

Click to edit title

Second level

Third level


Fourth level

Fifth level

CSE1720

Week 11, Class Meeting 30 (Lecture 21)

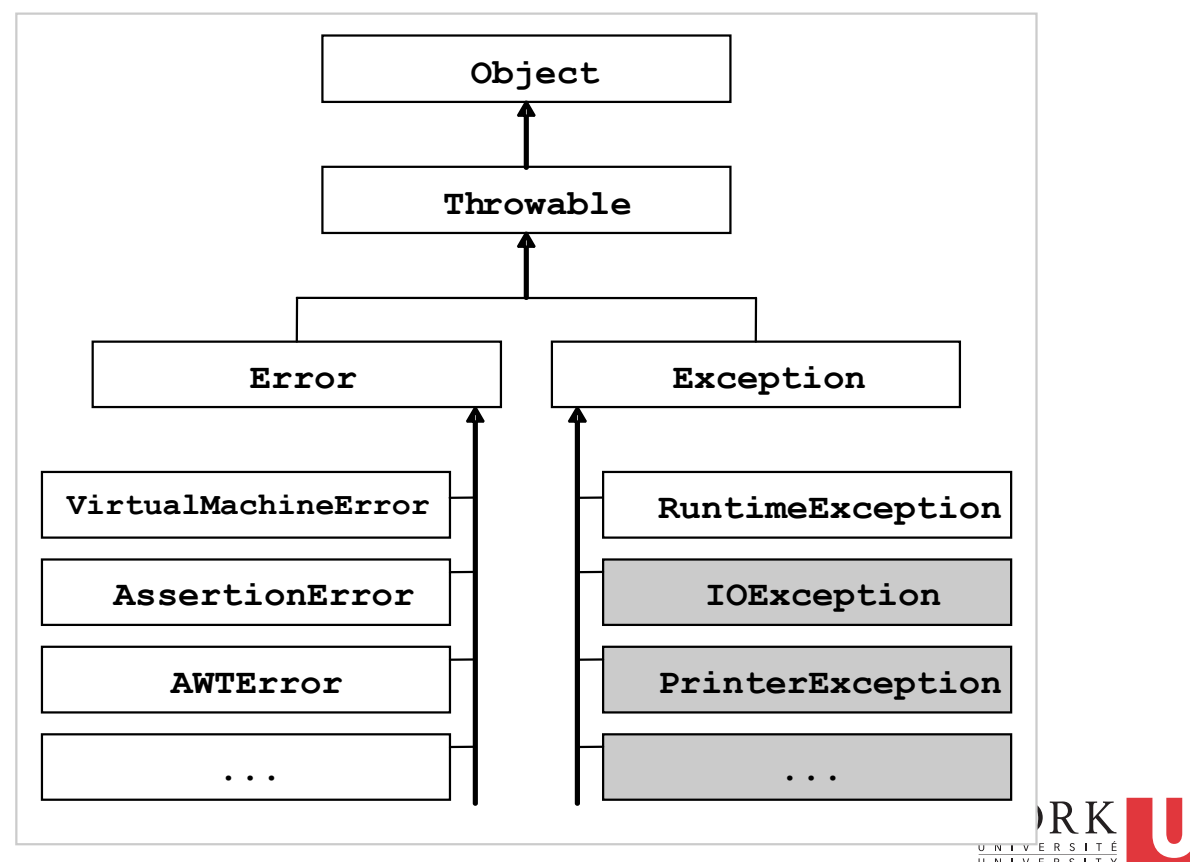
Winter 2013 ♦ Tuesday, March 26, 2013

YORK
UNIVERSITÉ
UNIVERSITY 

Topics

- exception handling – Chapter 11

11.3.1 The Hierarchy



3

Copyright ©

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**.

4

Copyright ©

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...

- 5 • **We** can either **handle it** or **delegate it**

Copyright ©



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**”

6

Copyright ©



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);
```

7

Copyright ©



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.

8

Copyright ©



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.
```

9

Copyright ©



11.2.1 The basic try-catch

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

10

Copyright ©



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
```

11

Copyright ©



11.2.2 Multiple Exceptions

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

12

Copyright ©



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.

13

Copyright ©



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.

14

Copyright ©



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 (?)
{
}
}
```

15

Copyright ©



Example, cont.

