Chapter 5 Exception handling

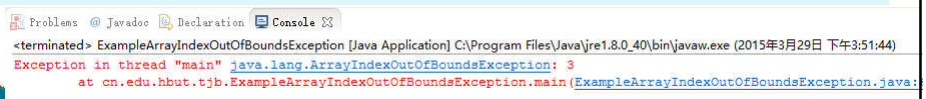
Introduction

- An exception is an event, which occurs during the execution of a program, and disrupts the normal flow of the program's instructions.
- The term exception is a shorthand for the phrase "exceptional event."
- Creating an exception object and handing it to the runtime system is called throwing an exception.

Introduction

- /**
- * The size of the array a is 3, however this program tends to visit the 4th element.
- * Therefore, an exception "ArrayIndexOutOfBoundsException" is thrown.
- */
- public class ExampleArrayIndexOutOfBoundsException {

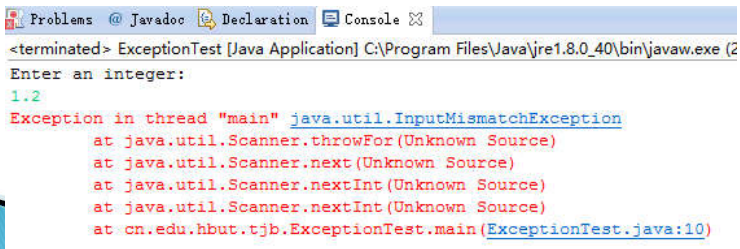
- It displays:
  - Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
  - at cn.edu.hbut.tjb.ExampleArrayIndexOutOfBoundsException.main(ExampleArrayIndexOutOfBoundsException.java:6)



3

Introduction

- import java.util.Scanner;
-
- public class ExceptionTest {
-
- public static void main(String[] args) {
- Scanner sc = new Scanner(System.in);
- System.out.println("Enter an integer: ");
- int n = sc.nextInt();// If an exception occurs here, the rest code are skipped and the program is terminated
- System.out.println("Your number is: " + n);
- }
-
- }



4

Introduction

- Java allows the program to catch and handle exceptions in order to let the program resume from abnormal situations.

5

Introduction

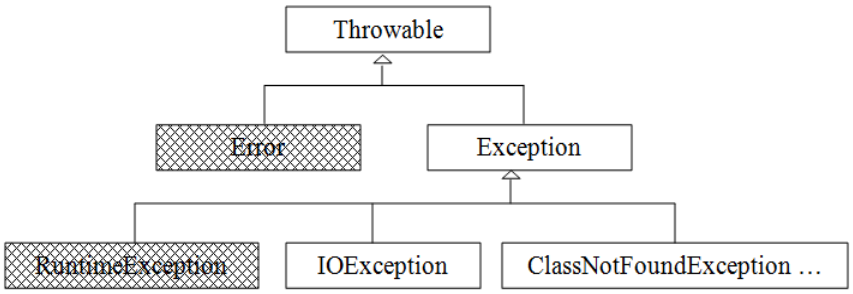
```
import java.util.InputMismatchException;
import java.util.Scanner;

public class HandleExceptionTest {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        boole
        do {
            try
                s
                i
                s
                c
            } ca
                System.out.println("Incorrect input: An integer is required. Try again.");
                sc.nextLine();//discard the incorrect input
                continue;
            }
        } while (!ok);
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> HandleExceptionTest [Java Application] C:\Program Files\Java\jre
Enter an integer:
1.3
Incorrect input: An integer is required. Try again.
Enter an integer:
13
Your number is: 13
```

