

Chapter 11

Exception Handling

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-1

Outline

- 11.1 What are Exceptions?
 - 11.1.1 Exception Handling
 - 11.1.2 The Delegation Model
- 11.2 Java's Exception Constructs
 - 11.2.1 The Basic try-catch Construct
 - 11.2.2 Handling Multiple Exceptions
 - 11.2.3 Other Constructs
- 11.3 Exception Objects
 - 11.3.1 The Throwable Hierarchy
 - 11.3.2 Object-Oriented Exception Handling
 - 11.3.3 Checked Exceptions
- 11.4 Building Robust Apps
 - 11.4.1 Validation versus Exception
 - 11.4.2 Logic Errors

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-2

11.1 What Are Exceptions?

An exception is an object that represents information about an error state that has arisen to the VM

Examples of error states:

- attempting to perform an illegal operation, such as:

input mismatch, divide by zero, invalid cast, ...

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-3

Example: The Quotient app

Given two integers, write a program to compute and output their quotient.

```
output.println("Enter the first integer:");
int a = input.nextInt();
output.println("Enter the second:");
int b = input.nextInt();

int c = a / b;
output.println("Their quotient is: " + c);
```

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-4

11.1 The important issues:

"Legal" Issue

If an exception is thrown by an implementer, was this part of its contract?

"Logistical" Issue

If an exception is thrown, what should the client do about it?

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-5

11.1 What Are Exceptions?

There are three sources that can lead to exceptions:

The End User

Garbage-in, garbage-out

The Programmer

Misunderstanding requirements and/or contracts

The Environment

The VM, the O/S, the H/W, or the network

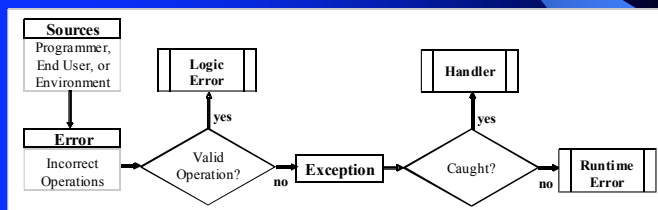
Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-6

11.1.1 Exception Handling

- An error source can lead to an **incorrect** operation
- An **incorrect** operations may be valid or **invalid**
- An **invalid** operation throws an **exception**
- An **exception** becomes a **runtime error** unless caught



Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-7

Example, cont.

Here is a sample run:

```

Enter the first integer:
8
Enter the second:
0

Exception in thread "main"
java.lang.ArithmeticException: / by zero
    at Quotient.main(Quotient.java:16)
    
```

In this case:

- The error source is the end user.
- The incorrect operation is invalid
- The exception was not caught

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-8

Example, cont.

Anatomy of an error message:

```

Enter the first integer:
8
Enter the second:
0
Exception in thread "main"
java.lang.ArithmeticException: / by zero
    at Quotient.main(Quotient.java:16)
    
```

Type Stack trace Message

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-9

11.1.2 The Delegation Model

- We, the client, delegate to method **A**
 - An invalid operation is encountered in **A**
 - **A** can either **handle it** or **delegate it**
 - If **A** handled it, no one would know
 - Not even the API of **A** would document this
 - Otherwise, the exception is delegated to **us**
- We can either **handle it** or **delegate it**
 - If we handle it, need to use try-catch
 - Otherwise, we delegate to the VM
- The VM's way of handling exceptions is to cause a **runtime error**.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-10

11.1.2 The Delegation Model

- We, the client, delegate to method **A**
 - **A** delegates to method **B**
 - An invalid operation is encountered in **B**
 - **B** can either **handle it** or **delegate it**
 - If **B** handled it, no one would know
 - Not even the API of **B** would document this
 - Otherwise, the exception is delegated to **A**
 - **A** can either **handle it** or **delegate it**
 - If **A** handled it, no one would know; otherwise it comes to us...
- We can either **handle it** or **delegate it**

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-11

The Delegation Model Policy:

Handle or Delegate Back

Note:

- Applies to all (components and client)
- The API must document any back delegation
- It does so under the heading: **"Throws"**

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-12

Example: SubstringApp

Given a string containing two slash-delimited substrings, write a program that extracts and outputs the two substrings.

```
int slash = str.indexOf("/");
String left = str.substring(0, slash);
String right = str.substring(slash + 1);
output.println("Left substring: " + left);
output.println("Right substring: " + right);
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-13

Example, cont.

Here is a sample run with `str = "14-9"`

```
int slash = str.indexOf("/");
String left = str.substring(0, slash);
String right = str.substring(slash + 1);
output.println("Left substring: " + left);
output.println("Right substring: " + right);
```

```
java.lang.IndexOutOfBoundsException:
String index out of range: -1
at java.lang.String.substring(String.java:1480)
at Substring.main(Substring.java:14)
```

The trace follows the delegation from line 1480 within `substring` to line 14 within the client.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-14

Example, cont.

Here is the API of `substring`:

```
String substring(int beginIndex, int endIndex)
Returns a new string that...
```

Parameters:

`beginIndex` - the beginning index, inclusive.
`endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-15

11.2.1 The basic try-catch

```
try
{
    ...
    code fragment
    ...
}
catch (SomeType e)
{
    ...
    exception handler
    ...
}
program continues
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-16

Example

Redo the last example with exception handling

```
try
{
    int slash = str.indexOf("/");
    String left = str.substring(0, slash);
    String right = str.substring(slash + 1);
    output.println("Left substring: " + left);
    output.println("Right substring: " + right);
}
catch (IndexOutOfBoundsException e)
{
    output.println("No slash in input!");
}
output.println("Clean Exit."); // closing
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-17

11.2.2 Multiple Exceptions

```
try
{ ...
}
catch (Type-1 e)
{ ...
}
catch (Type-2 e)
{ ...
}
...
catch (Type-n e)
{ ...
}
program continues
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-18

Example

Given a string containing two slash-delimited integers, write a program that outputs their quotient. Use exception handling to handle all possible input errors.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-19

Example

Given a string containing two slash-delimited integers, write a program that outputs their quotient. Use exception handling to handle all possible input errors.

Note that when exception handling is used, do not code defensively; i.e. assume the world is perfect and then worry about problems. This separates the program logic from validation.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-20

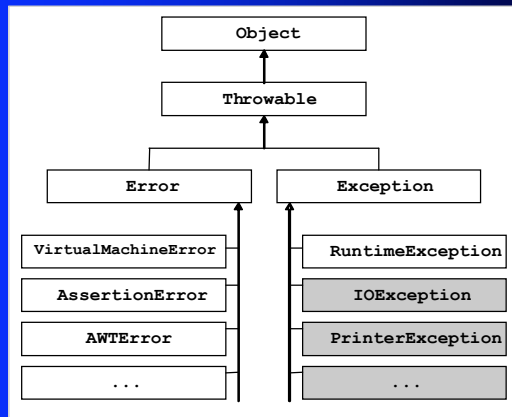
Example, cont.

```
try
{
    int slash = str.indexOf("/");
    String left = str.substring(0, slash);
    String right = str.substring(slash + 1);
    int leftInt = Integer.parseInt(left);
    int rightInt = Integer.parseInt(right);
    int answer = leftInt / rightInt;
    output.println("Quotient = " + answer);
}
catch (?)
{
}
```

Example, cont.

```
catch (IndexOutOfBoundsException e)
{
    output.println("No slash in input!");
}
catch (NumberFormatException e)
{
    output.println("Non-integer operands!");
}
catch (ArithmeticException e)
{
    output.println("Cannot divide by zero!");
}
output.println("Clean Exit."); // closing
```

11.3.1 The Hierarchy



11.3.2 OO Exception Handling

- They all inherit the features in **Throwable**
- Can create them like any other object:
`Exception e = new Exception();`
- And can invoke methods on them, e.g. `getMessage`, `printStackTrace`, etc.
- They all have a `toString`
- Creating one does not simulate an exception. For that, use the `throw` keyword:
`Exception e = new Exception("test");
throw e;`

Example

Write an app that reads a string containing two slash-delimited integers the first of which is positive, and outputs their quotient using exception handling. Allow the user to retry indefinitely if an input is found invalid.

As before but:

- What if the first integer is not positive?
- How do you allow retrying?

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-25

Example, cont.

```
for (boolean stay = true; stay;)
{
    try
    {
        // as before
        if (leftInt < 0) throw(??);
        ...
        output.println("Quotient = " + answer);
        stay = false;
    }
    // several catch blocks
}
```

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-26

Example, cont.

```
for (boolean stay = true; stay;)
{
    try
    {
        // as before
        if (leftInt < 0) throw(??);
        ...
        output.println("Quotient = " + answer);
        stay = false;
    }
    // several catch blocks
}
```

E.g. Runtime-Exception with a message

The order may be important

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-27

11.3.3 Checked Exceptions

- App programmers can avoid any RuntimeException through defensive validation
- Hence, we cannot force them to handle such exceptions
- Other exceptions, however, are "un-validatable", e.g. diskette not inserted; network not available...
- These are "checked" exceptions
- App programmers *must* acknowledge their existence
- How do we enforce that?
- The compiler ensures that the app either handles checked exceptions or use "throws" in its main.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 11-28

Example

Write a program that finds out the IP address of a given web server.

Hint: Use the `Socket` class (Lab 11)

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-29

11.4 Building Robust Applications

Key points to remember:

- Thanks to the compiler, **checked** exceptions are never "unexpected"; they are trapped or acknowledged
- **Unchecked** exceptions (often caused by the end user) must be avoided and/or trapped
- **Defensive programming** relies on validation to detect invalid inputs
- **Exception-based programming** relies on exceptions
- Both approaches can be employed in the same app
- Logic errors are minimized through early exposure, e.g. strong typing, assertion, etc.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

11-30