Principles of Software (CSCI 2600)
Fall 2017

www.cs.rpi.edu/~milanova/csci2600/

Ana Milanova, Lally 314, milanova@cs.rpi.edu
Office hours: Tuesdays and Fridays 4:00-5:00PM
Mailing list (includes instructor, TAs and mentors):
csci2600@cs.lists.rpi.edu

Outline

Logistics
www.cs.rpi.edu/~milanova/csci2600/

Goals and topics

Tools

Java (for C++ programmers!)

Logistics

Recommended books


Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison Wesley, 1995

Refactoring: Improving the Design of Existing Code by Martin Fowler, Addison Wesley, 1999

Logistics

Java Resources

Main Java website by Oracle: http://java.com
Java documentation:
http://docs.oracle.com/javase/
1.7 JDK: http://docs.oracle.com/javase/7/docs/api/
Java tutorial:
http://docs.oracle.com/javase/tutorial/
Java language specification:
http://docs.oracle.com/javase/specs/
Café au Lait: Java FAQ, News, and Resources

Syllabus

www.cs.rpi.edu/~milanova/csci2600/syllabus.htm
Topics, outcomes, policies and grading
10 in-class quizzes: 10%
2 midterm exams and a final exam: 50%
10 homework assignments: 40%
5% extra credit for attendance and participation
Logistics

- All assignments must be completed individually!
  - Principles of Software builds individual skills
  - Carry these skills to collaborative projects
  - Submitty has advanced plagiarism detection tools. We check against this and past semesters
- EXCEPTION: HW0

Academic Integrity

- Homework assignments must be completed individually
  - Discussion is allowed and encouraged, but carrying material out of discussion is not allowed
- We trust you
- But…,
- Cheating is extremely easy to catch
- We will not tolerate it
- This policy does not apply to HW0

Goals

- Principles of Software is about how to write correct and maintainable programs
- What does it mean for a program to be correct?
  - Specifications
- What are ways to achieve correctness?
  - Principled design and development
  - Abstraction and modularity
  - Documentation

Logistics

- Homework will be released and turned in through version control (Git)
  - You will checkout/clone your repository to obtain csci2600 project and hw0
  - To turn in a homework, commit/push into your repository then click submit in Submitty
  - To obtain new homework, update/pull your repo
  - More on version control, JUnit in a short while
- Install Eclipse and Subclipse (plugin that interfaces with Git, SVN, other VCS)

Late Homework

- Homework assignments must be submitted in Submitty by 2pm on the due date
- You have 5 late days for the semester, with a max of 2 late days per assignment
- Exceptions to policy only in emergency sanctioned by your CLASS dean

Goals

- What are ways to verify correctness?
  - Reasoning about code, verification
  - Testing
    - Debugging follows successful testing
Goals

What does it mean for a program to be *maintainable*?
- Well-documented and understandable
- "Open for extension but *closed* for modification"
  - Canonical example: We have an editor that manipulates shapes. We have coded Square and Circle and have written tons of code that manipulates Squares and Circles. It should be "easy" to add a Triangle — i.e., there should be only minimal change to the code that manipulates shapes

Goals

What are ways to *achieve maintainability*?
- Object-orientation and *polymorphism* greatly facilitate this goal
- Principled design and development
- Abstraction and modularity
- Documentation

Goals

Building good software is incredibly hard!
- Large software systems are enormously complex. Lots of "moving parts"!
- Software is constantly put to new uses sometimes without relevant experience!
- Software engineering is about:
  - Mitigating and managing complexity
  - Managing change
  - Dealing with software failures

Topics

- Reasoning about code
- Invariants
- Specifications
- Polymorphism, abstraction and modularity
- Design patterns
- Testing and debugging
- Refactoring
- GUIs, UI design, Software process
- Tools: Java, the wealth of Java libraries, Eclipse IDE, Git, JUnit, debuggers, testing coverage tools, other
  - Principles are more important than the tools!!

Topics

- You will learn a lot!
- You will carry what you learn into
  - SD&D - 4000-level software engineering class
    - Focuses on teamwork, software process, requirements
  - RCOS
  - Research projects
  - Internships and jobs

Outline

- Logistics
- Goals and topics
- Tools
- Overview of Java (for a C++ programmer)
Tools
- Java
- Eclipse: an Integrated Development Environment (IDE)
- Git: Version Control
- Subclipse: an Eclipse plugin, connects to Git/SVN from your Eclipse project
- JUnit: a testing framework for Java
- Submitty

Version Control
- Version control systems
  - Record changes to a set of files over time
  - Manage changes by multiple users, or by single user working on multiple machines
  - Revert to older version, review changes made over time, track all changes, review who introduced issues
- We will be using Git

Version Control
- Checkout/clone set of files, aka repository, on local machine. Should be done once
- Add files to version control
  - E.g., you create problem4.txt locally; you should add it to version control
  - With Eclipse/Subclipse, no need to worry about it
- Commit/Push files (push changes to repo)
  - E.g., you’ll edit Ball.java, then you’ll commit
- Update/Pull (pull changes from repo)

