Specifications
Via Interfaces and Commenting

Notes on Programming Practice and Software Quality

Specifications

- Specifications are descriptions
  - Sometimes of requirements or intentions or desired properties
  - Sometimes of actual properties or situations
- Specifications can take various forms
- Here we look at just two forms:
  - Java interfaces
  - Program comments

Interfaces in Java

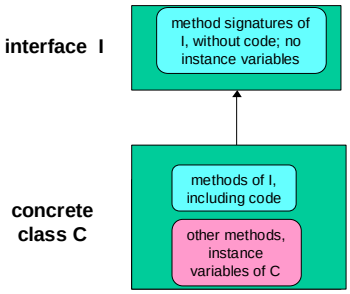
- A Java *interface* declares a set of method signatures
  - i.e., says what behavior exists
- Does not say how the behavior is implemented
  - i.e., does not give code for the methods
- Does not describe state
  - Exception – may describe permanent, unchangeable state
    "final" constants which never change

Interface Syntax in Java

- Looks similar to a class definition:

```
interface BankAccountInterface {
  double getBalance( );
  double deposit(double amount );
  }
```

- A concrete class that implements an interface
  - Contains "implements *InterfaceName*" in the class declaration
  - Must provide implementations of all methods declared in the interface
- Once an interface has been defined, you could define multiple classes which implement it (have the specified behavior)



Interfaces as Specifications

- Managers (or colleagues or teachers...) often say, "your class has to implement at least the following 3 methods, named just as I tell you and with these exact parameters:"
  1.
  2.
  3.
- The manager may not care much *how* you write the code, as long as it meets the interface specification
- Why define a Java interface and use "implements"?
  - Java will check that you did in fact follow the specification
  - Compile error if you leave out a required method
  - Compile error if you don't have the right return type, parameters types, etc.

Limitations of Interfaces as Specifications

- Interfaces only tell the form or syntax of the behavior

```
void withdrawFunds(double amount);
```

  - Interface tells us what the name of the method is, the return type, parameters
  - Does not tell us what a valid amount is (can it be negative?)
  - Does not tell us what a reasonable amount is (can it be a billion dollars?)
  - Does not tell us what should happen if the account has insufficent funds

Comments as Specifications

- Comments are directed toward human readers
- Can describe meaning and intention in great detail

JavaDoc

- Java provides a clean way of including documentation as part of the source code – JavaDoc comments
  - Begin with /** and end with */
- Can be automatically formatted to produce documentation pages
  - Command-line tool is part of the Sun SDK
  - Built-in support in most IDE's including BlueJ, Eclipse; not in DrJava (yet)
- Universally used and understood by Java programmers
- Writing JavaDoc comments is part of the *practice* of programming

JavaDoc Tags

- Special tags to control formatting
  - @author – specify author
  - @version – version number, date, etc.
  - @param – description of a method parameter
  - @return – description of a non-void method result
  - Others (links, see also, …), plus can use arbitrary html
- Used to produce all online Java API documentation

Programming Practice vs Bare Requirements

- A Java program will compile equally well with or without comments
- Using them is a matter of proper practice and procedure
- In every aspect of life, there are commonly understood practices, procedures, conventions, and rituals
  - Understood by all, followed by most
  - Usually not required in any legal sense
- Programming is no exception
  - Learning about and following good practices is a part of this course

    And part of the homework grading!

Another Good Practice

- Place a static method in each class, just for testing it.
  - No special name; could even be main().
  - Even simple tests are helpful
  - Run the test method every time the class is modified

```
/** A method to test out some of the BankAccount operations */
public static void test() {
  BankAccount myAccount = new BankAccount("Joe Bob");
  myAccount.deposit(100.00);
  myAccount.deposit(250.00);
  myAccount.withdraw(50.00);
  System.out.println(myAccount);  // automatically calls myAccount.toString()
 }

} // end of BankAccount
```

Required vs. Recommended

- Writing comments is "recommended"
- Creating test methods is "recommended"
- You've probably been given other recommendations:
  - variable naming, indentation, etc.
  - Use this library, don't use that library
- Why bother, when the only thing that matters is whether your program runs or not?
  - Answer #1: Whether your program runs or not is *not* the only thing that matters!
  - Answer #2: Recommended practices have been shown to increase software quality

Coupling and Cohesion

- Evaluating the quality of program structure is difficult.
- The following are two properties which measure important aspects of program structure:
- ***Coupling*** – the degree to which a class interacts with or depends on another class
- ***Cohesion*** – how well a class encapsulates a single notion
- Experience shows: a system is more robust and easier to maintain if
  - Coupling between classes/modules is minimized
  - Cohesion within classes/modules is maximized

Software Engineering and Practice

- Building good software is not just about getting it to produce the right output
- Many other goals may exist
- "Software engineering" refers to practices which promote the creation of good software, in all its aspects
  - Some of this is directly code-related: class and method design
  - Some of it is more external: documentation, style
  - Some of it is higher-level: system architecture
- Attention to software quality is important in CSE143
  - as it is in the profession