

Principles of Software (CSCI 2600) Fall 2015

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

Ana Milanova, Lally 314, milanova@cs.rpi.edu
Office hours: Tuesdays and Fridays 4:00-5:00PM

David Goldschmidt, Lally 209, goldschmidt@gmail.com
Office hours: Mondays 10:00-10:50AM, Tuesdays
12:00-1:45PM, Thursdays 10:00-11:50AM

Outline

- Logistics
www.cs.rpi.edu/~milanova/csci2600/
- Goals and topics
- Tools
- Java (for C++ programmers!)

Fall 15 CSCI 2600, A Milanova

2

Logistics

- Course webpage
<http://www.cs.rpi.edu/~milanova/csci2600>
 - **Announcements** – check regularly
 - **Schedule, Notes, Reading**
 - Schedule, lecture slides and assigned reading
 - **Homework**
 - Announces when new homework assignment is on
 - **RPILMS**
 - Discussion board, your grades

Fall 15 CSCI 2600, A Milanova

3

Logistics

- Recommended books
 - **Effective Java**, 2nd Edition, by Joshua Bloch, Addison Wesley, 2008
 - **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

Fall 15 CSCI 2600, A Milanova

4

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](#)

5

Logistics

- 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

Fall 15 CSCI 2600, A Milanova

6

Logistics

- All assignments must be completed individually!
 - Principles of Software builds **individual** skills
 - Carry these skills to collaborative projects
- **EXCEPTION: HW0**

Fall 15 CSCI 2600, A Milanova

7

Logistics

- Homework will be released and turned in through Subversion (SVN)
 - Checkout your repository to obtain **csci2600** project and **hw0**
 - To turn in a homework, **commit** into your repository then **submit** in Homework Server
 - To obtain a homework, **update** your repository
 - More on SVN, JUnit in just a short while
- Install Eclipse and Subclipse (plugin that interfaces with SVN)

8

Academic Integrity

- Homework assignments must be completed **individually**
 - Discussion is allowed and strongly 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**

Fall 15 CSCI 2600, A Milanova

9

Late Homework

- Homework assignments must be **submitted in Homework Server by 2pm** on the due date
- You have **5 late days** for the semester, with a max of **2 late days** per assignment
 - If you need to take a late day (or two) you must email us one hour ahead of the deadline
- Exceptions to policy only in emergency

Fall 15 CSCI 2600, A Milanova

10

Goals

- Principles of Software teaches you 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!

Fall 15 CSCI 2600, A Milanova

11

Goals

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

Fall 15 CSCI 2600, A Milanova

12



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 no changes to the code that manipulates shapes.

Fall 15 CSCI 2600, A. Milanova

13



Goals

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

Fall 15 CSCI 2600, A. Milanova

14



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

Fall 15 CSCI 2600, A. Milanova

15



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, Subversion, JUnit, debuggers, testing coverage tools, other
 - Principles are more important than the tools!!!

16



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

Fall 15 CSCI 2600, A. Milanova

17



Outline

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

Fall 15 CSCI 2600, A. Milanova

18

Tools

- **Java**
- **Eclipse**: an Integrated Development Environment (IDE)
- **Subversion (SVN)**: Version Control
- **Subclipse**: an Eclipse plugin, connects to SVN from your Eclipse project
- **JUnit**: a testing framework for Java
- **Homework Server** --- many thanks to Prof. Cutler!

19

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 Subversion (SVN)

Fall 15 CSCI 2600, A Milanova

20

Version Control

- **Checkout** 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** files (push changes to repository)
 - E.g., you'll edit Ball.java, then you must commit
- **Update** (pull changes from repository)

Fall 15 CSCI 2600, A Milanova

21

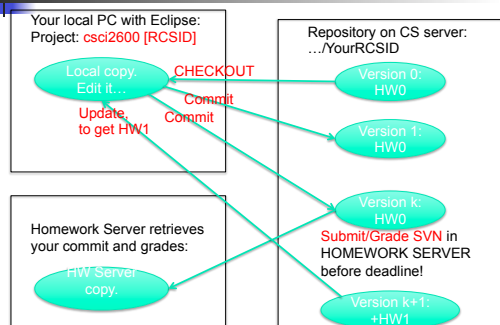
Version Control

- **Merges** happen when multiple users have modified the same file
 - Sometimes, conflicts happen
 - Rule: always update before commit!
- If you have a single copy in Eclipse (which I expect), you should not have to worry about merges and conflicts!
 - If you have a conflict, **notify us**

