

Exception Handling and Text I/O

Introduction

Exception Handling Overview

Exception Types

ArithmeticException, NullPointerException,
IndexOutOfBoundsException

Introduction

When a program runs into a runtime error (also known as exception), the program terminates abnormally. How can you handle the runtime error so that the program can continue to run or terminate gracefully? This is the subject we will introduce in this chapter.

Exception Handling Overview

```
import java.util.Scanner;
public class Application {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();
        int quotient = number1 / number2;
        System.out.println("The quotient is " + quotient);
    }
}
```

This program does not handle the runtime error. It displays it and the program stops.

Sample Run 1

Enter two integers: 5 2
The quotient is 2

Sample Run 2

Enter two integers: 3 0

Exception in thread "main" java.lang.ArithmeticException: / by zero
at Application.main(Application.java:8)

Exception Handling Overview

```
import java.util.Scanner;
public class Application {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();
        if (number2 != 0) {
            int quotient = number1 / number2;
            System.out.println("The quotient is " + quotient);
        }
        else
            System.out.println("Cannot divide by zero.");
    }
}
```

This program fixes the runtime error (exception) using an if statement.

Sample Run 1

Enter two integers: 5 2
The quotient is 2

Sample Run 2

Enter two integers: 3 0
Cannot divide by zero.

Exception Handling Overview

```
import java.util.Scanner;
public class Application {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1, number2, quotient;
        System.out.print("Enter two integers: ");
        number1 = input.nextInt();
        number2 = input.nextInt();
        try {
            quotient = number1 / number2;
            System.out.println("The quotient is " + quotient);
        }
        catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero.");
        }
        System.out.println("The end.");
    }
}
```

This program fixes the runtime error (exception) using a try catch statement.

Sample Run 1

Enter two integers: 5 2
The quotient is 2
The end.

Sample Run 2

Enter two integers: 3 0
Cannot divide by zero.
The end.

Exception Handling Overview

```
import java.util.Scanner;
public class Application {
    public static int quotient(int n1, int n2) {
        if (number2 == 0)
            throw new ArithmeticException("Divisor cannot by zero.");
        return n1 / n2;
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1, number2, q;
        System.out.print("Enter two integers: ");
        number1 = input.nextInt();
        number2 = input.nextInt();
        try {
            q = quotient(number1, number2);
            System.out.println("The quotient is " + q);
        }
        catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero.");
        }
        System.out.println("The end.");
    }
}
```

This program fixes the runtime error (exception) using a try catch statement.

Sample Run 1

Enter two integers: 5 2
The quotient is 2
The end.

Sample Run 2

Enter two integers: 3 0
Cannot divide by zero.
The end.

Throwing and Catching an Exception

The value thrown, in the example in slide 6 which is **new ArithmeticException("Divisor cannot be zero")**, is called an *exception*.

The execution of a **throw** statement is called *throwing an exception*.

The exception is an object created from an exception class. In this case, the exception class is **java.lang.ArithmeticException**.

The constructor **ArithmeticException(str)** is invoked to construct an exception object, where **str** is a message that describes the exception.

Throwing and Catching an Exception

When an exception is thrown, the normal execution flow is interrupted. As the name suggests, to “throw an exception” is to pass the exception from one place to another. The statement for invoking the method is contained in a **try** block and a **catch** block.

The **try** block contains the code that is executed in normal circumstances. The exception is caught by the **catch** block. The code in the **catch** block is executed to *handle the exception*.

Afterward, the statement after the **catch** block is executed.

Throwing and Catching an Exception

The **throw** statement is analogous to a method call, but instead of calling a method, it calls a **catch** block.

In this sense, a **catch** block is like a method definition with a parameter that matches the type of the value being thrown.

