# COMP 413: Intermediate Object-Oriented Programming

## Dr. Robert Yacobellis
## Advanced Lecturer
## Department of Computer Science

LOYOLA
UNIVERSITY
CHICAGO

# Getting Help for COMP 413

- The TA for this class is ...
  - **Sapna Bohra**, luc email address sbohra@luc.edu
  - She will announce any office hours and availability shortly

- You <u>may</u> be able to find additional CS tutors at:
  - http://luc.edu/cs/people/tutors/
  - ... depends on your current level of Java and Android / Android Studio knowledge vs. theirs

**LOYOLA UNIVERSITY CHICAGO**

# Week 1 Topics

- **Course overview**
- Motivation
- System and Software Requirements
  - Functional vs. nonfunctional
  - Testing, including test-driven development (TDD)
- Course Software
- Projects 1 & 2
- Prerequisite Assessment

LOYOLA UNIVERSITY CHICAGO

# What is This Course All About?

- **Taking your software development abilities to the next level beyond simple data structures**

- **Learning to design and implement more complex programs using good software engineering practices:**

  ◦ Designing with interfaces and composition
  ◦ Design patterns
  ◦ Agile development: Refactoring, Test-Driven Development
  ◦ Application of these practices to Android development

LOYOLA UNIVERSITY CHICAGO

# Course Textbooks

- Agile Principles, Patterns, and Practices in C# (**APPPC**) by Robert C. Martin and Micah Martin, Prentice Hall, © 2007, ISBN-13: 9780131857254 – **our primary textbook**; publisher's site
  - **Can also be rented on Amazon for the semester** – see course home page
- Object Oriented Programming Using Java (**OOPUJ**) by Simon Kendal, available online for free at bookboon.com
- Principles of OOD (online, compiled by Robert Martin)
- Coad: Design with Interfaces; Coad: Archetypes and Color
  - These two documents are in **Sakai** → **Course Content** → **General Resources**
- Two free online multi-part guides to using Android Studio:
  - http://i-programmer.info/programming/android/5887.html
  - https://examples.javacodegeeks.com/android/android-tutorial-beginners-with-examples/

Other recommended texts at http://laufer.cs.luc.edu/teaching/313/textbooks

LOYOLA UNIVERSITY CHICAGO

# Course Overview – 1 of 2

- **Weeks 1-3: Foundations**
  - ◦ Motivation and introduction
  - ◦ TDD (Test-Driven Design) & JUnit
  - ◦ Interfaces & composition, other Java review topics
  - ◦ Generics & collections; Object-inherited methods
  - ◦ **Projects 1 & 2**

- **Weeks 4-9: Agile Development and OO Design**
  - ◦ Agile development process, testing, & refactoring
  - ◦ Continuous integration and delivery
  - ◦ Design principles – SOLID, designing with interfaces
  - ◦ Agile OO modeling – UML, archetypes and colors
  - ◦ **Quizzes 1 & 2 (covering SE Radio podcasts) and Test 1**

LOYOLA UNIVERSITY CHICAGO

# Course Overview – 2 of 2

- **Weeks 10-12: Design Patterns**
  - Abstract Factory & Factory
  - Prototype & Composite
  - Observer & Visitor
  - Others from APPP and Head First Design Patterns
  - **Test 2**

- **Weeks 13-16: Concurrent & Distributed Programming**
  - Events, threads, and sharing
  - Distributed programming overview and principles
  - Connecting to Web Services
  - **Quiz 3 and Tests 3 & 4 (final)**

LOYOLA
UNIVERSITY
CHICAGO

# Sakai and Piazza

- **Material for this course will be hosted on Sakai as well as other server-based repositories like Bitbucket**

- **There is a link on Sakai to the Piazza discussion forum for this class – you are required to subscribe to Piazza and to participate in discussions there**
  - **You can use Piazza to ask questions, resolve issues, etc.**

