

CS1007 lecture #9 notes

tue 1 oct 2002

- news
 - homework #2 due today
 - short quiz #1 back today
 - no class on thu (inaguration)
- methods
- method overloading
- keyboard input
- introduction to recursion
- reading: *ch 4.7-4.13, ch 2.6*

methods — declaring them.

- like a variable, has:
 - data type:
 - * primitive data type, or
 - * class
 - name (i.e., identifier)
- also has:
 - arguments (optional)
 - * also called *parameters*
 - * *formal parameters* are in the blueprint, i.e., the method declaration
 - * *actual parameters* are in the object, i.e., the run-time instance of the class
 - throws clause (optional)
(*we'll defer discussion of this until later in the term*)
 - body
 - return value (optional)

methods — using them.

- program control jumps inside the body of the method when the method is *called* (or *invoked*)
- arguments are treated like local variables and are initialized to the values of the calling arguments
- method body (i.e., statements) are executed
- method *returns* to calling location
- if method is not of type *void*, then it also *returns* a value
 - return type must be the same as the method's type
 - calling sequence (typically) sets method's return value to a (local) variable; or uses the method's return value in some way (e.g., a print statement)

object relationships.

- are hierarchical
- example:

```
java.lang.Object
|
+-- java.lang.Number
    |
    +-- java.lang.Integer
```
- *is-a* relationship
 - an object that is an instance of a class
 - an *Integer* is a *Number*, which is a *Object*
 - children *inherit* properties of their parents; formally called *inheritance*
- *has-a* relationship
 - if an object declares data whose type is also a class

method overloading.

- using the same method name with formal parameters of different types

- example:

- `java.lang.System` has a variable called `out`
- which is a `java.io.PrintStream`
- whose declarations include:

```
public void println();
public void println( boolean x );
public void println( char x );
public void println( char[] x );
public void println( double x );
public void println( float x );
public void println( int x );
public void println( long x );
public void println( Object x );
public void println( String x );
```

keyboard input — ch 2.6

- the book uses a *package* called *tio*
- a *package* is a group of related *classes*
- we will use two classes from this package (right now):
 - `ReadInput`
 - `ReadException`
- the code is here: <http://www.columbia.edu/~cs1007/examples>

3 hello programs — hello.java

```
public class hello {
    public static void main( String[] args ) {
        System.out.println( "hello world\n" );
    } // end of main()
} // end of hello class
```

3 hello programs — hello1.java

```
public class hello1 {
    public static void main( String[] args ) {
        if ( args.length < 1 ) {
            System.err.println( "usage: java hello1 <person's name>" );
            System.exit( 1 );
        } // end if
        System.out.println( "hello "+args[0]+"!\n" );
    } // end of main()
} // end hello1 class
```

3 hello programs — hello2.java

```
public class hello2 {
    public static void main( String[] args ) {
        System.out.print( "who would you like to say hello to? " );
        ReadInput input = new ReadInput( System.in );
        String line = input.readLine();
        System.out.println( "hello "+line+"!\n" );
    } // end of main()
} // end hello2 class
```

recursion.

- recursion is defining something in terms of itself
- there are many examples in nature
- and in mathematics
- and in computer graphics, e.g., the Koch snowflake (textbook, p.485)

power function.

- *power* is defined recursively: $x^y = \begin{cases} \text{if } y == 0, & x^y = 1 \\ \text{if } y == 1, & x^y = x \\ \text{otherwise,} & x^y = x * x^{y-1} \end{cases}$

here it is in a Java method.

```
• public int power ( int x, int y ) {
    if ( y == 0 ) {
        return( 1 );
    }
    else if ( y == 1 ) {
        return( x );
    }
    else {
        return( x * power( x, y-1 ) );
    }
} // end of power() method
```

- Notice that `power()` calls itself!
- You can do this with any method *except* `main()`
- BUT beware of infinite loops!!!
- You have to know when and how to stop the recursion — what is the *stopping* condition

let's walk through `power(2,4)`.

	call	x	y	return value
1	<code>power(2,4)</code>	2	4	<code>2 * power(2,3)</code>
• 2	<code>power(2,3)</code>	2	3	<code>2 * power(2,2)</code>
• 3	<code>power(2,2)</code>	2	2	<code>2 * power(2,1)</code>
• 4	<code>power(2,1)</code>	2	1	2

- the first is the *original call*
- followed by three *recursive calls*