Objectives in Exception Handling

- To know what is exception and what is exception handling
- To declare exceptions in the method header
- To throw exceptions out of a method
- To write a `try-catch` block to handle exceptions
Syntax Errors, Runtime Errors, and Logic Errors

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.

An Example of Runtime Errors

```java
import java.util.Scanner;

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

What if the user enters a floating-point number?

Run
Exception Handling

Exceptions occur for various reasons

- `ArrayIndexOutOfBoundsException`: if we want to access an array element past the end of the array.
- `NullPointerException`: if we want to invoke a method on a reference variable with null value
- `StringIndexOutOfBoundsException`: if we want to access a character in a string using an out-of-bound index
- `FileNotFoundException`: open a file that doesn’t exist
- `IllegalArgumentException`: a method is passed an argument that is illegal or inappropriate
- Many more

Exception handling is to develop robust programs for mission-critical applications

Catch Runtime Errors

```java
import java.util.*;

public class HandleExceptionDemo {
    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean continueInput = true;

        do {
            try {
                System.out.print("Enter an integer: ");
                int number = scanner.nextInt();
                // Display the result
                System.out.println("The number entered is "+ number);
                continueInput = false;
            } catch (InputMismatchException ex) {
                System.out.println("Try again. (" +
                        "Incorrect input: an integer is required")
                );
                scanner.nextLine(); // discard input
            }
        } while (continueInput);
    }
}
```
Exception Classes

Exceptions are represented in the Exception class in Java

- System errors are represented in the Error class, not our business

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

We can create own exception classes by extending Throwable!

Java Exception Handling Model

- Three operations: declaring an exception, throwing an exception, and catching an exception

- Keywords: throw and throws

```java
class Method1 {
    void method1() {
        try {
            invoke method2;
        } catch (Exception ex) {
            Process exception;
        }
    }
}
class Method2 {
    void method2() throws Exception {
        if (an error occurs) {
            throw new Exception();
        }
    }
}
```
Declaring Exceptions

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

```java
public void myMethod() throws IOException
public void myMethod() throws IOException, OtherException
// if the method might throw multiple exceptions
```

Throwing Exceptions

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

- One throw statement can only throw one exception though one method can throw multiple exceptions.

Here is an example,

```java
throw new IllegalArgumentException("Wrong Argument");
or
IllegalArgumentException ex = new IllegalArgumentException("Wrong Argument");
throw ex;
```
Throwing Exceptions Example

- demonstrates exception declaring and throwing by modifying `setRadius` in the `Circle3` class.

```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");
}
```

- The argument is called the `exception message`, which can be obtained using `getMessage()`.

Catching an Exception

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

What about catching an exception in chained methods invoking?

```java
main method {
    try {
        invoke method1;
        statement1;
    }
    catch (Exception1 ex1) {
        Process ex1;
    }
    statement2;
}
```
Exception Handling Example

- demonstrates exception handling by modifying setRadius in the Circle3 class.

```java
public class CircleWithException {
    ...;
    public CircleWithException(double newRadius) {
        setRadius(newRadius);
    }
    public void setRadius(double newRadius) throws IllegalArgumentException {
        if (newRadius >= 0)
            radius = newRadius;
        else
            throw new IllegalArgumentException("Radius cannot be negative");
    }
}
```

```java
public class TestCircleWithException {
    public static void main(String[] args) {
        try {
            CircleWithException c1 = new CircleWithException(5);
            CircleWithException c2 = new CircleWithException(-5);
            CircleWithException c3 = new CircleWithException(0);
        } catch (IllegalArgumentException ex) {
            System.out.println(ex);
        }
        System.out.println("Number of objects created: " + CircleWithException.getNumberOfObjects());
    }
}
```

What if removing try...catch block?

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) {
        ...;
    }
}
```

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

(a) (b)
Creating Own Exception Classes

- What to do if we a method needs an exception object not available in the standard Exception class?
  - Derive own exception class from one the in the Exception hierarchy

```
public class SpecialFileException extends IOException {
    // Throws exception if file contains letter 'Z'
    // Constructors
    public SpecialFileException () {
    }
    public SpecialFileException (String message) {
        super(message);
    }
}
```

```
public class SomeApplication {
    public String readSpecialData(BufferedReader in) throws SpecialFileException {
        // Throws exception if file contains letter 'Z'
        while (...) {
            if (ch == 'Z') // character 'Z' encountered
                throw new SpecialFileException("Z encountered");
            ...
        }
    }
}
```

The finally Clause

- What to do if we want some code to be executed regardless of whether an exception occurs or is caught?
  - If no exception arises in the try block...
  - If one of the try statements courses an exception and caught in a catch block w/ or w/o exception rethrowing...
  - If one of the try statements courses an exception and NOT caught in a catch block... then passed to the method caller
  - The finally block executes even if there is a return statement prior to reaching the finally block

```
try {
    statements;
}
catch(TheException ex) {
    handling ex;
}
finally {
    finalStatements;
}
statementAfterTry;
```
**Program Execution Example I**

What is the printout of the following code?

```java
public class Test {
    public static void main(String[] args) {
        try {
            int value = 30;
            if (value < 40)
                throw new Exception("value is too small");
        }
        catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        System.out.println("Continue after the catch block");
    }
}
```

What would be the printout if the line

```java
int value = 30;
```

is changed to

```java
int value = 50;
```

---

**Program Execution Example II**

Suppose that `statement2` causes an exception in the following `try-catch` block:

```java
try {
    statement1;
    statement2;
    statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
} statement4;
```

Answer the following questions:
- Will `statement3` be executed?
- If the exception is not caught, will `statement4` be executed?
- If the exception is caught in the `catch` block, will `statement4` be executed?
- If the exception is passed to the caller, will `statement4` be executed?
When to Use Exceptions

When should you use the try-catch block in the code? You should use it to deal with *unexpected* error conditions. Do not use it to deal with simple, expected situations. For example:

```
try {
    System.out.println(refVar.toString());
} catch (NullPointerException ex) {
    System.out.println("refVar is null");
}
```

```
if (refVar != null)
    System.out.println(refVar.toString());
else
    System.out.println("refVar is null");
```