Lecture 9

Exceptions

References:

1. Tony Gaddis, Chapter 11, Starting out with Java: From Control Structures through Objects, 7 edition
2. Herbert Schildt, Chapter 10, The Complete Reference Java 10 edition, McGraw Hill

1

Chapter Topics

- Handling Exceptions
- Throwing Exceptions

2

Handling Exceptions (1 of 3)

- C
  - No direct support for error or exceptional handling
  - Return a -1 or NULL in case of an error
  - if statement
- Python
  - Written as try/except statement

```
try:
    statements
except exceptionName:
    statements
```


3

Handling Exceptions (2 of 3)

- An exception is an object
  - As the result of an error or an unexpected event
  - Exception thrown
  - Unhandled exceptions crashing a program
  - Java (OOP) allows you to create exception handlers
- Example: BadArray.java

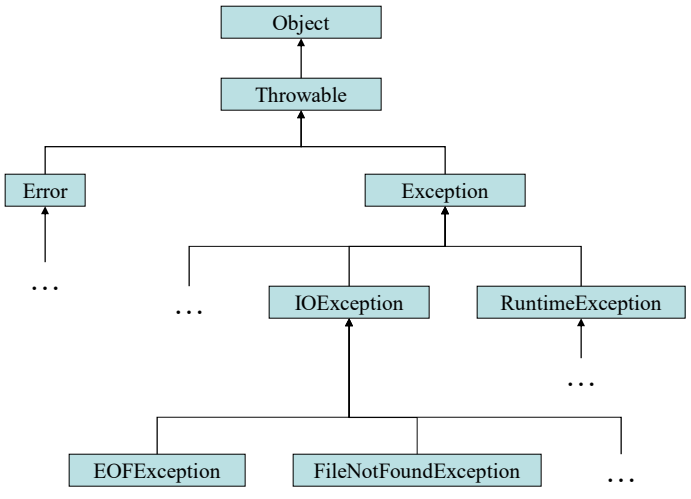
4

Handling Exceptions (3 of 3)

- Exception handling
  - Intercepting and responding to exceptions
  - Define exception handlers
- Default exception handler
  - Deals with unhandled exceptions
  - Prints an error message and crashes the program

5

Exception Classes



6

Exception Types

- Throwable
  - Built-in top-level class
- Exception
  - Used for exceptional conditions that user programs should catch
  - Using Exception, you can create your own customer exception types

7

Exception Types

- RuntimeException
  - Exceptions defined for the program
    - E.g., division by zero
    - E.g., invalid array indexing
- Error
  - Used by the Java run-time system
    - Related to the run-time environment itself
    - E.g., Stack Overflow
  - Created in response to catastrophic failures
    - Your program cannot handle

8

8

Handling Exceptions (1 of 3)

- To handle an exception, you use a *try* statement

```
try
{
   (try block statements...)
}
catch (ExceptionType ParameterName)
{
   (catch block statements...)
}
```

- The curly braces are required

9

Handling Exceptions (2 of 3)

- A *try block*
  - one or more statements that are executed
  - can potentially throw an exception
- A catch clause begins with the key word `catch`:

```
catch (ExceptionType ParameterName)
```

  - *`ExceptionType`* is the name of an exception class
  - *`ParameterName`* is a variable name which will reference the exception object

10

Handling Exceptions (3 of 3)

- Each exception object
  - `getMessage` method to retrieve the default error message for the exception
- Example:
  - ParseIntError.java

11

Handling Multiple Exceptions

- The code in the try block throwing more than one type of exception
- A `catch` clause needs each type of exception that could potentially be thrown
- The JVM will run the first compatible `catch` clause found
- The `catch` clauses must be listed from most specific to most general
- Example: MultipleCatches.java

12

Exception Handlers (1 of 3)

- Only one `catch` clause for each specific type of exception

```
try
{
  number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
  System.out.println("Bad number format.");
}
catch (NumberFormatException e) // ERROR!!!
{
  System.out.println(str + " is not a number.");
}
```


13

Exception Handlers (2 of 3)

- The `NumberFormatException` class derived from the `IllegalArgumentException` class

```
try
{
  number = Integer.parseInt(str);
}
catch (IllegalArgumentException e)
{
  System.out.println("Bad number format.");
}
catch (NumberFormatException e) // ERROR!!!
{
  System.out.println(str + " is not a number.");
}
```