- **There is also a link to Panopto, a recording system that I <u>may</u> use to record class lectures – if so, you will be able to find those recordings shortly after classes**

LOYOLA
UNIVERSITY
CHICAGO

# Week 1 Topic: Motivation

- **Embedded systems – batch vs. event-based programs**
  - **http://www.embeddedsystem.com/**
  - **http://en.wikipedia.org/wiki/Embedded_system**
  - **http://www.embedded.com/**
  - **Embedded systems overview**

- **What makes software good?**
  - **http://en.wikipedia.org/wiki/Software_quality**
  - **http://en.wikipedia.org/wiki/Requirements_analysis**

LOYOLA UNIVERSITY CHICAGO

# Week 1 Topic: Requirements

- **Requirements – functional vs. nonfunctional**
  - **http://en.wikipedia.org/wiki/Non-functional_requirement**
  - **Functional vs. nonfunctional requirements overviews**

- **The importance of testing**
  - **http://www.examiner.com/article/importance-of-testing-software-development**
  - **Testable requirements material**

- **Test-Driven Development (TDD) and Junit**
  - **http://en.wikipedia.org/wiki/Test-driven_development**

LOYOLA UNIVERSITY CHICAGO

# And Now for Something Completely Different ... Test-Driven Development (TDD)

"Only ever write code to fix a failing test." That's test-driven development, or TDD,[1] in one sentence. First we write a test, then we write code to make the test pass. Then we find the best possible design for what we have, relying on the existing tests to keep us from breaking things while we're at it. This approach to building software encourages good design, produces testable code, and keeps us away from over-engineering our systems because of flawed assumptions. And all of this is accomplished by the simple act of driving our design each step of the way with executable tests that move us toward the final implementation.

Test first, then code, and design afterward. Does the thought of "designing afterward" feel awkward? That's only natural. It's not the same kind of design we're used to in the traditional design-code-test process. In fact, it's such a different beast that we've given it a different name, too. We call it *refactoring* to better communicate that the last step is about transforming the current design toward a better design. With this little renaming operation, our TDD cycle really looks like that in figure 1.4: *test-code-refactor*.
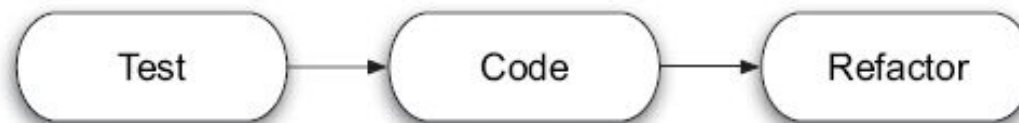
**Note: we must reuse our tests to make sure the code <u>continues</u> to work after the refactoring!**

Test → Code → Refactor

**Figure 1.4   Test-code-refactor is the mantra we test-driven developers like to chant. It describes succinctly what we do, it's easy to spell, and it sounds cool.**

Source: Koskela: Test Driven: Practical TDD and Acceptance TDD for Java Developers

LOYOLA UNIVERSITY CHICAGO

‹#›

# Why Do Test-Driven Development?

- **There are two major quality-related problems in development: high defect rates and lack of maintainability.**

### The challenge: solving the right problem right

The function of software development is to support the operations and business of an organization. Our focus as professional software developers should be on delivering systems that help our organizations improve their effectiveness and throughput, that lower the operational costs, and so forth.

Even after several decades of advancements in the software industry, the quality of the software produced remains a problem. Considering the recent years' focus on time to market, the growth in the sheer volume of software being developed, and the stream of new technologies to absorb, it is no surprise that software development organizations have continued to face quality problems.

There are two sides to these quality problems: high defect rates and lack of maintainability.