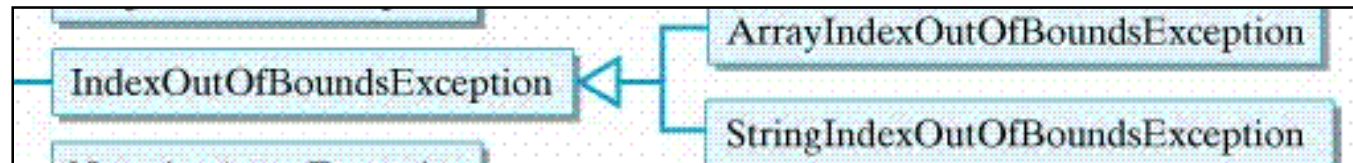
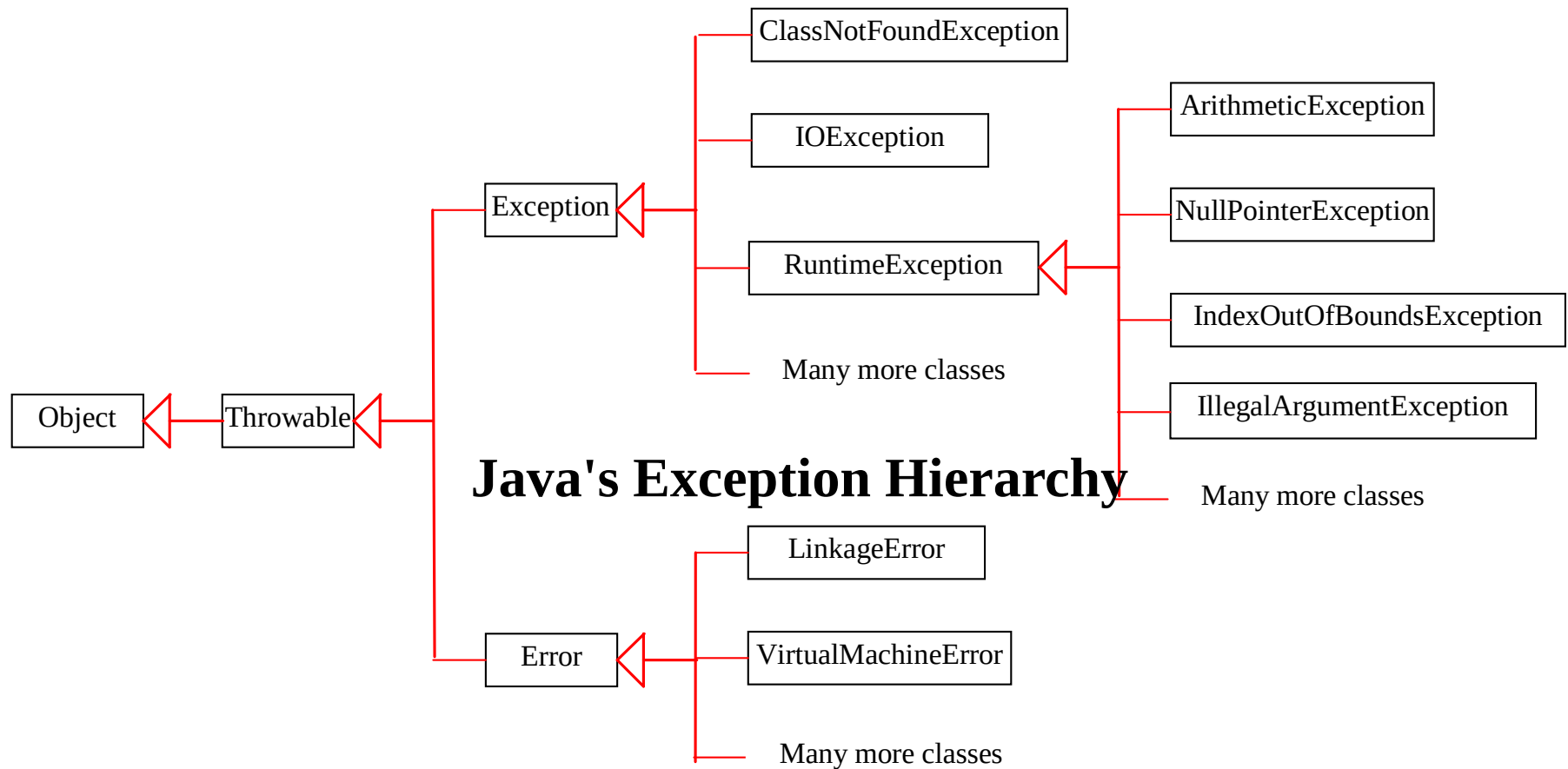
Unlike a method, however, after the **catch** block is executed, the program control does not return to the **throw** statement; instead, it executes the next statement after the **catch** block.