6

Introduction



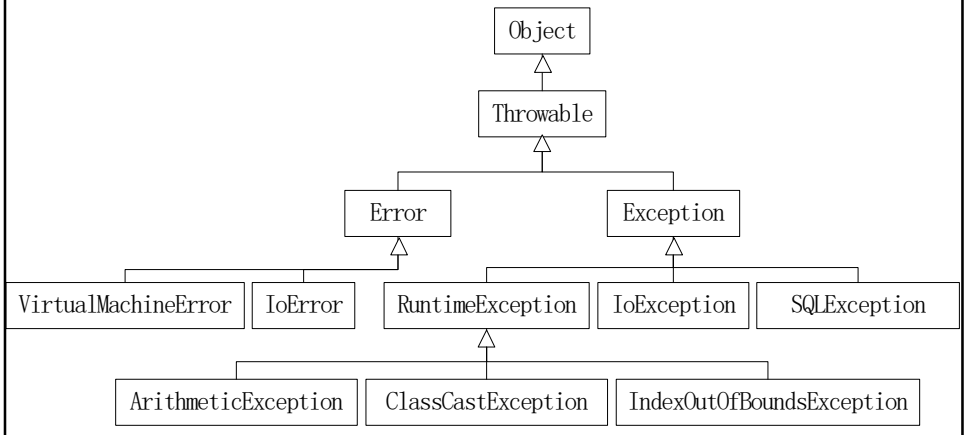
7

Introduction

- An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.
- Once an error happens, there is little one can do except terminating the program gracefully.
- The Exception objects are caused by program and by external circumstances. They can be caught and handled in your programs.

8

Hierarchy of Exceptions



Predefined Exception Classes

java.lang
类 Exception

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception

所有已实现的接口：
Serializable

直接已知子类：
AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CertificateException, ClassNotFoundException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalAccessException, IllegalClassFormatException, InstantiationException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvocationTargetException, IOException, JMException, LastOwnerException, LineUnavailableException, MidiUnavailableException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NoSuchFieldException, NoSuchMethodException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ServerNotActiveException, SQLException, TimeoutException, TooManyListenersException, TransformerException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URISyntaxException, UserException, XAException, XMLParseException, XPathException

Introduction

- RuntimeExceptions are known as **unchecked** 免检exceptions.
- All other exceptions are known as **checked** 检查exceptions, which imply that the compiler forces the programmer to check and deal with them.
- (First meaning of checked/unchecked exceptions in Java)

11

5.1 Introduction

- In most cases, unchecked exceptions reflect programming logic errors that are not recoverable.
- For example, a NullPointerException
- A checked exception needs to either be caught in a catch block, or be thrown on further via a throws clause of a method declaration.

12

5.1 Introduction

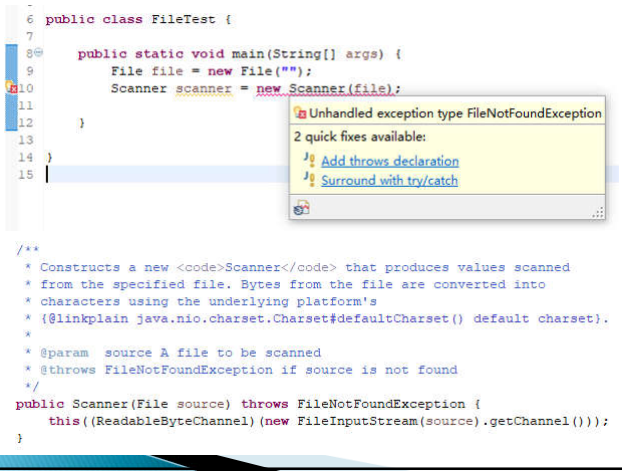
- If an exception is a subclass of RuntimeException, it is an unchecked exception.
- An unchecked exception can be caught, but it does not have to be.

- Class Error and its subclasses also are unchecked.
- In general, you must always consider dealing with the possibility of a checked exception. If you don't, you will get a compilation error.
- (Second meaning of checked/unchecked exceptions in Java)
- The exceptions have bean handled are checked exceptions
- The exceptions have not bean handled are unchecked exceptions

13

Handling Exceptions

- Java code that might throw certain exceptions must be enclosed by either of the following:
  - A method that specifies that it might throw the exception. The method must provide a throws clause that lists the exception.



14

Handling Exceptions

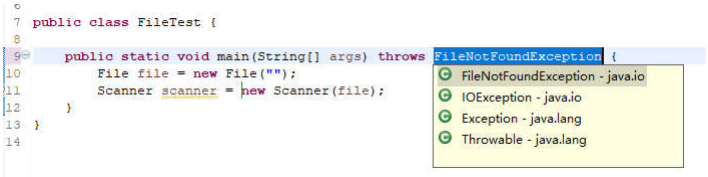
- IOEception is a kind of checked exception and it should be handled in your code.

