CS2136: Paradigms of Computation

Class 18: Java: Numbers Exceptions

Copyright 2001, 2002, 2003 Michael J. Ciaraldi and David Finkel
Numbers: Representation and Conversion
Number Representation

z All Java implementations are supposed to use the same internal representations for numbers.

y int = 32 bits signed binary.
x -(2^{31}) through (2^{31} - 1)
x -2147483648 through +2147483647

y long = 64 bits signed binary.
x -(2^{63}) through (2^{63} - 1)

y float = 32 bits.

y double = 64 bits.
float and double Representation

- Conform to IEEE 754 floating-point standard.
- Inspired by scientific notation.
- Besides numbers, can represent:
  - Positive infinity
  - Negative infinity
  - Not a Number (NaN)
What we really want is \( m \times 10^n \), but computers work in binary.

**float has:**
- 1 bit sign
- 23 bits mantissa
- 8 digits exponent

**double has:**
- 1 bit sign
- 52 bits mantissa
- 11 digits exponent
**float Representation**

The finite nonzero values of type float are of the form $s \times m \times 2^e$, where

- $s$ is +1 or -1,
- $m$ is a positive integer less than $2^{24}$,
- and $e$ is an integer between -149 and 104, inclusive.
**double Representation**

The finite nonzero values of type double are of the form $s \times m \times 2^e$, where:

- $s$ is +1 or -1,
- $m$ is a positive integer less than $2^{53}$,
- $e$ is an integer between -1075 and 970, inclusive.

<table>
<thead>
<tr>
<th>sign</th>
<th>mantissa</th>
<th>exponent</th>
</tr>
</thead>
</table>

7
How Does It Work?

z Convert the decimal number to the equivalent $s \times m \times 2^e$.

z The mantissa $(m)$ is usually scaled so the high-order bit is 1. This is “normalization”.

y Float example: instead of converting 131072 to $1 \times 2^{17}$, use $16384 \times 2^3$.

z Then offset the exponent.

z You can hold a number up to about $10^{308}$ in a double.
What’s the Problem?

- Can only hold so many significant figures.
  - float has about 9.
  - double has about 19.
- Many even decimal fractions are not even binary fractions.
  - $0.5 = 1 \times 2^{-1}$ exactly.
  - $0.1$ can only be approximated.
Exceptions
Exceptions

z Many errors cannot be caught at compile time.

z What should be done if the run-time system detects one at run time?
  y Run-time system ignores them: bad.
  y Crash the program: bad.
  y Return an error flag: common.
  y Generate an exception: the Java way.
Error Flags vs. Exceptions

Error flags:
- Enable fine-grained control.
- Require checking every statement.
- Are only supported for some errors.

Exceptions:
- Can centralize error handling.
- Can span multiple statements and multiple levels of the calling hierarchy.
- Can be either coarse- or fine-grained.
- Can treat all errors with the same mechanism.
Features of Exceptions

z Separates error handling code from what is done normally.
  y Makes code much more understandable.

z When an error occurs, an exception is thrown.

z Somewhere, the exception is caught, and then processed.
What To Do? Possibilities!

z Terminate the current context, handle the exception, then proceed after the handler.
  y This is what Java supports.

z Suspend the current context, handle the exception, then resume at the point where the exception occurred.
  y Sounds good, but rarely used in practice.
How Exceptions Work

z An exceptional situation has occurred.
z An exception (an object) is created and thrown.
  y throw new MyException();
  y throw new MyException("My exception has occurred");
How Exceptions Work II

z The exception is **propagated up the call chain**.

y To the run-time context above the current one, and so on.
   x i.e. to the method calling this one, the method calling that method, etc.

y Note: This does **not** propagate up the class hierarchy.
How Exceptions Work III

z Then, either:

y The exception is caught with a catch statement. --OR--

y The exception reaches main, where the interpreter prints out the message and a stack trace, then exits.
The “try” Statement

Ordinarily, a method just exits when an exception occurs.

And throws the exception, of course.

“try” says that you want Java to watch for an exception, not just pass it on.

Enclose the code in a try block (a.k.a. guarded region).

```java
try {
    // code which might generate exceptions
}
```
The “catch” Statement

- Says what to do when an exception arrives.
- Follows the try block.

```java
try {
    // code which might
    // generate exceptions
} catch (Exception e) {
    // exception handler
}
```

- Inside the brackets, can access e to see what the exception is.
Multiple catch Statements

If you want to treat different exceptions differently.

```
try {
    // code which might generate exceptions
} catch (SomeException e) {
    // exception handler
} catch (someOtherException e) {
    // exception handler
}
```
Multiple catch Statements II

- Exception-handling mechanism looks for first handler which matches the type of the exception.
- Any other exceptions just get passed on, up the call stack.
The Class
“java.lang.Throwable”

- Any object that can be thrown and caught is an instance of class Throwable, or one of its subclasses.
- Throwable has two subclasses.
Subclasses of Throwable

z Error: Serious errors you usually cannot recover from, so don’t try to catch.
  y AWTError
  y LinkageError
  y VirtualMachineError
  y ThreadDeath [Might try to catch this one.]

z Exception: Everything else.
  y We will concentrate on this.
The Class “Exception”

- Create with the constructors:
  - `new Exception()`  -OR—
  - `new Exception(String someString)`

- Some other methods:
  - `getMessage()`
  - `printStackTrace()`
  - `fillInStackTrace()`
Some Standard Exceptions

- CloneNotSupportedException
- DataFormatException
- InstantiationException
- IOException
- RuntimeException
- UserException
Some More Standard Exceptions

- Arithmetic Exception
- Array Store Exception
- ClassCastException
- IllegalArgumentException
- IndexOutOfBoundsException
- NullPointerException
Create Your Own Exceptions

- Extend the class Exception.
- Can add data fields.
- In your method, either catch the exception or declare that this method throws this exception.

```java
void f() throws someException, someOtherException { }
```
- When you override a method, you can only throw exceptions the base class could.
Rethrowing

z An exception handler can either:
  y Completely process the exception.
  y Do some processing, then rethrow the exception.
  y Do some processing, modify the exception with fillInStackTrace(), then rethrow.
  y Do some processing, then instantiate and throw a new exception.

z See sample code.
General Rules on Exceptions

z Detect exceptional conditions at the place where they can occur, then throw an Exception.

z Catch Exceptions where:
  y They can be handled properly.
  y You can recover from them.

z Usually you will not try to recover from the error at the same place it is first detected.
Next Time

z Swing and GUIs