Java Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Online Course

```
1 class Lecture8 {
2
3             "Exceptions and Exception Handling"
4
5 }
6
7 // Keywords:
8 try, catch, finally, throw, throws, assert
```

Introduction

- An exceptions is an event which disrupts the normal flow of the program.[1]
  - For example, open a missing file.
- When an error occurs within a method, the method creates an exception object and hands it off to the runtime system.
- This is called throwing an exception.
- The runtime system searches the call stack for a method that contains a block of code that can handle the exception, called exception handler
- The exception handler chosen is said to catch the exception.

[1] Note that the exception should be a force majeure.

The Handling Blocks: try-catch-finally

- Now we proceed to introduce the three components of the exception handler: the try, catch, and finally blocks.
- First we put the normal operations which may throw exceptions in the try block.
- Then we write down the handlers for specific exceptions.[2]
  - You may consider a multi-catch (using | to separate them).[3]
  - Usually, we put the super-type **Exception** in the last catch clause to catch the exceptional exceptions.
- Java provides the finally block, which is always executed when the try block exits.
  - This block is mainly used for cleanup, say closing a file.

[2] Try to handle each exception but not once at all.

[3] The grouped exceptions in the same catch clause should be siblings.

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

public class ExceptionDemo {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        try {
            System.out.println("Enter an integer?");
            int x = input.nextInt();
        } catch (InputMismatchException e) {
            System.out.println("Not an integer.");
        } catch (Exception e) {
            System.out.println("Unknown exception.");
        } finally {
            input.close();
            System.out.println("Cleanup is done.");
        }

        System.out.println("End of program.");
    }

}
```

Exception Family[4]

- The topmost class of exception family is **Throwable**.
- All subtypes of **Throwable** could be categorized into two groups: unchecked exceptions and checked exceptions.
- Checked exceptions must be checked at compile time.
  - For example, **IOException** and **Exception**.
- Unchecked exceptions are not forced by the compiler to either handle or specify the exception.
  - For example, **RuntimeException**.

[4] See `https://www.programcreek.com/2009/02/diagram-for-hierarchy-of-exception-classes/`.

Throwing Exceptions

- As a library maker, we sometimes disallow the behaviors from the users.
- Java provides the throwing mechanism by using throw (issuing) and throws (translation).

```
1  public class Circle {
2
3      private double radius;
4
5      public Circle(double r) throws Exception {
6          if (r <= 0)
7              throw new Exception("r <= 0");
8          radius = r;
9      }
10
11 }
```

Customized Exceptions

- It is clear that we exploit the inheritance mechanism to create our own exception family.

```
1 public class InvalidRadiusException extends Exception {
2
3     public InvalidRadiusException(double r) {
4         super("Invalid radius: " + r); // Pass error message.
5     }
6
7 }
```

```java
public class Circle {

    private double radius;

    public Circle(double r) throws InvalidRadiusException {
        if (r <= 0)
            throw new InvalidRadiusException(r);
        radius = r;
    }

}
```

```java
public class NewExceptionDemo {

    public static void main(String[] args) {

        try {
            new Circle(-10); // Check the result!
        } catch (InvalidRadiusException e) {}

    }

}
```

Digress: Assertion

- An assertion is a statement that enables you to test your assumptions about the program, as an internal check.
- Before running the program, add "-ea" to the VM arguments so that these assertion statements can be tested.

```
1  public class AssertDemo {
2
3      public static void main(String[] args) {
4
5          int x = 1;
6          assert("x is not equal to 2.", x == 2);
7          // AssertionError occurs!!
8          System.out.println("End of program.");
9
10     }
11
12 }
```

Unit Test: JUnit

- However, we should avoid writing testing codes together with the normal codes!
- Try JUnit: `https://junit.org/`

Fin.