Index of /~cs61b/lectures/13

Name	Last modified	Size	Description
Parent Directory	24-Sep-98 21:22	-	

```
                     CS61b: Lecture 13
                    Wednesday, September 23
ANNOUNCMENTS:
-------------
Midterm 1 is Monday from 10-11am, Last name A-M in 155 Dwinnelle, N-Z in Sibley.
Midterm 1 will cover up through lecture 12 (abstract classes and interfaces).
There will be a review session on Sunday evening at 6pm in 306 Soda.
Change in final grade weights (from general info handout, 200 points total):
  Project weights were originally: pj1 15, pj2 15, pj3 20, pj4 20
  Project weights will now be:     pj1 15, pj2 20, pj3 20, pj4 15
Change in lecture schedule: 3 more lectures on Java.

INTERFACES REVIEW:
------------------
  * An interface is like an abstract class, except
     * Only has methods signatures and constant, static fields.
     * All members are implicitly public.
  * A class may implement any number of interfaces (unlike abstract classes!!).
  * Cannot apply new to an interface; never any pointers to ‘bare’  interfaces.
  * An interface may extend any number of interfaces (super interfaces).

HANDLING EXCEPTIONS
-------------------
So far we have written lots of programs that:
  1) "assume" preconditions about the inputs to methods
  2) crash (hopefully with an error message) if they don’t
  3) return null/non-null or false/true depending on failure

There are really two kinds of errors in program:
  1) fatal ones: out of memory, bug in the program, etc.
  2) exceptional cases: bad input, array out of bounds

Java has a better way of handling the recoverable ones called "exceptions."
The basic idea:
  1) a method may either return normally (with a "good" return value)
  2) or it may raise an exception (and also return some value)

Example that we have seen:
   String input = br.readLine();

May throw an IOException.  Two styles of handling it:
  1) throw the exception out of this method too
  2) catch the exception and try to handle it

THROWS
------
For option 1, we simply add a "throws" to the method signature:
   public static int readInt1(BufferedReader br) throws IOException {
```

```
      String input = br.readLine();
      return Integer.parseInt(input);
   }
```

TRY/CATCH

Java construct for exception handling:

```
       try {
            Statements0
        } catch (E1 x) { Statements1 }
          catch (E2 x) { Statements2 }
        ...
```

What this does:

```
 1)  Execute Statements0.
 2)  If it completes normally, just go on (to ...).
 3)  If there is an exception thrown with: throw E1;
    in Statements0 or in the methods invoked in Statements0
     -> stop executing Statements0 where you are
     -> execute Statements1 and then continue with the ...
```

EXAMPLE:

```
  public static int readInt2(BufferedReader br) {
    String input;
    try {
      input = br.readLine().trim();
    } catch (IOException ioe) {
      input = "I got an exception!!";
    }
    System.out.println("You typed: " + input);
    return Integer.parseInt(input);
  }
```

Exceptions may carry values, e.g., a String explaining what happened:

```
   input = "I got an exception: " + ioe.getMessage() + "!!";
```

The type being thrown doesn't have to exactly match the type in the catch:
  "throw E1" matches a "catch E2" if E1 is a subclass of E2

EXCEPTION TYPES

Exceptions are instances of exception types.

  * Exception types are subclasses of Throwable.
  * Most exception types are subclasses of Exception, which extends Throwable.

```
   class BadExpressionException extends Exception {
      public BadExpressionException() { super(); }
      public BadExpressionException(String s) { super(s); }
   }
```

CHECKED VS. UNCHECKED EXCEPTIONS
----
1) IOException and most you will define yourself must be caught or rethrown.
2) Some, like the NumberFormatException thrown by parseInt, need not.
   The program will simply "crash" with the uncaught exception.
   These inherit from "Error" or "RuntimeException".

THROWING EXCEPTIONS

Throwing an exception is done with a "throw" statemtent.

```
        throw new BadExpressionException();
        throw new BadExpressionException("Missing right paren.");
```

Exceptions throwable from a method must be listed in the function header,
except for exception types that are subclasses of RuntimeException and Error.

```
   class Expression {
       public Expression parse() throws BadExpressionException
         { /* Body of parse */ }   }
```

HANDLING EXCEPTIONS

If you invoke a method that "throws" some exceptions, you may

1) Catch the exception and handle it
```
           try { S0 } catch (NoSuchStudent s) { add(s); finish S0; }
```
2) Catch the exception and remap it to another one by throwing that
```
           try { S0 }
              catch (NoSuchStudent s) { throw new ExpressionOverflow(s); }
```
3) Declare the exception in your own "throws" class, which will let it propagate
```
          void myMethod () throws NoSuchStudent {
               ... call any methods that throw NoSuchStudent ...   }
```