JUnit
- A unit testing framework for Java
- Supports writing and running unit tests
- Aside: what is unit testing?
  - Scope of testing is one unit
    - E.g., subroutine, class
    - In object-oriented programming, the smallest testable unit is the class. Unit testing is class testing
  - Followed by integration testing, then system testing
JUnit 4.x
- Uses annotations to guide test run
  - @BeforeClass --- static method to configure test run, framework runs it before all tests
    - Creates an instance of the class under test
  - @Test --- annotation marks a method as test method, JUnit framework runs this method
- Test methods
  - assertEquals(message, expected result, actual expression)
  - assertTrue(message, boolean expression)

JUnit Example
```java
class SaleTest {
    private static Sale sale = null;
    private static double ITEM_PRICE = 2.5;

    @BeforeClass
    public static void setupBeforeTests() {
        sale = new Sale();
    }

    @Test
    public void testGetTotal() {
        sale.makeLineItem("item1", 1, ITEM_PRICE);
        sale.makeLineItem("item2", 2, ITEM_PRICE);
        assertEquals("sale.getTotal()", 7.5, sale.getTotal());
    }
}
```

JUnit
- Study the JUnit tests in your homework
  - Understand the annotations
  - Understand the different assert methods
  - Why the “tolerance” argument of assertEquals
  - What is the difference between an Error and a Failure?

JUnit Example
```java
public class SaleTest {
    private static Sale sale = null;
    private static double ITEM_PRICE = 2.5;

    @BeforeClass
    public static void setupBeforeTests() {
        sale = new Sale();
    }

    @Test
    public void testGetTotal() {
        sale.makeLineItem("item1", 1, ITEM_PRICE);
        sale.makeLineItem("item2", 2, ITEM_PRICE);
        assertEquals("sale.getTotal()", 7.5, sale.getTotal());
    }
}
```

JUnit Example
```java
public class SaleTest {
    private static Sale sale = null;
    private static double ITEM_PRICE = 2.5;

    @BeforeClass
    public static void setupBeforeTests() {
        sale = new Sale();
    }

    @Test
    public void testGetTotal() {
        sale.makeLineItem("item1", 1, ITEM_PRICE);
        sale.makeLineItem("item2", 2, ITEM_PRICE);
        assertEquals("sale.getTotal()", 7.5, sale.getTotal());
    }
}
```

Unit Testing
- Modern software development methodologies such as Extreme Programming (XP), Unified Process (UP) place great emphasis on unit testing
  - They advocate test-driven development (TDD)
    - Also known as test-first development
  - Key point: developer writes the unit test first, imagining the class that is tested

Test-driven Development
- Key point: write tests first
  - The unit tests actually get written!
  - Programmer satisfaction leading to more consistent test writing
  - Clarification spec and interface behavior
  - Repeatable, automated verification
  - Confidence to change things!

Follow www.cs.rpi.edu/~milanova/csci2600/handouts/Setup.html to set up infrastructure and get HW0

Note: as of now, Sept. 1st, 2pm, repos are not available. I hope we'll have everything set up over the weekend.
Outline

- Logistics
- Goals and topics
- Tools
- Overview of Java (for a C++ programmer)

Java

- It helps if you have experience with Java
- … If not, you can pick up
- ASK Questions!
- What are some important differences with C++?

Java: Differences with C++

- Model for variables
- Type safety
- Compilation vs. interpretation
- Other: classes and inheritance, reflection
- Other

Models for Variables

- Value model for variables
  - A variable is a location that holds a value
  - i.e., a named container for a value
    - \( a := b \)
    - \( b \) - l-value (the location)
    - \( c \) - r-value (the value held in that location)
- Reference model for variables
  - A variable is a reference to an object which has value
  - Every variable is an l-value
    - Requires dereference when r-value needed (usually, but not always implicit)

Models for Variables: Example

- Value model for variables
  1. \( b := 2 \)
     - \( b \): 2
  2. \( c := b \)
     - \( c \): 2
  3. \( a := b + c \)
     - \( a \): 4
- Reference model for variables
  1. \( b := 2 \)
     - \( b \)
  2. \( c := b \)
     - \( c \)
  3. \( a := b + c \)
     - \( a \)

Questions

- What is the model for variables in C/C++?
  - Value model
- Python?
  - Reference model
- Java?
  - Mixed model:
    - Value model for variables of simple type (e.g., int, float)
    - Reference model for variables of class type (e.g., String)
Models for Variables

- This has different meaning in C++ and in Java.

... Foo p; // p is a local variable
double d = p.bar();
...

Equality Testing: == and equals()

- Java uses the reference model for class types.

```java
class 2DPoint {
    int x; // x-coordinate
    int y; // y-coordinate
    2DPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }
    } a = new 2DPoint(2,5);
b = new 2DPoint(2,5);
c = b;
true or false? a == b ?
true or false? b == c ?
true or false? a.equals(b) ?
true or false? b.equals(c) ?
```

Equality of Strings

- The String class implements equals.
- When testing strings for equality, use equals(), not == !!!

```java
String a = new String("Ana");
String b = new String("Ana");
a == b is ??
a.equals(b) is ??
```
**Pointer Types**

- In C/C++, we need pointers
  - To allocate memory dynamically on the heap
  - To define recursive types (types defined in terms of themselves) such as linked lists. Think why.

- In Java, references _are_ pointers
  - A reference (address) can be on the stack or on the heap, referred object is always on the heap
  - Defining recursive types is easy

---

**Types and Type Safety**

- What is the role of types?
  - Data abstraction
  - Safety!

- Next time: type safety, reasoning about code