

CSE 8B—Intro to CS: Java

Winter, 2006

January 17 (Day 3)
Exceptions

Instructor: Neil Rhodes

Administrative

Change in discussion section

- Monday 4-5 now on Friday 4-5

2

Dealing with errors

How can a method notify the caller that an error occurred?

- Return an error code
- Throw an *exception*

3

Returning an error code

```
int makeDinner() {
    int error;

    error = makeMainCourse();
    if (error == noError)
        error = makeSalad();
    if (error == noError)
        error = makeDessert();
    return error;
}
```

What if we have a long
chain of methods calling each
other?

Each must carefully pass
up error message from
methods it calls

```
int planEvening() {
    int error;

    error = makeDinner();
    if (error != noError)
        error = goOutToDinner();
    return error;
}

int makeDessert() {
    int error;

    error = preheatOven();
    if (error == noError)
        error = getOutIngredients();
    if (error == noError)
        error = getOutUtensils();
    if (error == noError)
        error = bakeInOven();
    // must turn off oven no matter
    // what!
    error = turnOffOven();

    return error;
}
```

4

Throwing exceptions

When some exceptional event happens, call

- Throw, passing an Exception object

which will:

- cause this method to exit *immediately*
- cause the method that called it to exit immediately
- ...
- until an *exception handler* is reached that will handle the exception
- If no exception handler is found, abort the program!

Where have we seen exceptions occur in our programming so far?

5

Throwing exceptions (first try)

```
void makeDinner() {
    makeMainCourse();
    makeSalad();
    makeDessert();
}
```

```
void planEvening() {
    try {
        makeDinnerr();
    }
    catch (Exception e){
        goOutToDinner();
    }
}
```

```
void makeDessert() {
    preheatOven();
    getOutIngredients();
    getOutUtensils();
    bakeInOven();
    turnOffOven();
}
```

```
void getOutUtensils() {
    ...
    if (cantFindMixingBowl())
        throw new Exception("no mixing
        bowl");
    ...
}
```

6

Throwing exceptions (second try)

```
void makeDinner()
throws Exception {
    makeMainCourse();
    makeSalad();
    makeDessert();
}
```

Methods must specify which exceptions they throw (with some exceptions we'll talk about later).

```
void planEvening() {
    try {
        makeDinnerr();
    }
    catch (Exception e){
        goOutToDinner();
    }
}
```

```
void makeDessert() throws
Exception {
    preheatOven();
    getOutIngredients();
    getOutUtensils();
    bakeInOven();
    turnOffOven();
}
```

```
void getOutUtensils() throws
Exception {
    ...
    if (cantFindMixingBowl())
        throw new Exception("no mixing
        bowl");
    ...
}
```

7

Throwing exceptions (third try)

```
void makeDinner()
throws Exception {
    makeMainCourse();
    makeSalad();
    makeDessert();
}
```

We need to turn off the oven if an exception occurs during makeDessert!

```
void planEvening() {
    try {
        makeDinnerr();
    }
    catch (Exception e){
        goOutToDinner();
    }
}
```

```
void makeDessert() throws
Exception {
    preheatOven();
    try {
        getOutIngredients();
        getOutUtensils();
        bakeInOven();
    } finally {
        turnOffOven();
    }
}
```

```
void getOutUtensils() throws
Exception {
    ...
    if (cantFindMixingBowl())
        throw new Exception("no mixing
        bowl");
    ...
}
```

8

finally

The finally allows cleanup to take place if an exception occurs within a method (directly or indirectly)

The exception that caused the finally clause to execute continues to be propagated

Structure:

```
▪ try {  
    code that could generate an exception  
}  
finally {  
    code to be executed whether an exception was generated or not  
}
```

Always clean up after yourself!

9

Advantages of Exceptions

The code for the normal case is clean and uncluttered

- Not littered with tests of error codes, ifs, etc.