14

Exception Handlers (3 of 3)

- The previous code rewritten to work:

```
try
{
  number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
  System.out.println(str + " is not a number.");
}
catch (IllegalArgumentException e) //OK
{
  System.out.println("Bad number format.");
}
```


15

The `finally` Clause (1 of 3)

- Optional `finally` clause

```
try
{
  (try block statements...)
}
catch (ExceptionType ParameterName)
{
  (catch block statements...)
}
finally
{
  (finally block statements...)
}
```


16

The `finally` Clause (2 of 3)

- The *finally block* is one or more statements,
  - that are always executed after the try block has executed and
  - after any catch blocks have executed if an exception was thrown.
- The statements in the finally block execute whether an exception occurs or not.

17

The `finally` Clause (3 of 3)

- An exception causes the method to return prematurely, e.g.,
  - A method opens a file on entry and closes it on exit
  - But a method may not close a file when an exception occurs
- Example, FinallyDemo.java

18

18

Multi-Catch (Java 7)

- Beginning in Java 7, you can specify more than one exception in a `catch` clause:

```
try
{
}
catch(NumberFormatException | InputMismatchException ex)
{
}
```

Separate the exceptions with the | character.

19

Uncaught Exceptions

- When an exception is thrown
  - Must be handled by the program, or by the default exception handler
- If there is no exception handler inside the method:
  - Control of the program passed to the previous method in the call stack.
- If control reaches the `main` method:
  - the main method must either handle the exception, or
  - the program is halted and the default exception handler handles the exception
- Example: StackTrace.java

20

Checked and Unchecked Exceptions (1/4)

- Checked and Unchecked exceptions
- Unchecked Exceptions
  - Compiler does not check to see if a method handles these exceptions
  - In Java, Error and *RuntimeException* classes are unchecked exceptions
    - E.g., ArithmeticException, ArrayIndexOutOfBoundsException
  - In C++, all exceptions are unchecked

Checked and Unchecked Exceptions (2/4)

- Checked Exceptions
  - must handle the exception, or
  - it must have a `throws` clause listed in the method header.
  - E.g., ClassNotFoundException, IllegalAccessException, …

Checked and Unchecked Exceptions (3/4)

```
// This method will not compile!
public void displayFile(String name)
{
   // Open the file.
   File file = new File(name);
   Scanner inputFile = new Scanner(file);
   // Read and display the file's contents.
   while (inputFile.hasNext())
   {
     System.out.println(inputFile.nextLine());
   }
   // Close the file.
   inputFile.close();
}
```


23

Checked and Unchecked Exceptions (4/4)

- The code in this method is capable of throwing checked exceptions
- The keyword `throws`

```
public void displayFile(String name)
    throws FileNotFoundException
```

- Example:
  - ThrowsDemo.java, ThrowsDemo1.java

24

Throwing Exceptions

- You can write code that:
  - throws one of the standard Java exceptions, or
  - an instance of a custom exception class that you have designed.
- The `throw` statement to manually throw an exception

```
throw new ExceptionType(MessageString);
```

- Example: ThrowDemo.java

25

Creating Exception Classes (1 of 3)

- Just define a subclass of Exception (a subclass of Throwable) for each of error conditions
- Exception defines constructors
  - Exception(), Exception(String msg), …
- Example: MyExceptionDemo.java

26

26

Creating Exception Classes (2 of 3)

- Example:
  - BankAccount.java
  - NegativeStartingBalance.java
  - AccountTest.java

27

Creating Exception Classes (3 of 3)

- Some examples of exceptions that can affect a bank account:
  - A negative starting balance is passed to the constructor.
  - A negative interest rate is passed to the constructor.
  - A negative number is passed to the deposit method.
  - A negative number is passed to the withdraw method.
  - The amount passed to the withdraw method exceeds the account's balance.
- We can create exceptions that represent each of these error conditions.

28

`@exception` Tag in Documentation Comments

- General format

  `@exception` *ExceptionName Description*

- The following rules apply
  - The `@exception` tag in a method's documentation comment must appear after the general description of the method.
  - The description can span several lines. It ends at the end of the documentation comment (the `*/` symbol) or at the beginning of another tag.

29