ECE 122
Engineering Problem Solving with Java
Lecture 17
Method Development and Testing
Outline

° Problem: How can I handle more complicated method design

° Splitting up problems into a series of steps
  • Algorithms

° What gets passed between the methods
  • Method overloading

° Software testing
Method Design

- As we've discussed, high-level design issues include:
  - identifying primary classes and objects
  - assigning primary responsibilities

- After establishing high-level design issues,
  - Important to address low-level issues such as the design of key methods

- For some methods, careful planning is needed to make sure they contribute to program design
Method Design

- An algorithm is a step-by-step process for solving a problem
- Examples: a recipe, travel directions
- Every method implements an algorithm that determines how the method accomplishes its goals
- An algorithm may be expressed in pseudocode
  - An outline of a program, written in a form that can easily be converted into real programming statements

Sorting

- while not at end of list
- compare adjacent elements
- if second is greater than first
- switch them
- get next two elements
- if elements were switched
- repeat for entire list
Method Decomposition

° A method should be relatively small, so that it can be understood as a single entity

° A potentially large method should be decomposed into several smaller methods as needed for clarity

° A public service method of an object
  • May call one or more private support methods to help it accomplish its goal

° Support methods might call other support methods if appropriate
Objects as Parameters

- Another important issue related to method design involves parameter passing
- Parameters in a Java method are passed by value
- A copy of the actual parameter (the value passed in) is stored into the formal parameter
  - In the method header
- Passing parameters is similar to an assignment statement
- When an object is passed to a method, the actual parameter and the formal parameter become aliases of each other
Passing Objects to Methods

° What a method does with a parameter may or may not have a permanent effect (outside the method)

° Difference between changing the internal state of an object versus changing the object reference

° For primitives: copy of data value passed

° For arrays: reference passed

° For objects: reference passed

° For arrays and other objects, data change is permanent
Method Overloading

- *Method overloading* is the process of giving a single method name multiple definitions.

- The method name is not sufficient to determine which method is being called.

- The *signature* of each overloaded method must be unique.

- The signature includes the number, type, and order of the parameters.
Method Overloading

- The compiler determines which method is being invoked by analyzing the parameters

```java
float tryMe(int x)
{
    return x + .375;
}

float tryMe(int x, float y)
{
    return x*y;
}
```

Invocation

```
float tryMe(int x)
{
    return x + .375;
}
```

```
float tryMe(int x, float y)
{
    return x*y;
}
```
The `println` method is overloaded:

- `println (String s)`
- `println (int i)`
- `println (double d)`

and so on...

The following lines invoke different versions of the `println` method:

```java
System.out.println ("The total is:");
System.out.println (total);
```
Overloading Methods

° The return type of the method is **not** part of the signature

° That is, overloaded methods cannot differ only by their return type

° Constructors can be overloaded

° Overloaded constructors provide multiple ways to initialize a new object
  
  • The correct parameter list must be specified when the object is created
Testing

- Testing can mean many different things
- It certainly includes running a completed program with various inputs
- It also includes any evaluation performed by human or computer to assess quality
- Some evaluations should occur before coding even begins
- The earlier we find an problem, the easier and cheaper it is to fix
Testing

- The goal of testing is to find errors
- As we find and fix errors, we raise our confidence that a program will perform as intended
- We can never really be sure that all errors have been eliminated
- So when do we stop testing?
  - Conceptual answer: Never
  - Snide answer: When we run out of time
  - Better answer: When we are willing to risk that an undiscovered error still exists
Reviews

° A *review* is a meeting in which several people examine a design document or section of code

° It is a common and effective form of human-based testing

° Presenting a design or code to others:
  • makes us think more carefully about it
  • provides an outside perspective

° Reviews are sometimes called *inspections* or *walkthroughs*
Test Cases

° A test case is a set of input and user actions, coupled with the expected results

° Often test cases are organized formally into test suites which are stored and reused as needed

° For medium and large systems, testing must be a carefully managed process

° Many organizations have a separate Quality Assurance (QA) department to lead testing efforts
Defect and Regression Testing

- *Defect testing* is the execution of test cases to uncover errors

- The act of fixing an error may introduce new errors

- After fixing a set of errors we should perform *regression testing*
  - Running previous test suites to ensure new errors haven't been introduced

- It is not possible to create test cases for all possible input and user actions

- Therefore we should design tests to maximize their ability to find problems
Black-Box Testing

- In *black-box testing*, test cases are developed without considering the internal logic.
- They are based on the input and expected output.
- Input can be organized into *equivalence categories*.
- Two input values in the same equivalence category would produce similar results.
- A good test suite will cover all equivalence categories and focus on the boundaries.
White-Box Testing

- *White-box testing* focuses on the internal structure of the code
- The goal is to ensure that every path through the code is tested
- Paths through the code are governed by any conditional or looping statements in a program
- A good testing effort will include both black-box and white-box tests
Summary

• Method design is critical to successful program implementation
  • Can’t be too complicated but must cover basics

• Becoming familiar with method design is really a matter of practice
  • Learning through doing

• Method overloading is an important feature of OOP
  • Allows for different choices in using methods

• Whole courses cover software testing.
  • This course primarily examines using the debugger in DrJava