An exception can be propagated through unaware methods

10

The structure of an Exception

An exception is an object

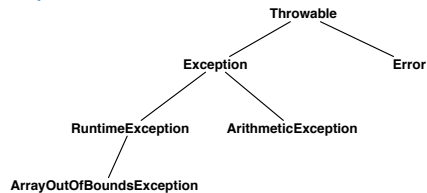
Every exception has:

- String returnable by getMessage()

Can have

- other data (as defined by the class)

Hierarchy of Exceptions



11

Can have multiple catch clauses

If an exception occurs, the list of catch clauses is executed to find the first one that qualifies

```
try {  
    int x = emptyArray[0];  
} catch (ArrayIndexOutOfBoundsException e) {  
    // executes if an ArrayIndexOutOfBoundsException is thrown  
} catch (Exception e) {  
    // executes if a RuntimeException (or any subclass except  
    // ArrayIndexOutOfBoundsException) is thrown  
}
```

12

Throws clauses

Part of a method signature is the list of exceptions the method could throw

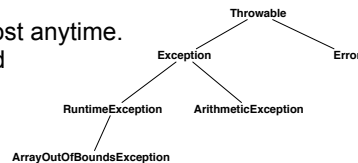
- Actually, there are some exceptions that need not be listed (*unchecked* exceptions)

Part of the contract of a method is not only what parameters it takes, and what it returns, but what exceptions it might throw

- Therefore, should be documented in the method signature
- Exception to this rule

Unchecked exceptions (Error or RuntimeException) need not be declared

- Cause they can happen almost anytime. Virtually every method would have to declare them.



13

Party line on Exception vs. RuntimeException

RuntimeException is for those things that indicate a programming error

- Can't be reasonably expected to recover

Error is for those things that indicate a problem with the Java Virtual Machine

- Can't load a class, for example

Other Exceptions are for things that a client could reasonably recover from

- Record not found in database
- Illegal employee ID number

If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.

--Sun documentation

14

Why declare?

If a client might reasonably recover, then the client needs to know what exceptions may be thrown.

15

Catch or declare

If you call a routine that could generate a (checked) exception, either:

- Catch the exception (and deal with it!)
- Declare it in the throws clause of your method signature.

Problem:

- Any time you call a method that declares that it throws an exception, if you don't care about the exception, you must add it to your throws clause
 - You can't just catch it and ignore it. Whoever called you might want to know about it!

16

Never, never, never!

```
public void foo(...) throws WierdException {
    ...
}

public void bar() {
    ...
    try {
        foo(...);
    } catch (WierdException e) {
        // I want to shut javac up and let me compile!
    }
}
```

17

An idea on Best Practices

1. What action can the client code take when the exception occurs?

- Nothing: make the exception unchecked
- Something: make the exception checked

2. Don't blindly pass on the same exception type from a lower-level to a higher-level

- For example, if makeDessert calls getOutUtensils (that throws MixingBowlMissingException), makeDessert should catch that exception and generate a new exception: dessertFailedException

<http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html>

18

An idea on Best Practices

3. Prefer unchecked exceptions for all programming errors

- Example: NullPointerException, IllegalArgumentException

4. Document exceptions

- With either javadoc (Rational.html, Complex.html)
- or, unit tests

5. Always clean up after yourself

6. Never use exceptions for flow control

- For example, break out of a loop by throwing an exception and catching it outside the loop

7. Do not suppress or ignore exceptions

8. Do not catch top-level exceptions

- Don't catch(Exception e)

9. Log exceptions just once

- catch(Exception e) { e.printStackTrace(); }

<http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html>

19

Testing exceptions with unit tests

Add an exception handler

- If the exception handler executes, the test should succeed, so call:
 - `assertTrue(true)`
- If the exception doesn't get thrown where it should, the the next line of code will be executed, which should cause the test to fail:
 - `fail("Should throw a ... exception");`

```
public void testIndexOutOfBoundsException() {

    int []emptyArray = new int[0];

    try {
        int x = emptyArray[0];
        fail("Should throw an ArrayIndexOutOfBoundsException");
    } catch (ArrayIndexOutOfBoundsException expected) {
        assertTrue(true);
    }
}
```

20

Programming assignment 2

Modeling complex numbers

- A complex number is a sum of a real number and an imaginary number:
 - $z = a + ib$
- where a and b are real numbers
 - In our assignment, a and b will be *rational* numbers (Rational.java provided to you).
- Note that i is defined as the square root of -1

Useful tutorial

- <http://www.tech.plym.ac.uk/maths/resources/PDFLaTeX/complex.pdf>