ex) {
            System.out.println("Exception: an integer "
                  + "cannot be divided by zero ");
      }
      System.out.println("Execution continues ...");
   }
}
```

Exception-Handling Overview

- The program contains a **try** block and a **catch** block.
  - The **try** block contains the code that is executed in normal circumstances.
  - The **catch** block contains the code that is executed when number2 is 0.
- The value thrown, in this case **new ArithmeticException ("Divisor cannot be zero")**, is called an **exception**.
- The execution of a throw statement is called *throwing an exception*. The **exception** is an object created from an **exception class**.
  - In this case, the exception class is java.lang.ArithmeticException.

Exception Types



Exceptions

- **ClassNotFoundException**: Attempt to use a class that does not exist.
  - if you tried to run a nonexistent class using the java command,
  - or if your program were composed of three class files, only two of which could be found.
- **IOException**: Related to input/output operations, such as invalid input, reading past the end of a file, and opening a nonexistent file.

Runtime exceptions

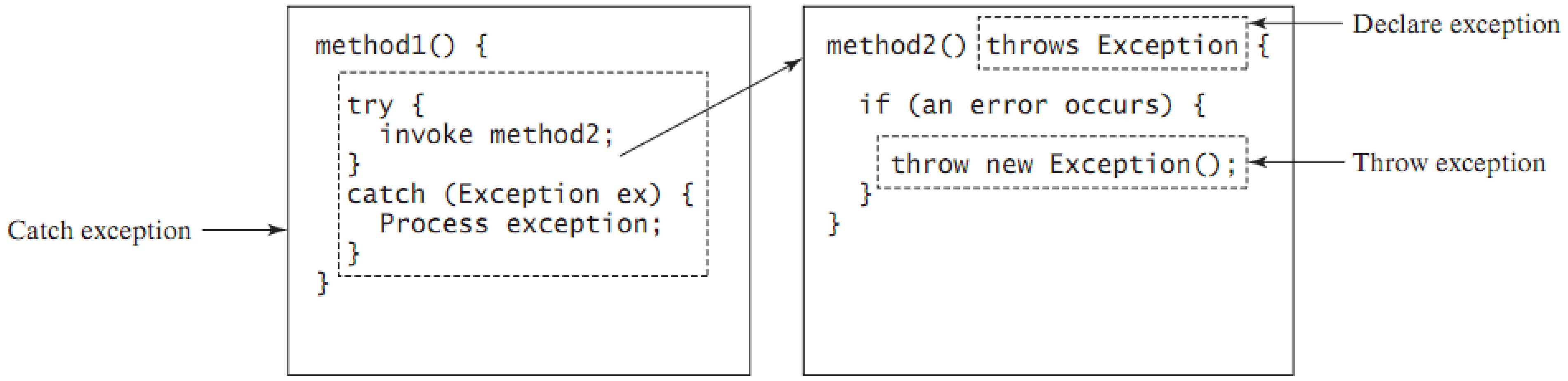
- **Runtime exceptions** are represented in the **RuntimeException** class, which describes programming errors, such as *bad casting*, *accessing* an out-of-bounds array, and *numeric* errors.
  - Runtime exceptions are generally thrown by the JVM

Runtime exceptions

- **ArithmeticException**: Dividing an integer by zero. Note that floating-point arithmetic does not throw exceptions.
- **NullPointerException**: Attempt to access an object through a null reference variable.
- **IndexOutOfBoundsException**: Index to an array is out of range.
- **IllegalArgumentException**: A method is passed an argument that is illegal or inappropriate.

More on Exception Handling

- Java's exception-handling model is based on three operations:
  - ***Declaring*** *an exception,*
  - ***Throwing*** *an exception,*
  - ***Catching*** *an exception.*



```
method1() {
  try {
    invoke method2;
  }
  catch (Exception ex) {
    Process exception;
  }
}
```

```
method2() throws Exception {
  if (an error occurs) {
    throw new Exception();
  }
}
```

Declaring Exceptions

- To declare an exception in a method, use the **throws** keyword in the method header:

  public void myMethod() **throws** IOException

- The **throws** keyword indicates that myMethod might throw an IOException.
- If the method might throw multiple exceptions, add a list of the exceptions, separated by commas, after **throws**:

  public void myMethod() **throws** Exception1, Exception2, ..., ExceptionN

Throwing Exceptions

- A program that detects an error can create an instance of an appropriate exception type and throw it.
- Example:
  - The program detects that an argument passed to the method violates the method contract (e.g., the argument must be nonnegative, but a negative argument is passed); T
  - The program can create an instance of **IllegalArgumentException** and throw it, as follows:

```
IllegalArgumentException ex =
  new IllegalArgumentException("Wrong Argument");
throw ex;
```

Catching Exceptions

- When an exception is thrown, it can be *caught* and *handled* in a **try-catch** block, as follows:

```
try {
  statements;  // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
```

Catching Exceptions

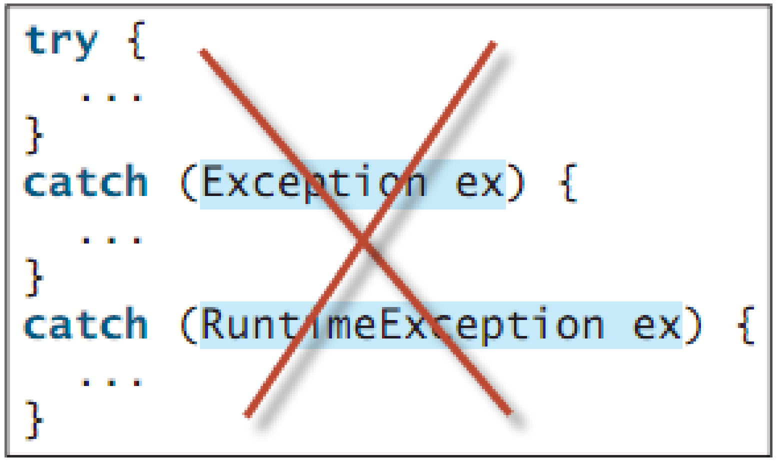
- If **no exceptions** arise during the execution of the try block, the **catch** blocks are skipped.
- If one of the statements inside the **try** block throws an exception, Java *skips the remaining* statements in the **try** block and starts the process of finding the code to handle the exception.
- The code that handles the exception is called the **exception handler.**

Catching Exceptions

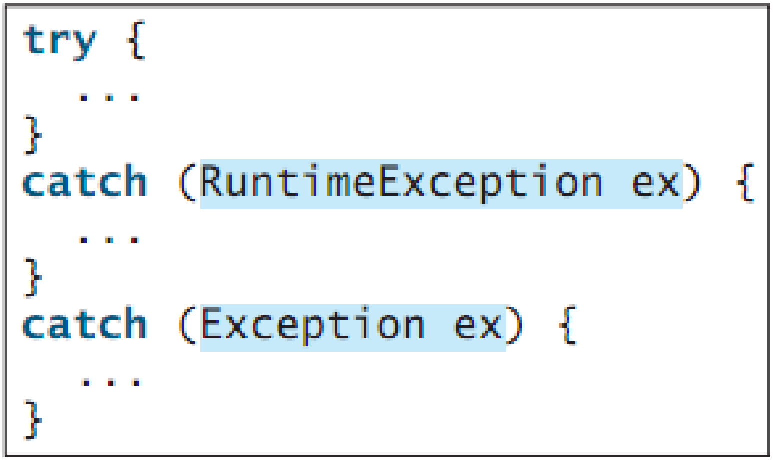
- Each **catch** block is examined in turn, from first to last, to see whether the type of the exception object is an instance of the exception class in the **catch** block.
  - The exception object is assigned to the variable declared, and the code in the **catch** block is executed.
  - If no handler is found, Java exits this method, passes the exception to the method that invoked the method, and continues the same process to find a handler.
  - If no handler is found in the chain of methods being invoked, the program terminates and prints an error message on the console.
- The process of finding a handler is called **catching an exception**.

Order of exception handlers

- The order in which exceptions are specified in **catch** blocks is important. A compile error will result if a **catch** block for a *superclass* type appears before a **catch** block for a *subclass* type.



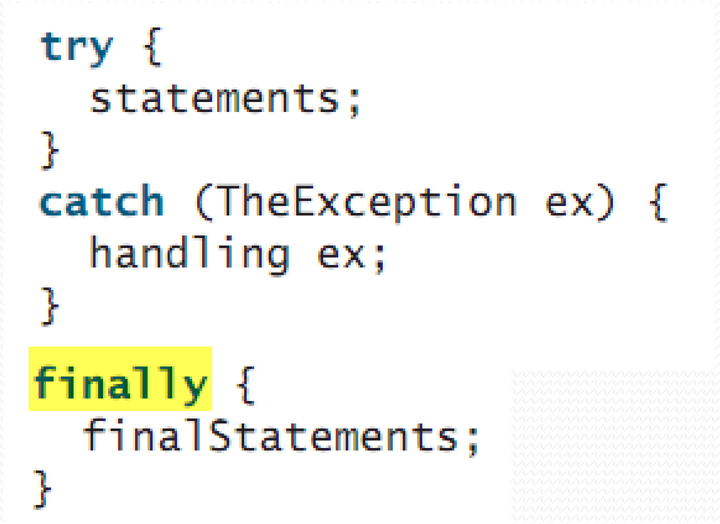
```
try {
   ...
}
catch (Exception ex) {
   ...
}
catch (RuntimeException ex) {
   ...
}
```



```
try {
   ...
}
catch (RuntimeException ex) {
   ...
}
catch (Exception ex) {
   ...
}
```

The finally clause

- You may want some code to be executed regardless of whether an exception occurs or is caught. Java has a **finally** clause that can be used to accomplish this objective.



```
try {
  statements;
}
catch (TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}
```

The finally clause

- If no exception arises in the **try** block, **finalStatements** is executed, and the next statement after the **try** statement is executed.
- If a statement causes an exception in the **try** block that is caught in a **catch** block,
  - the rest of statements in the **try** block are skipped,
  - the **catch** block is executed,
  - and the finally clause is executed.
  - The next statement after the **try** statement is executed.
- If one of the statements causes an exception that is not caught in any **catch** block:
  - the other statements in the **try** block are skipped,
  - the **finally** clause is executed,
  - and the exception is passed to the caller of this method.

Reference

- **Introduction to Java Programming** 8th , Y. Daniel Liang.