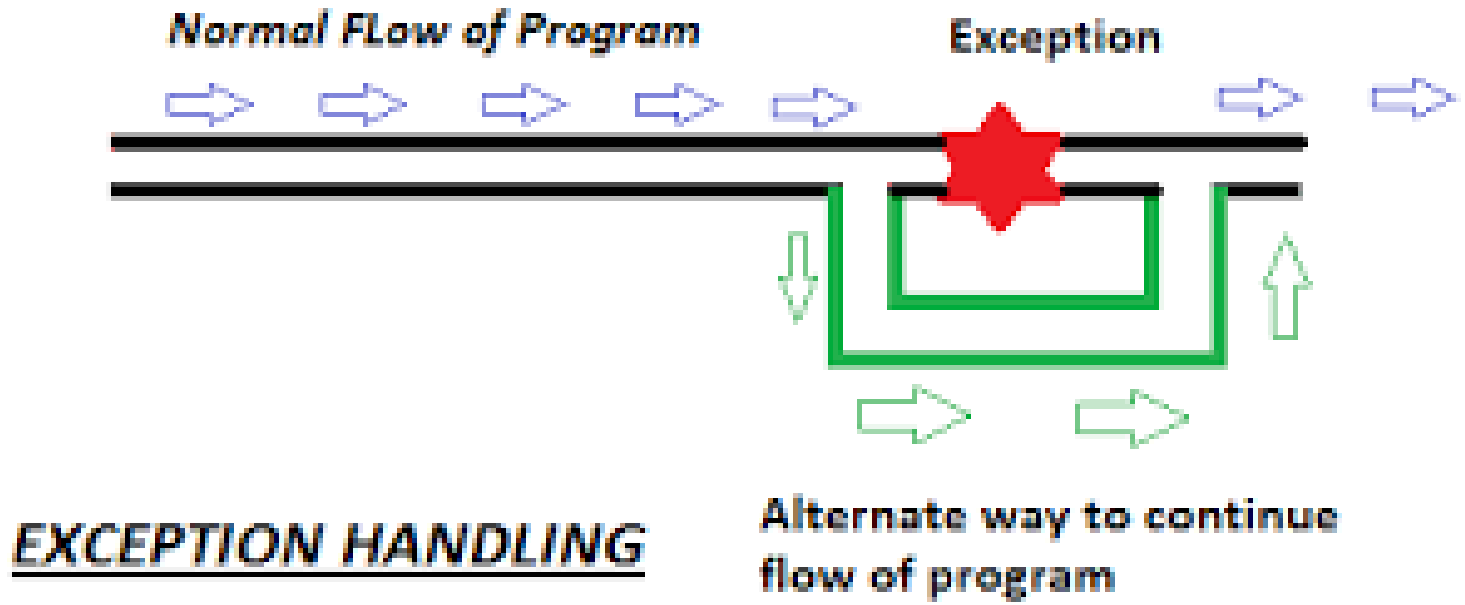
Java Programming AP Edition
U4C12 Exception Handling and I/O

EXCEPTION OVERVIEW

ERIC Y. CHOU, PH.D. IEEE SENIOR MEMBER

Exceptions


Normal FLow of Program
Exception
EXCEPTION HANDLING
Alternate way to continue flow of program

Introduction

Exception Handling enables a program to deal with exceptional situations and continue its normal execution.

Runtime errors occur while a program is running if the JVM detects an operation that is impossible to carry out. For example, if you access an array using an index that is out of bounds, you will get a runtime error with an **ArrayIndexOutOfBoundsException**. If you enter a double value when your program expects an integer, you will get a runtime error with an **InputMismatchException**.

In Java, runtime errors are thrown as exceptions. **An exception is an object that represents an error or a condition that prevents execution from proceeding normally.** If the exception is not handled, the program will terminate abnormally. How can you handle the exception so that the program can continue to run or else terminate gracefully? This chapter introduces this subject and text input and output.

Exception Classes are information classes like Class class returned by getClass().

Issues Involves Exceptions

The **exception handling** in java is one of the powerful mechanism to handle the **runtime** errors so that normal flow of the application can be maintained.

- Why it happens and where does it happens?
- How to raise an exception and how to catch it?
- How to handle it?

Exception-Handling Overview Exemplary Run-Time Errors

- Show runtime error (**Quotient.java**) occurs when division by 0.
- Fix it using an if statement (**QuotientWithIf.java**) (check and prevent the division by 0 to happen)
- Using method to quarantine the division error (**QuotientWithMethod.java**)
- Introduce try-catch (**QuotientWithException.java**)

