CSE 1720

Lecture 11
Exception Handling

Reminder

Midterm Exam
Thursday, Feb 16, 10-11:30
CLH J – Curtis Lecture Hall, Room J
will cover all material up to and including Tues Feb 14th

- Tues, Feb 7 – topic: exceptions
- Thurs, Feb 9 – Midterm overview, Recap, Review, Study preparation
- Tues, Feb 14 –valentine's day celebration of continued coverage of the topic of exceptions

Topics

- exception handling – Chapter 11

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, ...

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);
```


Throwing exceptions

- example `L11AppQuotient` demonstrates arithmetic operation throwing an exception
- example `L11App01` demonstrates arithmetic operation throwing an exception
- example `L11App02` demonstrates difference between int and double quotient/division and modulo operation, in terms of exception-throwing behaviours

"Throwers" of exceptions

- methods (as per the post condition)
- arithmetic operators
  - integer division, integer modulo
  - not floating point division, floating point modulo

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, this is part of the post condition

11.1 What Are Exceptions?

There are three sources that can lead to exceptions:

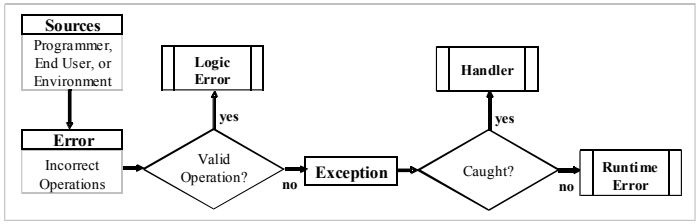
The End User
Garbage-in, garbage-out

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

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"

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);
```

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.

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.
```


11.2.1 The basic try-catch

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


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
```

Catching exceptions

- example `L11App01` demonstrates arithmetic operation throwing an exception
- example `L11App02` demonstrates difference between int and double quotient/division and modulo operation, in terms of exception-throwing behaviours
- example `L11App03` demonstrates basic try-catch block

11.2.2 Multiple Exceptions

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


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.

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.

Example, cont.

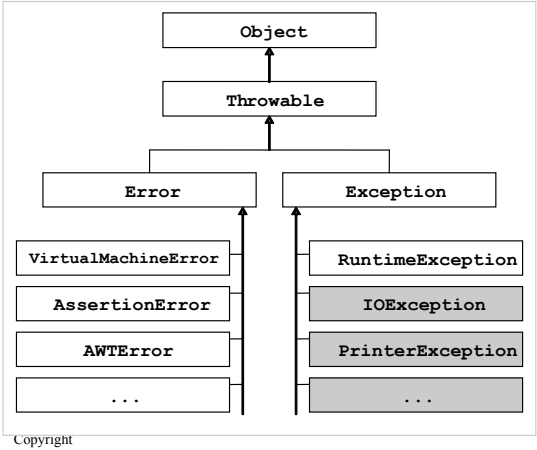
```
try
{
   int slash = str.indexOf("/");
   String left = str.substring(0, slash);
   String right = str.substring(slash + 1);
   int leftInt = Integer.parseInt(left);
   int rightInt = Integer.parseInt(right);
   int answer = leftInt / rightInt;
   output.println("Quotient = " + answer);
}
catch (?)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

}
```


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.

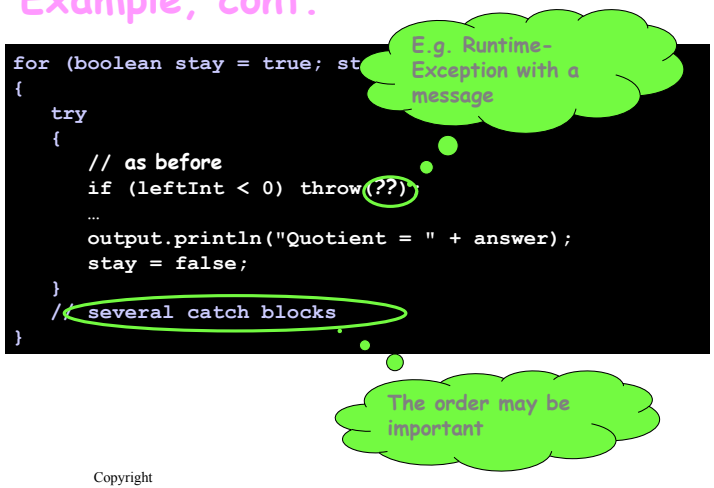
As before but:

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

Example, cont.

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


Example, cont.



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.

Example

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

Hint: Use the Socket class (Lab 11)

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.**