Object Intro and Miscellaneous

Checkout *ObjectIntroAndMisc* project from SVN

Help from Peers

- Having a peer help you with some strange bug or specific problem – Great Idea!
- Discussing your approach to a problem with a peer – still OK
- Letting a peer copy your code/Emailing code to a peer – NEVER OK
- Every person has a unique code style, it's easy to tell when two sets of code are too similar

Q1

```
/**
 * Has a static method for computing n!
 * (n factorial) and a main method that
 * computes n! for n up to Factorial.MAX.
 *
 * @author Mike Hewner & Delvin Defoe
 */
public class Factorial {
    /**
     * Biggest factorial to compute.
     */
    public static final int MAX = 17;

    /**
     * Computes n! for the given n.
     *
     * @param n
     * @return n! for the given n.
     */
    public static int factorial (int n) {
        ...
    }

    ...

}
```

Javadoc comments

Java provides Javadoc comments (they begin with /**) for both:

- Internal documentation for when someone reads the code itself
- External documentation for when someone re-uses the code

Comment your own code now, as indicated by this example. Don't forget the @author tag in HelloPrinter.

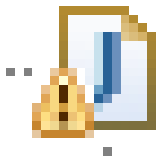
Q2 - 3

Writing Javadocs

- Written in special comments: /** … */
- Can come before:
  - Class declarations
  - Field declarations
  - Constructor declarations
  - Method declarations
- Eclipse is your friend!
  - It will generate Javadoc comments automatically
  - It will notice when you start typing a Javadoc comment

In all your code:

- See http://www.rose-hulman.edu/class/csse/csse220/201720/Homework/programGrading.html
- Write appropriate comments:
  - Javadoc comments primarily for classes.
  - Explanations of anything else that is not obvious in any spot.
- Give self-documenting variable and method names:
  - Use name completion in Eclipse, Ctrl-Space, to keep typing cost low and readability high
- Use Ctrl-Shift-F in Eclipse to format your code.
- Take care of all auto-generated TODO's.
  - **Then delete the TODO comment**.
- Correct ALL compiler warnings. Quick Fix is your friend!



Debugging—Key Concepts

- Breakpoint
- Single stepping
- Inspecting variables

Q4

Debugging—Demo

- Debugging Java programs in Eclipse:
  - Launch using the debugger
  - Setting a breakpoint
  - Single stepping: *step over* and *step into*
  - Inspecting variables
- Complete **WhackABug** exercise

Identifiers (Names) in Java

- The rules:
  - Start with letter or underscore (_)
  - Followed by letters, numbers, or underscores

- The conventions:
  - `variableNamesLikeThis`
  - `methodNamesLikeThis(…)`
  - `ClassNamesLikeThis`
- You should follow the conventions!

Class – What, When, Why, & How?

What:

- A blueprint for a custom **type**

When:

- Define a class when you're representing a concept (think nouns)
- When no other existing type can do what you want/need

Class – What, When, Why, & How?

Why:

- Keep similar concepts together
- Encapsulation (we'll expand on this next time)

How:

```
public class ClassName {
    //fields
    //methods
}
```

Constructors – What, When, Why, How?

What:

- Special method called when a new instance of a class is created
- Initializes the new instance
- Like the __init__ method in Python

When:

- Define a constructor when special initialization of a class is required
- Otherwise, Java implicitly creates a no-argument constructor if you don’t add one

Constructors – What, When, Why, How?

Why:

- Allows you to ensure that a new instance of a class is a setup exactly how it needs to be before use of other methods/fields
- Puts it in a good state

How:

```
public class MyClass {
    public MyClass() {
        //initialization code
    }
    public MyClass(ParamType paramName) {
        //initialization code
    }
}
```

Object Constructors

- int num = 5;
  - This works for primitive typed data

- What about “objects” (made from classes)?

Using Constructors

In Java, all variables must have a type

The constructor arguments specifies that the new rectangle called box should be at the origin with a height and width of 5.

```
Rectangle box = new Rectangle(0, 0, 5, 5);
```

Every variable must have a name.

The new operator is what actually makes the new object, in this case a new rectangle.

Q6

Object Constructors

- Every “object” must be created
  - How do we create them?
- Open ObjectConstructorPractice.java
  - Let’s do the first couple of TODOs together
- On your own: Try creating a variable of the String class using a constructor (in the main method somewhere).

new Keyword– What, When, Why, How?

What:

- Used to create a new instance of a class
- Calls the constructor in the class

When:

- Creating a new instance of a class
  - If the class definition is the blueprint for the house, a house that has been built is the “new instance” of the blueprint.

new Keyword– What, When, Why, How?

Why:

- To make a new instance

How:

- MyClass instance = new MyClass();
  - This will call the constructor with the matching parameters in MyClass
- Also used for arrays (as we've seen before):
  - int[] arr = new int[5];

Using Objects and Methods

- Works just like Python:
  - *object.method(argument, ...)*

"Who does what, with what?"

Implicit argument

Explicit arguments

The dot notation is also used for *fields*

- Java Example:

```
String name = "Bob Forapples";
PrintStream printer = System.out;

int nameLen = name.length();
printer.printf("'%s' has %d characters", name, nameLen);
```

Implementing classes

- Live coding with Bank Account object

Now code the StudentAssignments class yourself

- Uncomment the stuff in StudentAssignmentsMain to see what the class ought to do
- Then create the class and add the constructors and methods you need
- If you finish early, add a function to compute the student's average grade