Java Programming
Chapter 9

Exception

Java Error

- Java Errors
  - Even experienced Java developers make mistakes. The key is learning how to **spot** and **fix** them
- Types of Errors in Java
  - Compile-Time Error
    - Detected by the compiler. Prevents code from running.
  - Runtime Error
    - Occurs while the program is running. Often causes crashes.
  - Logical Error
    - Code runs but gives incorrect results. Hardest to find.

Java Error

- Common Compile-Time Errors
  - Compile-time errors occur when the program cannot compile due to syntax or type issues.

1) Missing Semicolon

```
int x = 5
System.out.println(x);
```

```
error: ';' expected
```

2) Undeclared Variables

```
System.out.println(myVar);
```

```
cannot find symbol
  symbol: variable myVar
```

3) Mismatched Types

```
int x = "Hello";
```

```
incompatible types: String cannot be converted to int
```

Java Error

- Common Runtime Errors
  - **Runtime errors** occur when the program compiles but crashes or behaves unexpectedly.

1) Division by Zero

```
int x = 10;
int y = 0;
int result = x / y;
System.out.println(result);
```

Result:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

2) Array Index Out of Bounds

```
int[] numbers = {1, 2, 3};
System.out.println(numbers[8]);
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index 8 out of bounds for length 3
```

Java Error

- Logical Errors
  - **Logical errors** happen when the code runs, but the result is not what you thought:

```
int x = 10;
int y = 2;
int sum = x - y;
System.out.println("x + y = " + sum);
```

```
x + y = 8
```

Logical Error: The code mistakenly subtracts instead of adds.

Java Error

- Good Habits to Avoid Errors
  - Use meaningful variable names
  - Read the error message carefully. What line does it mention?
  - Check for missing semicolons or braces
  - Look for typos in variable or method names

Java Debugging

- What is Debugging?
  - **Debugging** is the process of identifying and fixing errors or bugs in your code.
  - It often involves:
    - Reading error messages
    - Tracing variable values step by step
    - Testing small pieces of code independently

Java Debugging

- Print Statements for Debugging
  - The most basic (and often most effective) way to debug Java code is to use System.out.println() to print values and check the flow of the program.
  - In this example, the first line "Before division" will print, but the second line is never reached because the program crashes due to division by zero:

```java
int x = 10;
int y = 0;

System.out.println("Before division");  // Debug output

int result = x / y;  // Crashes

System.out.println("Result: " + result);  // Never runs
Result:
  return go(f, seed, [])
}
```

```
Before division
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Java Debugging

- Check Variable Values
  - If something unexpected happens, print out the values of your variables:

```java
int age = 17;
System.out.println("Age: " + age);

if (age ⩾ 18) {
  System.out.println("Access granted");
} else {
  System.out.println("Access denied");
}
```

Java Debugging

- Debugging with IDEs
  - Modern IDEs like **IntelliJ IDEA**, **Eclipse**, and **NetBeans** come with built-in debugging tools.
    - Set **breakpoints** to pause the program at specific lines
    - Step through code line by line
    - Inspect variable values in real time

Java Exceptions

- Java Exceptions
  - different types of errors can occur while running a program - such as coding mistakes, invalid input, or unexpected situations.

  - When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).

Java Exceptions

- Exception Handling (try and catch)
  - Exception handling lets you catch and handle errors during runtime - so your program doesn't crash.
  - It uses different keywords:
    - The **try** statement allows you to define a block of code to be tested for errors while it is being executed.
    - The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.
    - The try and catch keywords come in pairs:

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Java Exceptions

```java
public class Main {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
        at Main.main(Main.java:4)
```

Java Exceptions

- If an error occurs, we can use try...catch to catch the error and execute some code to handle it:

```java
public class Main {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

Java Exceptions

- Finally
  - The finally statement lets you execute code, after try...catch, regardless of the result:

```java
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

Java Exceptions

- The throw
  - The throw statement allows you to create a custom error.
  - The throw statement is used together with an exception type.
  - There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc:

Java Exceptions

- Throw an exception if age is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```java
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Java Exceptions

Errors and Exception Types

Error/Exception	Description
ArithmeticError	Occurs when a numeric calculation goes wrong
ArrayIndexOutOfBoundsException	Occurs when trying to access an index number that does not exi st in an array
ClassNotFoundException	Occurs when trying to access a class that does not exist
FileNotFoundException	Occurs when a file cannot be accessed
InputMismatchException	Occurs when entering wrong input (e.g. text in a numerical inpu t)
IOException	Occurs when an input or output operation fails
NullPointerException	Occurs when trying to access an object referece that is null
NumberFormatException	Occurs when it is not possible to convert a specified string to a numeric type
StringIndexOutOfBoundsException	Occurs when trying to access a character in a String that does n ot exist

Multiple Exceptions

- Multiple Exception
  - Sometimes, different errors (exceptions) can happen in the same try block. You can handle them with multiple catch blocks.

```java
public class Main {
  public static void main(String[] args) {
    try {
      int[] numbers = {1, 2, 3};
      System.out.println(numbers[10]);  // ArrayIndexOutOfBoundsException
      int result = 10 / 0;              // ArithmeticException
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Array index does not exist.");
    }
    catch (ArithmeticException e) {
      System.out.println("Cannot divide by zero.");
    }
    catch (Exception e) {
      System.out.println("Something else went wrong.");
    }
  }
}
}
```

Multiple Exceptions

- Multiple Exception
  - You should always put **more specific exceptions first**, and **general ones later**. Otherwise, the general catch will grab the error and the specific ones will never run:

```java
try {
  int result = 10 / 0;
}
catch (Exception e) {
  System.out.println("General error.");
}
catch (ArithmeticException e) {
  // This will never be reached
  System.out.println("Divide by zero.");
}
```

Always put Exception (the general one) at the end.

Java Exception

- throws

```java
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

```java
public class Main {
  static void checkAge(int age) throws ArithmeticException {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

 public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Java Exception

- Definition and Usage
  - The throws keyword indicates what exception type may be thrown by a method.
  - There are many exception types available in Java: ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.
  - Differences between throw and throws:

throw	throws
Used to throw an exception for a method	Used to indicate what exception type may be throw n by a method
Cannot throw multiple exceptions	Can declare multiple exceptions
•**Syntax:**throw is followed by an object (new *type*) •used inside the method	•**Syntax:**throws is followed by a class •and used with the method signature

https://github.com/kkh3363/bsu_java2025