LOYOLA UNIVERSITY CHICAGO

# Why Do Test-Driven Development?

- **TDD can be used to solve both of these problems**

## Solution: being test-driven

Just like the problem we're facing has two parts to it—poorly written code and failure to meet actual needs—the solution we're going to explore in the coming chapters is two-pronged as well. On one hand, we need to learn how to build the thing right. On the other, we need to learn how to build the right thing. The solution I'm describing in this book—being test-driven—is largely the same for both hands. The slight difference between the two parts to the solution is in *how* we take advantage of tests in helping us to create maintainable, working software that meets the customer's actual, present needs.
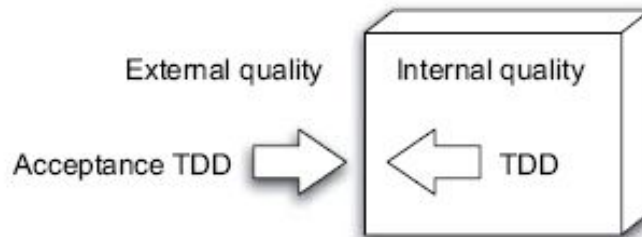
External quality          Internal quality

Acceptance TDD ⇨          ⇦ TDD

**Figure 1.1**
**TDD is a technique for improving the software's internal quality, whereas acceptance TDD helps us keep our product's external quality on track by giving it the correct features and functionality.**

## What's in it for me?

- I rarely get a support call or end up in a long debugging session.
- I feel confident in the quality of my work.
- I have more time to develop as a professional.

LOYOLA UNIVERSITY CHICAGO

# TDD and API Design

- **Writing tests first is key to good program API design**

When we write a test in the first step of the TDD cycle, we're really doing more than just writing a test. We're making design decisions. We're designing the API— the interface for accessing the functionality we're testing. By writing the test before the code it's testing, we are forcing ourselves to think hard about how we want the code to be used. It's a bit like putting together a jigsaw puzzle. As illustrated by figure 1.5, it's difficult to get the piece you need if you don't know the pieces with which it should connect.
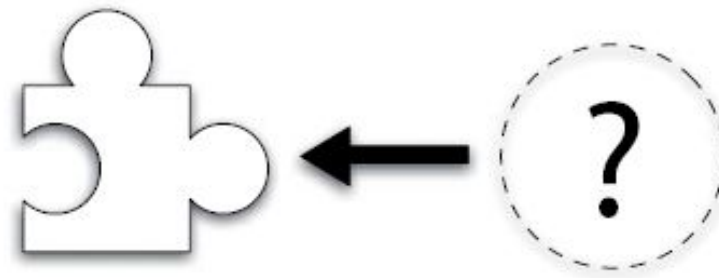
**Figure 1.5**

How do we know what our interface should be like if we don't try to use it? We don't. Writing the test before the code makes us think about our design from the code user's (the developer's) perspective, leading to a usable API.

It's not easy to create simple-to-use APIs. That's why we need all the help we can get. As it turns out, driving our design with tests is extremely effective and produces modular, testable code. Because we're writing the test first, we have no choice but to make the code testable. By definition, the code we write *is* testable— otherwise it wouldn't exist!

LOYOLA UNIVERSITY CHICAGO

# Tool Support for TDD in Java – JUnit

- **JUnit is a <u>unit testing framework</u> for Java**

## Unit-testing with xUnit

A number of years ago, Kent Beck created a unit-testing framework for SmallTalk called SUnit (http://sunit.sf.net). That framework has since given birth to a real movement within the software development community with practically every programming language in active use having gotten a port of its own.[12] For a Java developer, the de facto standard tool—also based on the design of SUnit—is called JUnit, available at http://www.junit.org/. The family of unit-testing frameworks that are based on the same patterns found in SUnit and JUnit is often referred to as *xUnit* because of their similarities: If you know one, it's not much of a leap to start using another (provided that you're familiar with the programming language itself).