Exception Advantages

Obviously the first advantage of exception handling is that the program execution will not stop when an exception occurs. This is shown in slides 5 and 6.

In slide 6 you can see another *advantage* of using exception handling. It enables a method to throw an exception to its caller. Without this capability, a method must handle the exception or terminate the program.

Exception Types



Each of the classes above identifies a particular type of exception, and each is a subclass of the Exception class. Obviously a subclass defines a more specific exception than its superclass.

Some of Java's Important Exceptions.

Class	Description
<code>ArithmeticException</code>	Division by zero or some other arithmetic problem
<code>ArrayIndexOutOfBoundsException</code>	An array index is less than zero or greater than or equal to the array's length
<code>FileNotFoundException</code>	Reference to a file that cannot be found
<code>IllegalArgumentException</code>	Calling a method with an improper argument
<code>IndexOutOfBoundsException</code>	An array or string index is out of bounds
<code>NullPointerException</code>	Reference to an object that has not been instantiated
<code>NumberFormatException</code>	Use of an illegal number format, as when calling a method
<code>StringIndexOutOfBoundsException</code>	A string index is less than zero or greater than or equal to the string's length

The Java class library contains a number of predefined exceptions, some of which are shown in the figure below. The most general type of exception, `java.lang.Exception`, is located in the `java.lang` package, but most of its subclasses are contained in other packages. Some of the various `IOException` classes are contained in the `java.io` package.

Exceptions that we will cover

1. `ArithmeticException` (Covered in slides 5, 6, and 14)
2. `InputMismatchException`
3. `NullPointerException`
4. `IndexOutOfBoundsException`
5. `IllegalArgumentException`
6. `Exception`
7. `IOException` (Will be covered with files)
8. `FileNotFoundException` (Will be covered with files)

ArithmeticException

The arithmetic exception is thrown whenever a wrong mathematical or arithmetic operation appears in the code during run time. It appears whenever the denominator of a fraction is 0.

We learned how to handle an `ArithmeticException` in slides 5 and 6.

NullPointerException & IndexOutOfBoundsException Example

```
public class Student{
    private int id;
    private String name;
    public Student(int i, String n) {
        id = i;
        name = n;
    }
    public void print() {
        System.out.println("The ID is" + id + " The name is " + name);
    }
}

import java.util.Scanner;
public class Application {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        Student s[]=new Student[10];
    }
}
```

NullPointerException & IndexOutOfBoundsException Example (Continued)

```
s[0]=new Student(1,"A");  
s[1]=new Student(2,"B");  
int index;  
System.out.println("Enter an index:");  
index = scan.nextInt();  
try {  
    s[index].print();  
}  
catch (IndexOutOfBoundsException e1) {  
    System.out.println("Index Out of bounds");  
}  
catch (NullPointerException e) {  
    System.out.println("Attempting to access a null reference");  
}  
System.out.println("The end");  
}
```

Handling NullPointerException & IndexOutOfBoundsException

The program in the previous two slides will output the message 'Index Out of bounds.' if the user enters an index < 0 or > 9 .

The program will output the message 'Attempting to access a null reference.' if the user enters an index of a cell that contains null such as cells 2 to 9.

Otherwise, the program will output the id and name of the student object at the specified index (if index is 0 or 1) .