Fall 15 CSCI 2600, A Milanova

22

Homework



23

JUnit

- A **unit testing** framework for Java
 - Supports writing and running unit tests
- Aside: what is **unit testing**?
 - Scope of testing is one unit (also called **module** or component)
 - 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

Fall 15 CSCI 2600, A Milanova

24

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)

25

JUnit Example

Convention: Name of the class under test plus suffix Test. This is a test of class Sale

```
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());
    }
}
```

Fall 15 CSCI 2600, A Milanova

26

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?

Fall 15 CSCI 2600, A Milanova

27

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

Fall 15 CSCI 2600, A Milanova

28

Test-driven Development

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

Fall 15 CSCI 2600, A Milanova

29

Follow

www.cs.rpi.edu/~milanova/csci2600/handouts/Setup.html

to set up software infrastructure and HW0

Fall 15 CSCI 2600, A Milanova

30

Outline

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

Fall 15 CSCI 2600, A. Milanova

31

Java

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

Fall 15 CSCI 2600, A. Milanova

32

Java: Differences with C++

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

Fall 15 CSCI 2600, A. Milanova

33

Models for Variables

- Value model for variables
 - A variable is a location that holds a value
 - I.e., a named container for a value
 - $a := b$
 - l-value (the location)
 - 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)

Fall 15 CSCI 2600, A. Milanova

34

Models for Variables: Example

- Value model for variables
 - $b := 2$ $b: \boxed{2}$
 - $c := b$ $c: \boxed{2}$
 - $a := b + c$ $a: \boxed{4}$
- Reference model for variables
 - $b := 2$ $b \rightarrow \boxed{2}$
 - $c := b$ $c \rightarrow \boxed{2}$
 - $a := b + c$ $a \rightarrow \boxed{4}$

Fall 15 CSCI 2600, A. Milanova

35

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)

Fall 15 CSCI 2600, A. Milanova

36

Models for Variables

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

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

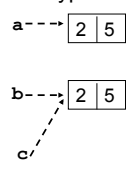
Fall 15 CSCI 2600, A Milanova

37

Equality Testing: == and equals()

- Java uses the reference model for class types. 2 ways to test equality

```
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)` ?

38

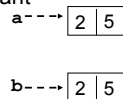
Equality Testing: == and equals()

- In Java, `==` tests for **reference equality**. This is the strongest form of equality

- Often, we need a weaker form of equality, **value equality**



- In our `2DPoint` example, we want `a` to be "equal" to `b` because the `a` and `b` objects hold the same value



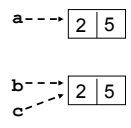
Fall 15 CSCI 2600, A Milanova

39

Equality Testing: == and equals()

- What happened in the `2DPoint` example?

true or false? `a == b` ?
 true or false? `b == c` ?
 true or false? `a.equals(b)` ?
 true or false? `b.equals(c)` ?



2DPoint inherited `Object.equals`:

```
public boolean equals(Object o) {
    return this == o;
}
```

- Override** `Object.equals` to get value equality!
- More on this later in the class

40

Equality of Strings

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

```
String a = new String("Ana");
String b = new String("Ana");
a == b is False
a.equals(b) is True
```

Fall 15 CSCI 2600, A Milanova

41

Equality Testing

- In languages with reference model for variables, we have two equality tests
 - One tests **reference equality**, whether two references refer to the same object
 - `==` in Java. E.g., `x == y`
 - `is` in Python. E.g., `print x is y`
 - Other tests **value equality**. Even if the two references do not refer to the same object, the distinct objects can have the same value
 - `.equals()` in Java
 - `==` in Python

42



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
 - The reference (address) can be on the stack or on the heap, referred object **is always on the heap**
 - Defining recursive types is easy

43



Types and Type Safety

- What is the role of types?
 - Data abstraction
 - Safety!
- Next time: type safety, reasoning about code

Fall 15 CSCI 2600, A Milanova

44