What exactly does a *unit-testing framework* mean in the context of xUnit? It means that the library provides supporting code for writing unit tests, running them, and reporting the test results. In the case of JUnit, this support is realized by providing a set of base classes for the developer to extend, a set of classes and interfaces to perform common tasks such as making assertions, and so forth. For running the unit tests written with JUnit, the project provides different *test runners*—classes that know how to collect a set of JUnit tests, run them, collect the test results, and present them to the developer either graphically or as a plain-text summary.

LOYOLA UNIVERSITY CHICAGO

# 15-Minute Break

# Week 1 Topic: Course Software

- **Required Software to download and install**
  - ◦ **Java development kit (JDK) 8 update 102 (current default)**
  - ◦ **Android Studio Integrated Development Environment (IDE)**
  - ◦ **Git Version Control System (VCS) if using Android Studio (AS)**
  - ◦ **SourceTree Git client for Windows & Mac (if not using AS)**
  - ◦ **Gradle Build System (included in Android Studio, optional)**
  - ◦ **Secure Shell (SSH – optional)**

- **Also, create a Bitbucket userid using your Loyola email address (Bitbucket provides cloud-based hosting for VCS repositories – if you use your Loyola email address you can have unlimited shares)**

LOYOLA
UNIVERSITY
CHICAGO

# Tools; Android Studio Demo

## Tools downloads and other related sites:

- http://www.oracle.com/technetwork/java/javase/downloads/
- http://git-scm.com/downloads (optional if using SourceTree)
- http://www.sourcetreeapp.com/ (optional if using Git & Android Studio)
- http://www.gradle.org/ (optional – included with Android Studio)
- http://developer.android.com/sdk/index.html (Android Studio download)
- http://bitbucket.org (sign up for Bitbucket account with <u>Loyola email</u>)
- https://github.com/LoyolaChicagoCode (source for class projects)
- http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html (PuTTY/SSH on Windows – optional, saves using Bitbucket password)
- https://confluence.atlassian.com/display/BITBUCKET/Use+the+SSH+protocol+with+Bitbucket

**LOYOLA UNIVERSITY CHICAGO**

# Week 1 Topic: Projects 1 & 2

- **Project 1 – install software and connect Android Studio to your new Bitbucket repository – due 9/<u>13</u>**
  - Install Java SDK through Android Studio as above from http://laufer.cs.luc.edu/teaching/313/software
  - Create a Bitbucket account <u>using your Loyola email</u> (this guarantees that you can have <u>unlimited</u> shares of your repos)
  - Import the <u>hello-java</u> project as <u>cs413f16p1</u> from https://github.com/LoyolaChicagoCode/hello-java and share that repository with me and with our TA
    - Use ryacobellis313@msn.com as my sharing email address
  - Fix the project so the tests pass, add a screenshot in a new *doc* folder, commit and push, and tell us to grade it

# Week 1 Topic: Projects 1 & 2

- **Project 2 – import and fix a second Android project**
  - ◦ Become more familiar with test-driven development; review linear data structures and their Java implementations
  - ◦ Import the <u>cs2-lab5-listperformance-java</u> project from https://github.com/LoyolaChicagoCode/cs2-lab5-listperformance-java as a new repository <u>cs413f16p2</u> and share it with me and our TA
    - Use ryacobellis313@msn.com as my sharing email address
  - ◦ Fix the project so the tests pass by completing the parts marked <u>TODO</u>, add a README.txt file at the top level of the project answering questions embedded in the code, commit your changes and push them to your Bitbucket repository, and tell us to grade it on Piazza or via a Sakai submission

LOYOLA UNIVERSITY CHICAGO

# Week 1 Prerequisite Assessment

- **Note: this assessment does not count as part of your grade – it's to help me evaluate how well-prepared you are for the course and to recommend help or additional preparation if needed**

# Week 1 Topics

- **Course overview**
- **Motivation**
- **System and Software Requirements**
  - **Functional vs. nonfunctional**
  - **Testing, including test-driven development (TDD)**
- **Course Software**
- **Projects 1 & 2**
- **Prerequisite Assessment**