Chapter 17 Exceptions and Assertions

CS430/630: subset of slides from Java textbook by Liang with a few added comments

Objectives

- To know what is exception and what is exception handling (§17.2).
- To distinguish exception types: Error (fatal) vs. Exception (non-fatal), and checked vs. unchecked exceptions (§17.3).
- To declare exceptions in the method header (§17.3).
- To throw exceptions out of a method (§17.3).
- To write a try-catch block to handle exceptions (§17.3).
- To explain how an exception is propagated (§17.3).
- To rethrow exceptions in a try-catch block (§17.4).
- To use the finally clause in a try-catch block (§17.5).
- To know when to use exceptions (§17.6).

Syntax Errors, Runtime Errors, and Logic Errors

You learned that there are three categories of errors: syntax errors, runtime errors, and logic errors. Syntax errors arise because the rules of the language have not been followed. They are detected by the compiler. Runtime errors occur while the program is running if the environment detects an operation that is impossible to carry out. Logic errors occur when a program doesn’t perform the way it was intended to.

Catch Runtime Errors

```java
import java.util.*;

public class handleExceptionDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean continuedInput = true;
        do {
            System.out.print("Enter an integer: ");
            int number = scanner.nextInt();
            System.out.println("The number entered is "+ number);
            continuedInput = false;
        } catch (InputMismatchException ex) {
            System.out.println("Incorrect input: an integer is required");
            scanner.nextInput(); // discard input
        } while (continuedInput);
    }
}
```

Runtime Errors

```java
import java.util.Scanner;

public class ExceptionDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int number = scanner.nextInt();
        System.out.println("The number entered is "+ number);
    }
}
```

Exception Classes

```
Exception

Error (fatal)

RuntimeError

Checked Exceptions

unchecked

NullPointerException

IllegalArgumentException

Exception

IOException

ClassNotFoundException

VirtualMachineError

IOException...

Several more classes

NullPointerException

IllegalArgumentException

```
System Errors

System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

Exceptions

Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program. Some are checked exceptions.

Unchecked Exceptions

In most cases, unchecked exceptions reflect programming logic errors that are not recoverable. For example, a NullPointerException is thrown if you access an object through a reference variable before an object is assigned to it; an IndexOutOfBoundsException is thrown if you access an element in an array outside the bounds of the array. These are the logic errors that should be corrected in the program. Unchecked exceptions can occur anywhere in the program. To avoid cumbersome overuse of try-catch blocks, Java does not mandate you to write code to catch unchecked exceptions.

Checked or Unchecked Exceptions
Declaring, Throwing, and Catching Exceptions

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

method2() throws Exception {
  if (an error occurs) {
    throw new Exception();
  }
}
catch exception throw exception
declare exception

Declaring Exceptions

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

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

Throwing Exceptions

When the program detects an error, the program can create an instance of an appropriate exception type and throw it. This is known as **throwing an exception**. Here is an example,

```java
throw new TheException();
TheException ex = new TheException();
throw ex;
```

Throwing Exceptions Example

```java
/** Set a new radius */
public void setRadius(double newRadius)
  throws IllegalArgumentException
{
  if (newRadius >= 0)
    radius = newRadius;
  else
    throw new IllegalArgumentException("Radius cannot be negative");
}
```

Catching Exceptions

```java
try {
  statements; // Statements that may throw exceptions
} catch (Exception1 exVar1) {
  handler for exception1;
} catch (Exception2 exVar2) {
  handler for exception2;
} ...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

Catching Exceptions

- If the exception is of type Exception3, it will cause the catch of method2 to execute.
- If the exception is of type Exception2, it will cause the catch of method1 to execute.
- If the exception is of type Exception1, it will cause the catch of method to execute.
Catch or Declare Checked Exceptions

Java forces you to deal with checked exceptions. If a method declares a checked exception (i.e., an exception other than `Error` or `RuntimeException`), you must invoke it in a `try-catch` block or declare to throw the exception in the calling method. For example, suppose that method `p1` invokes method `p2` and `p2` may throw a checked exception (e.g., `IOException`), you have to write the code as shown in (a) or (b).

```java
void p1() {
    try {
        p2();
    } catch (IOException ex) {
        ...
    }
}
```

(a)

```java
void p1() throws IOException {
    p2();
}
```

(b)

Rethrowing Exceptions

```java
try {
    statements;
} catch(TheException ex) {
    perform operations before here; throw ex;
}
```

The finally Clause

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

CS430/630: we'll use `finally` to close our JDBC objects, needed because they involve OS resources.

Trace a Program Execution

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

Suppose no exceptions in the statements

Trace a Program Execution

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

Next statement in the method is executed

The final block is always executed
Trace a Program Execution

try {
  statement1;
  statement2;
  statement3;
} catch(Exception1 ex) {
  handling ex;
} finally {
  finalStatements;
} Next statement;

Suppose an exception of type Exception1 is thrown in statement2.

The exception is handled.

The final block is always executed.

The next statement in the method is now executed.

statement2 throws an exception of type Exception2.

Handling exception.
Trace a Program Execution

```java
try {
    statement1;
    statement2;
    statement3;
} catch(Exception1 ex) {
    handling ex;
} catch(Exception2 ex) {
    handling ex;
    throw ex;
} finally {
    finalStatements;
}
Next statement;
```

Execute the final block

```java
try {
    statement1;
    statement2;
    statement3;
} catch(Exception1 ex) {
    handling ex;
} catch(Exception2 ex) {
    handling ex;
    throw ex;
} finally {
    finalStatements;
}
Next statement;
```

Rethrow the exception and control is transferred to the caller

Cautions When Using Exceptions

- Exception handling separates error-handling code from normal programming tasks, thus making programs easier to read and to modify. Be aware, however, that exception handling usually requires more time and resources because it requires instantiating a new exception object, rolling back the call stack, and propagating the errors to the calling methods.

When to Throw Exceptions

- An exception occurs in a method. If you want the exception to be processed by its caller, 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.

However, ignoring an error is a "code smell": always explain in a comment if you choose to do nothing about an exception.