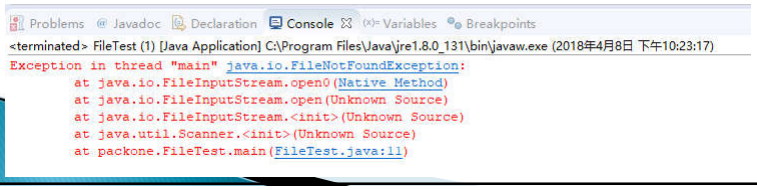
```
public class FileTest {

public static void main(String[] args) {
    File file = new File("");
    Scanner scanner = new Scanner(file);
    }
}
```

15

Handling Exceptions

```
public class FileTest {

public static void main(String[] args) throws FileNotFoundException
{
    File file = new File("");
    Scanner scanner = new Scanner(file);
    }
}
```





16

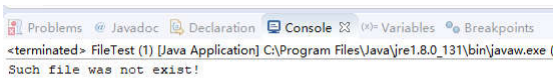
Handling Exceptions

- A try statement that catches the exception. The try must provide a handler for the exception.

17

Handling Exceptions

```
public class FileTest {

    public static void main(String[] args) {
        File file = new File("");
        try {
            Scanner scanner = new Scanner(file);
        } catch (FileNotFoundException e) {
            System.out.println("Such file was not exist!");
        }
    }
}
```



18

Handling Exceptions

- If the statement that caused the exception is inside a try block it goes to that code within the catch block accompanied the try block.
- After the catch clause is executed, execution resumes after the end of the entire try statement.
- It is impossible to return to the point at which the exception was thrown.

19

- If there is no try statement around the code that threw the exception, Java goes up the call stack and looks at the statement that called this method, and checks for a try statement surrounding the call.
- If it finds an enclosing try statement that has a catch clause for this kind of exception, the catch clause is executed and then execution continues from the immediately following statement of that try statement.
- Java continues moving up the call stack until it finds an enclosing try statement.
- If no enclosing try statement and catch clause is found, the exception-handling system receives the exception and prints an error message and terminates the program.

20

Handling Exceptions

- The code that handles the exception in catch block is known as an **exception handler**.
- A try block may associate exception handlers in one or more catch blocks
- The process of looking for a handler is called **catching an exception**.

21

example

```
public class Test{
   public static void main(String[] args) {
      int a[] = new int[3];
      System.out.println("Assigning");
      try{
         a[3] = 25;
         System.out.println("Excuting");
      }
      catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Your have got an exception");
      }
   System.out.println("Assigned");
   }
}
```

22

Handling Exceptions

- try {
- // code block that may throw exceptions
- } catch (ArithmeticException e) {
- // Code bloack for handling an ArithmeticException exception
- } catch (IndexOutOfBoundsException e) {
- // Code block for handling an IndexOutOfBoundsException exception
- }

23

Multiple catch

```
public class Test{
   public static void main(String[] args) {
            int a[] = new int[3];
            try
             {
                        // a[3] = 10;
                        a[3] = 2/0;
            }
            catch(ArithmeticException e)
             {
                        System.out.println("arithmetic Exception");
            }
            catch(ArrayIndexOutOfBoundsException e)
             {
                        System.out.println("array index out of bounds Exception");
            }
            catch(Throwable e)
             {
                        System.out.println("Exception");
            }
   }
}
```

24

Handling Exceptions

- A **match** means that the thrown object can legally be assigned to the exception handler's argument.
- In other words, matching an exception doesn't require strict equality between the exception and its handler. A derived-class object will match a handler for the base class.

```
try{
    int age=Integer.parseInt("24L"); //NumberFormatException
}catch(Exception e){
    System.out.println("caught an Exception");
}catch(NumberFormatException e){
    System.out.println("caught a Number Format Exception");
}
```

25

The finally Block

- The try-block statement has the general form:
  - try{
  -
  - }catch(Exception <object_name>){
  -
  - }finally{
  -
  - }
- finally creates a block of code that will be executed after the try/catch block is completed.
- The finally block will execute whether or not an exception is thrown.

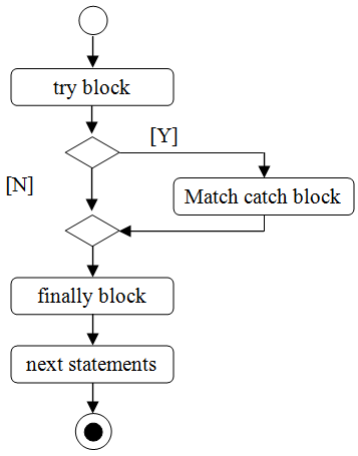
26

The finally Block

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
public class ExampleThrows_4 {
        public String getPassword() {
                Scanner sc = null;
                String s = null;
                try {
                        sc = new Scanner(new File("D:\\java\\test.txt"));
                        s = sc.next();
                } catch (IOException e) {
                        // Return the detail message string of this e, possible null.
                        System.err.println(e.getMessage());
                        // Terminates the currently running Java Virtual Machine
                        System.exit(1);
                } finally {
                        if (sc != null) {
                                sc.close();
                        }
                }
                return s;
        }
}
```

27

The finally Block



28

The finally Block

