CSE1720

Week 11, Class Meeting 29 (Lecture 20)

Winter 2013 ◆ Thursday, March 21, 2013

YORK UNIVERSITÉ UNIVERSITY

Topics

- exception handling – Chapter 11

YORK UNIVERSITÉ UNIVERSITY

11.1 What Are Exceptions?

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

<u>Examples of error states:</u>

-attempting to perform an illegal operation, such as:

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

What is a *clean exit*?
What is a *crash*?

- A clean exit is when an app ends in a controlled and orderly manner
  - flush all output buffers
  - complete all pending transactions
  - close all network connections
  - free up all used resources
- A crash is a non-clean exit
  - abrupt termination
  - may be accompanied by error messages that do not originate from the program

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);
```


"Throwers" of exceptions

- methods (as per the post condition)
- arithmetic operators
  - integer division, integer modulo
  - not floating point division, floating point modulo
- Virtual Machine itself
  - memory is full

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?

Recap

- implementers offers services in the form of utility and non-utility classes
- we, as clients, make use of the services offered by implementers
  - utility classes are classes that cannot be instantiated; for utility classes to be useful, their methods and/or fields should be static
  - non-utility classes are classes that can be instantiated; the may include both non-static and static methods and/or fields
- the "terms and conditions of use" for services are described in the API
  - pre conditions
  - post condition (the specification of the return and/or the condition under which an exception is thrown)

Recap

- "no precondition" means pre is `true` (sec 2.3.3)
  - precondition is "the statement that the client should ensure is true as a condition of using this service"
  - if pre is `true`, then the client doesn't need to do anything
- "returns" and "throws" are parts of the post condition

substring

```
public String substring(int beginIndex)
```

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

Parameters:
`beginIndex` - the beginning index, inclusive.
Returns:
the specified substring.
Throws:
`IndexOutOfBoundsException` - if `beginIndex` is negative or larger than the length of this `String` object.

Ways to think about the "throws" section of the API…

✗ WRONG
- Exceptions are thrown as punishment to a client for violating the pre-condition.
- Thrown exceptions are like run-time errors: they are bad and a sign that something went wrong.

✓ CORRECT
- The API does not (should not) specify what happens if the precondition is not met.
- When the API specifies that an exceptions is thrown in a particular scenario, <u>this is part of the post condition</u>

11.1 What Are Exceptions?

There are three sources that can lead to exceptions:

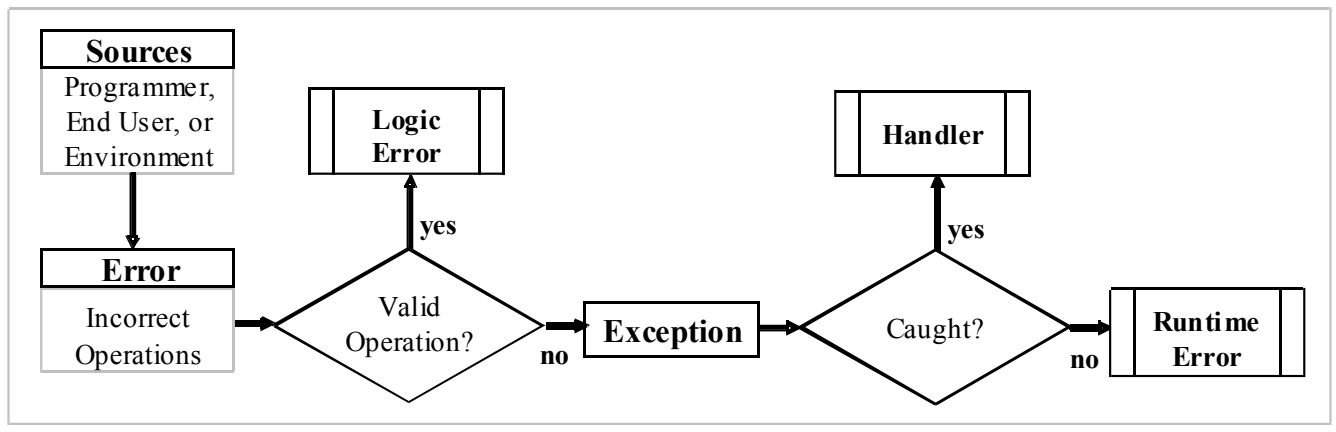
The End User
Garbage-in, garbage-out

The Programmer
Misunderstanding requirements and/or contracts

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

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



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**

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

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.

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