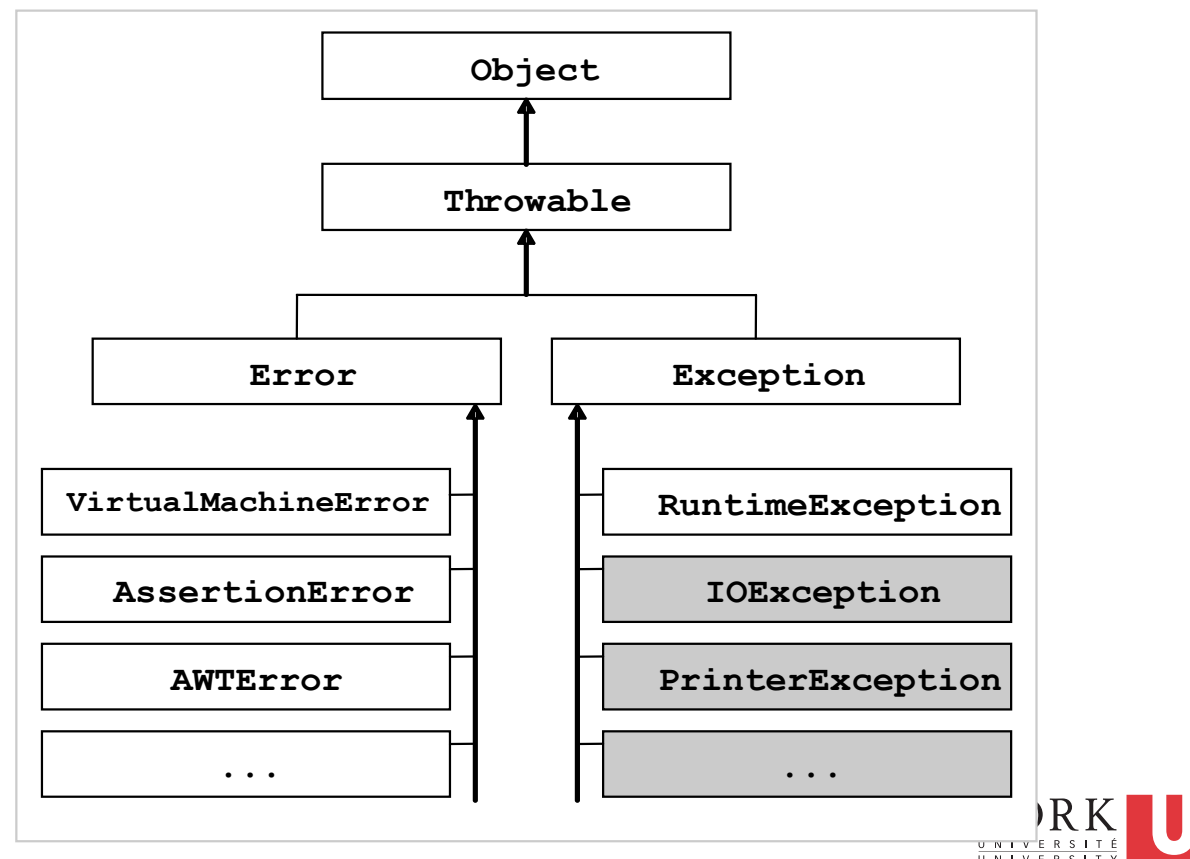
```
catch (IndexOutOfBoundsException e)
{
    output.println("No slash in input!");
}
catch (NumberFormatException e)
{
    output.println("Non-integer operands!");
}
catch (ArithmeticException e)
{
    output.println("Cannot divide by zero!");
}
output.println("Clean Exit."); // closing
```

16

Copyright ©



11.3.1 The Hierarchy



17

Copyright ©

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;
```

18

Copyright ©

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?

19

Copyright ©



Example, cont.

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

20

Copyright ©



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

21

Copyright ©



11.3.3 Checked Exceptions

- In the Exception hierarchy, there is an important distinction between
 - the branch of RuntimeExceptions, and
 - the other branches, such as IOException and ClassNotFoundException
- the exceptions of type RuntimeException, in principle, can be avoided through defensive programming
 - such exceptions can be prevented from arising through careful programming
 - app programmers cannot be “forced” to handle such exceptions

22

22



11.3.3 Checked Exceptions

- Other exceptions such as `IOException` and `ClassNotFoundException`, however, cannot be easily validated **even in principle**
 - how can the app realistically monitor, at all times, the state of resources upon which it depends, such as network availability, media availability, the file system, the functioning of the OS and the WM
 - these conditions are, in essence, “un-validatable” (this is a made-up word ☺)

23

11.3.3 Checked Exceptions

- For “un-validatable” exceptions, the compiler enforces the acknowledgement rule.
- The “un-validatable” exceptions are referred to as **checked** exceptions
 - App programmers *must* **acknowledge** their existence
 - The compiler ensures that the app either handles checked exceptions or use “**throws**” in its main.

24

Example

Write a program that opens a File, given a pathname, and then reads an object from that file

Hint: See L09App1 (reproduced as L12App01)

25

Copyright ©



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.

26

Copyright ©

