Dealing with errors

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

- Return an error code
- Throw an exception

Returning an error code

```java
int makeDinner() {
    int error;
    error = makeMainCourse();
    if (error == noError)
        error = makeSalad();
    if (error == noError)
        error = makeDessert();
    return error;
}
```

```java
int planEvening() {
    int error;
    error = makeDinner();
    if (error != noError)
        error = goOutToDinner();
    return error;
}
```

```java
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;
}
```

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

Each must carefully pass up error message from methods it calls.
### 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?

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

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

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

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

### Throwing exceptions (first try)

### Throwing exceptions (second try)

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

### Throwing exceptions (third try)

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

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

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

```
Throwable
  Exception
    Error
      RuntimeException
      ArithmeticException
      ArrayOutOfBoundsException
```

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
  }
```
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.

---

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

---

Why declare?

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

---

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!
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!
   }
}

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

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 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);
    }
}
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