```
import java.io.IOException;
import java.io.PrintWriter;

public class FinallyTest {
    public static void main(String[] args) {
        PrintWriter output = null;
        try {
            output = new PrintWriter("test.txt");
            output.println("Finally block test.");
        } catch (IOException e) {
            System.out.println(e.getMessage());
            System.exit(0);
        } finally {
            if (output != null) {
                output.close();
            }
        }
    }
}
```

- To ensure that a file closes under all circumstances, there is usually a file closing statement in the finally block

29

The finally Block

- Exception handling separates the code executed in an abnormal state from normal execution.
- This arrangement makes programs easier to read and to modify.

30

The finally Block

- In general, simple exceptions that may occur in individual methods are best handled locally without throw.
- When a method has to deal with an unexpected exception, a try-catch block should be used.

User-defined Exceptions

- When you are going to use your own exceptions, there are three steps:
  - defining an exception,
  - declaring throws,
  - and using the exception.

User-defined Exceptions

- Defines the user exception OutOfScoreRangeException

```
public class OutOfScoreRangeException extends Exception{
    public OutOfScoreRangeException(String msg, int score) {
        super(msg);
        this.score = score;
    }
    public int getScore() {
        return this.score;
    }
    private int score;
}
```

33

User-defined Exceptions

- throw exceptions for the proper situation using the **throw** statement

```
public class Score {
        public Score(int s) throws OutOfScoreRangeException {
                if (s <= 100 && s >= 0) {
                score = s;
                } else {
                throw new OutOfScoreRangeException("Out of
score range exception ", s);
                }
        }
        private int score;
}
```

34

User-defined Exceptions

```
public class TestScore {
    public static void main(String[] args) {
        try {
            Score s1 = new Score(98);
            Score s2 = new Score(800);
            System.out.print(s1);
            System.out.print(s2);
        } catch (OutOfScoreRangeException e) {
            System.out.println(e.getMessage() + e.getScore());
        }
    }
}
```

35

Benefits of Java Exception Handling Framework

- Benefits:Separating Error-Handling Code from "Regular" Code
- The Java exception handling framework provides the ability to separate error handling code from regular code.

36

Benefits of Java Exception Handling Framework

- Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program.

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

- What happens if the file can't be opened?
- What happens if the length of the file can't be determined?
- What happens if enough memory can't be allocated?
- What happens if the read fails?
- What happens if the file can't be closed?

37

Benefits of Java Exception Handling Framework

```
errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;}
```

38

Benefits of Java Exception Handling Framework

- Exceptions enable you to write the main flow of your code and to deal with the exceptional cases elsewhere.

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

39

Benefits of Java Exception Handling Framework

```
public static void main(String[] args) {
Stack s1 = new ArrayStack();
if (s1.pop())
        System.out.println("The element on the top has been released.");
else {
        System.out.println("No element on the top has been released since the stack is empty.");
}
}
```

This kind of code portion is hard to read and understand for programmers.

40

Benefits of Java Exception Handling Framework

- Exception approach:

```
public static void main(String[] args) {
    Stack s1 = new ArrayStack();
    try {
        s1.pop();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
```

41

Benefits of Java Exception Handling Framework

- Note that the pop() method used in Code 5.15 could be declared as:

```
    public void pop() throws Exception {
        if (this.isEmpty())
            throw new Exception(this + " is empty");
        else {
            topOfStack--;
        }
    }
```

42

Benefits of Java Exception Handling Framework

- Benefit: Propagating Errors Up the Call Stack
- The method can ignore any exceptions thrown within it in case the method has no idea how to process them

Benefits of Java Exception Handling Framework

- Suppose that the readFile method is the fourth method in a series of nested method calls made by the main program: method1 calls method2, which calls method3, which finally calls readFile.

```
method1 {
   call method2;
}

method2 {
   call method3;
}

method3 {
   call readFile;
}
```

Benefits of Java Exception Handling Framework

- Suppose also that method1 is the only method interested in the errors that might occur within readFile.

