

Java Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Java 405
Spring 2024

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

Introduction


- An **exception** is to interrupt “normal” program flows.¹
 - For example, opening a non-existing file results in **FileNotFoundException**.
- When the callee **throws an exception object**, this object should be well-handled by the caller, by providing proper **exception handlers**.
- In other words, a specific exception handler **catches** the associated exception.

¹Note that an exception should be a force majeure event. 

The Handling Block: try-catch-finally

- Now we proceed to introduce the three components of exception handlers: the **try**, **catch**, and **finally** blocks.
- First, wrap the normal operations which may throw exceptions in the **try** block.
- We then 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.

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

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

```
1 import java.util.Scanner;
2 import java.util.InputMismatchException;
3
4 public class ExceptionDemo {
5
6     public static void main(String[] args) {
7
8         Scanner input = new Scanner(System.in);
9
10        try {
11            System.out.println("Enter an integer?");
12            int x = input.nextInt();
13        } catch (InputMismatchException e) {
14            System.out.println("Not an integer.");
15        } catch (Exception e) {
16            System.out.println("Unknown exception.");
17        } finally {
18            input.close();
19            System.out.println("Cleanup is done.");
20        }
21
22        System.out.println("End of program.");
23    }
24
25 }
```

Exception Hierarchy⁴

- The topmost class of the exception hierarchy is **Throwable**.
- All **Throwable** subclasses are 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**.

⁴See [Diagram of Exception Hierarchy](#).

Throwing Exceptions

- As a library maker, we disallow some user's behaviors.
- 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
7         if (r <= 0) throw new Exception("Invalid radius.");
8         radius = r;
9
10    }
11
12 }
```

Customized Exceptions

- Use class inheritance to create our own exceptions.

```
1 public class InvalidRadiusException extends Exception {  
2  
3     public InvalidRadiusException(double r) {  
4  
5         super(String.valueOf(r));  
6  
7     }  
8  
9 }
```



```
1 public class Circle {
2
3     private double radius;
4
5     public Circle(double r) throws InvalidRadiusException {
6
7         if (r <= 0) throw new InvalidRadiusException(r);
8         radius = r;
9
10    }
11
12 }
```

```
1 public class NewExceptionDemo {
2
3     public static void main(String[] args) {
4
5         try {
6             new Circle(-10);
7         } catch (InvalidRadiusException e) {
8             System.out.println(e); // Check the result!
9         }
10
11    }
12
13 }
```

Assertion

- An assertion is a statement that enables you to test your assumption 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 AssertionDemo {
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

- Writing test codes is to automate the testing routine for future changes.
 - What works in the past should work after modification.⁵
- However, we should avoid writing test codes together with the normal codes!
- In practice, you may use JUnit⁶ to write test cases for your project.

⁵See also [Test-Driven Development \(TDD\)](#).

⁶See <https://junit.org/>.

Fin.