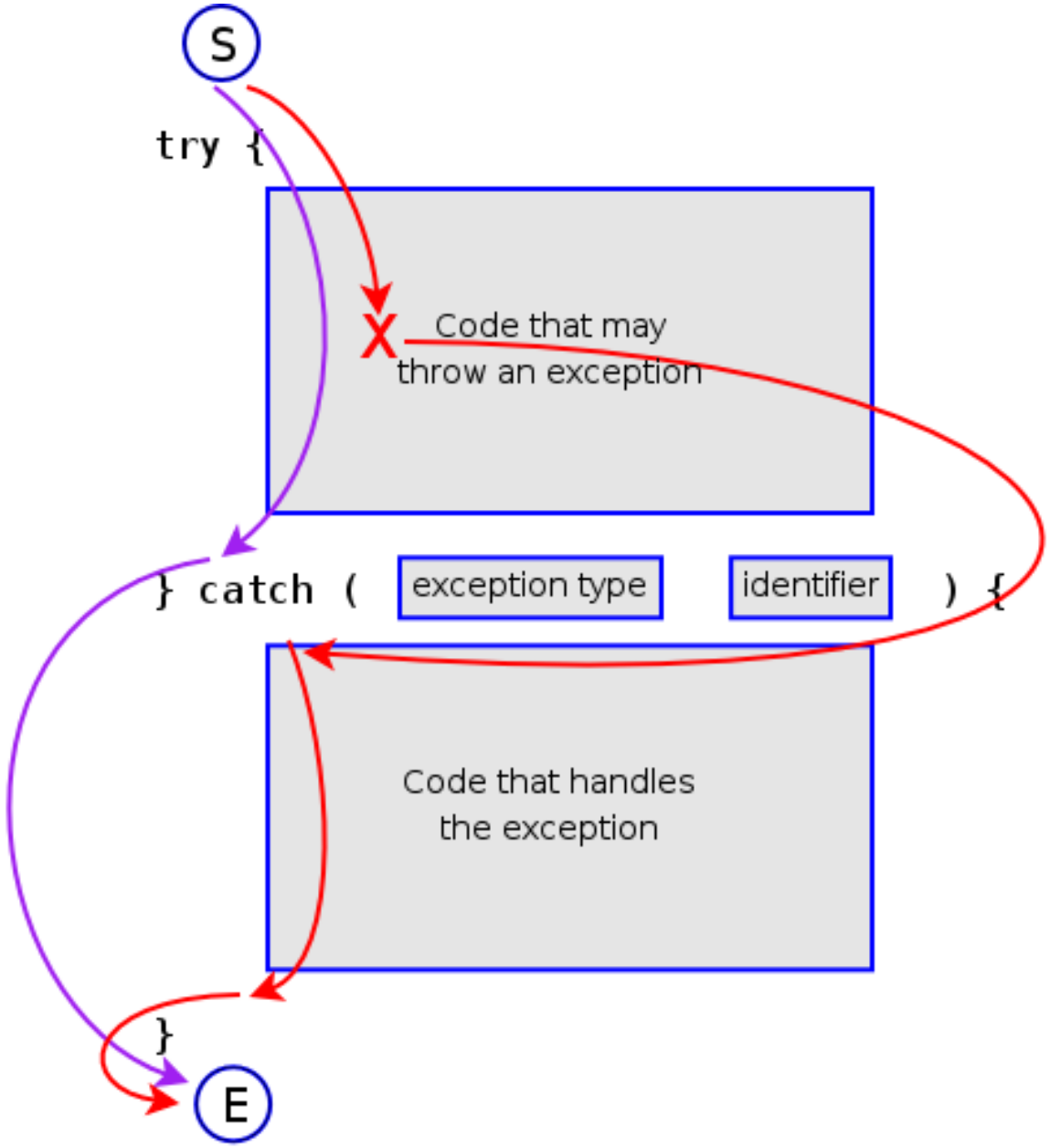
Exception Advantages

Handle run-time errors with Exceptions.
(**QuotientWithException.java**)

Now you see the *advantages* 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 throws and catch

Exception classes are **information** class which contains types of exception and the other information.



From the Method to Exception Handling

The method **quotient** returns the quotient of two integers. If **number2** is **0**, it cannot return a value, so the program is terminated. This is clearly a problem. You should not let the method terminate the program—the *caller* should decide whether to terminate the program.

How can a method notify its caller an exception has occurred? Java enables a method to throw an exception that can be caught and handled by the caller. Listing 12.3 can be rewritten, as shown in QuotientWithException.java.

If **number2** is **0**, the method throws an exception by executing

throw new ArithmeticException(**"Divisor cannot be zero"**);

The value thrown, in this case **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.

Exception Handling

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. 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 Handling

like calling a method

The identifier **ex** in the **catch**–block header

catch (ArithmeticException ex)

acts very much like a parameter in a method. Thus, this parameter is referred to as a **catch**–block parameter. The type (e.g., **ArithmeticException**) preceding **ex** specifies what kind of exception the **catch** block can catch. Once the exception is caught, you can access the thrown value from this parameter in the body of a **catch** block.

In summary, a template for a **try**-**throw**-**catch** block may look like this:

```
try {
    Code to run;
     A statement or a method that may throw an
     exception;
    More code to run;
}
catch (type ex) {
    Code to process the exception;
}
```

An exception may be thrown directly by using a **throw** statement in a **try** block, or by invoking a method that may throw an exception.

Handling InputMismatchException

Errors when input data type mismatched (**InputMismatchException.java**)

By handling InputMismatchException, your program will continuously read an input until it is correct.