```
method1 {
   errorCodeType error;
   error = call method2;
   if (error)
      doErrorProcessing;
   else
      proceed;
}
errorCodeType method2 {
   errorCodeType error;
   error = call method3;
   if (error)
      return error;
   else
      proceed;
}
errorCodeType method3 {
   errorCodeType error;
   error = call readFile;
   if (error)
      return error;
   else
      proceed;
}
```

45

Benefits of Java Exception Handling Framework

- Suppose also that method1 is the only method interested in the errors that might occur within readFile.

```
method1 {
   try {
      call method2;
   } catch (exception e) {
      doErrorProcessing;
   }
}

method2 throws exception {
   call method3;
}

method3 throws exception {
   call readFile;
}
```

46

Benefits of Java Exception Handling Framework

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class FlowSeperation {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner src = null;
        try {
            src = new Scanner(new File("D:\\java\\Test.txt"));
            while (src.hasNext()) {
                System.out.println(src.next());
            }
        } finally {
            if (src != null) {
                src.close();
            }
        }
    }
}
```

47

Benefits of Java Exception Handling Framework

- Benefit: Grouping and Differentiating Error Types
- Because all exceptions thrown within a program are objects, the grouping or categorizing of exceptions is a natural outcome of the class hierarchy.

48

Benefits of Java Exception Handling Framework

- IOException is the most general and represents any type of error that can occur when performing I/O. Its descendants represent more specific errors.
- FileNotFoundException means that a file could not be located on disk

```
catch (FileNotFoundException e) {
    ...
}
catch (IOException e) {
    ...
}
catch (Exception e) {
    ...
}
```

49

Assertions

- An assertion is an expression that represents a condition that a programmer believes to be true at a specific place in a program.
- If an assertion isn't true, an error occurs.

50

Assertions

```
public class Score {
        public Score(int s) {
                assert s <= 100 && s >= 0;
                score = s;
        }
        private int score;
}

public class AssertionTest {
        public static void main(String[] args) {
                Score s = new Score(101);
        }

}
```

- The assert keyword is followed by a conditional expression

51

Assertions

- If the assertion that follows the **assert** keyword is not true, an AssertionError exception is thrown.
- To make the error message associated with an assertion more meaningful, you can specify a string in an assert statement, as in the following example:
  - assert s <= 100 && s >= 0: "Score >100 or score < 0";

52

AssertTest.java

```
package cn.edu.hbut.tjb;

public class AssertTest {

    public static void main(String[] args) {
        boolean isOk = 1>2;
        try{
            assert isOk : "程序错误";
            System.out.println("程序正常");
          }catch(AssertionError err){
            System.err.println(err.getMessage());
        }

    }
}
```

Problems | Javadoc | Declaration | Console

<terminated> AssertTest [Java Application] C:\Program Fil

程序错误

AssertTest.java

```
package cn.edu.hbut.tjb;

public class AssertTest {

    public static void main(String[] args) {
        boolean isOk = 3>2;
        try{
            assert isOk : "程序错误";
            System.out.println("程序正常");
          }catch(AssertionError err){
            System.err.println(err.getMessage());
        }

    }
}
```

Problems | Javadoc | Declaration | Console

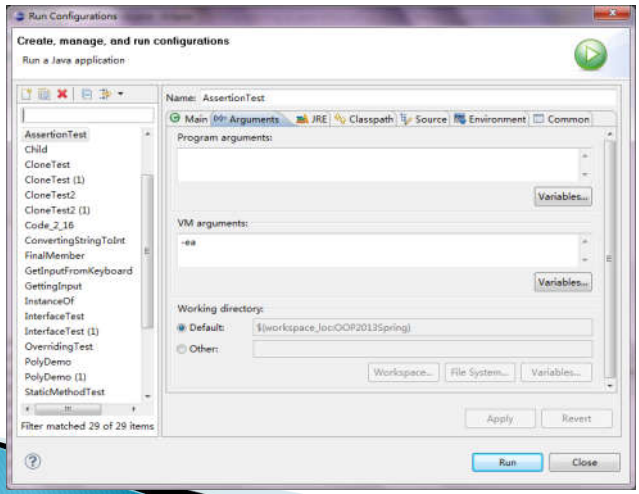
<terminated> AssertTest [Java Application] C:\Program Files\J

程序正常

53

Enable Assertion

- java -ea Myapp



54