Objectives

- Testing
- Unit testing
- JUnit Framework
  - In Eclipse

Review

- Start Eclipse for later exercise
- Describe and compare the two software development models we discussed
- How can we categorize prototypes?
  - What are their characteristics?
- Describe the general testing process
- What is a set of test cases called?

Review: Software Testing Process

- Test Suite: set of test cases
- Input
  - Test Case
- Program
  - Program Under Test
- Actual Output
  - Expected Output
  - pass or fail

Types of Testing

- Black-box testing
  - Test functionality (e.g., the calculator)
  - No knowledge of the code
  - Examples of testing: boundary values
- White-box testing
  - Have access to code
  - Goal: execute all code
- Non-functional testing
  - Performance testing
  - Usability testing (HCI)
  - Security testing
  - Internationalization, localization
- Acceptance testing
  - If customer accepts the product

Levels of Testing

- Unit
  - Tests minimal software component, in isolation
  - For us, Class-level testing
  - Web: Web pages (Http Request)
- Integration
  - Tests interfaces & interaction of classes
- System
  - Tests that completely integrated system meets requirements
- System Integration
  - Test system works with other systems, e.g., third-party systems
Why Unit Test?

• Verify code works as intended in isolation
• Find defects early in development
  ➢ Easier to test small pieces
  ➢ Less cost than at later stages

Levels of Testing: Costs

• Unit
  ➢ Tests minimal software component, in isolation
  ➢ For us, class-level testing
  ➢ Web: Web pages (HTTP Request)
• Integration
  ➢ Tests interfaces & interaction of classes
• System
  ➢ Tests that completely integrated system meets requirements
• System Integration
  ➢ Test system works with other systems, e.g., third-party systems

Why Unit Test?

• Verify code works as intended in isolation
• Find defects early in development
  ➢ Easier to test small pieces
  ➢ Less cost than at later stages
• As application evolves, new code is more likely to break existing code
  ➢ Suite of (small) test cases to run after code changes
  ➢ Also called regression testing

Some Approaches to Testing Methods

• Typical case
  ➢ Test typical values of input/parameters
• Boundary conditions
  ➢ Test at boundaries of input/parameters
  ➢ Many bugs live “in corners”
• Parameter validation
  ➢ Verify that parameter and object bounds are documented and checked
  ➢ Example: pre-condition that parameter isn’t null

All black-box testing approaches

Another Use of Unit Testing:
Test-Driven Development

• A development style, evolved from Extreme Programming
• Idea: write tests first, without code bias
• How it works:
  ➢ Write the tests that the code/new functionality should pass
    • Like a specification for the code (pre/post conditions)
    • All tests will initially fail
  ➢ Write the new code and make sure that it passes all test cases

Software Testing Issues

• How should you test? How often?
  ➢ Code may change frequently
  ➢ Code may depend on others’ code
  ➢ A lot of code to validate
• How do you know that an output is correct?
  ➢ Complex output
  ➢ Human judgment?
• What caused a code failure?

Need a systematic, automated, repeatable approach
Characteristics of Good Unit Testing

Why would these be characteristics of good (unit) testing?

- Automatic
- Thorough
- Repeatable
- Independent

- Automatic
  - Since unit testing is done frequently, don’t want humans slowing the process down
  - Running test cases
  - Evaluating results
  - Input: in test itself or from a file

- Thorough
  - Covers all code/functionality/cases

- Repeatable
  - Reproduce results (correct, failures)

- Independent
  - Test cases are independent from each other
  - Easier to trace fault to code

JUnit Framework

- A framework for unit testing Java programs
  - Supported by Eclipse and other IDEs
  - Developed by Erich Gamma and Kent Beck

- Functionality
  - Write tests
    - Validate output, automatically
  - Automate execution of test suites
  - Display pass/fail results of test execution
    - Stack trace where fails
  - Organize tests, separate from code

- But, you still need to come up with the tests!

JUnit

Testing with JUnit

- Typical organization:
  - Set of testing classes
  - Testing classes packaged together in a tests package
  - Separate package from code testing

- A test class typically
  - Focuses on a specific class
  - Contains methods, each of which represents another test of the class

Structure of a JUnit Test

1. Set up the test case (optional)
   - Example: Creating objects
2. Exercise the code under test
3. Verify the correctness of the results
4. Teardown (optional)
   - Example: reclaim created objects
Annotations
- Testing in JUnit 4: uses annotations
- Provide data about a program that is not part of the program itself
- Have no direct effect on operation of the code
- Example uses:
  - @Override: method declaration is intended to override a method declaration in parent class
    - If method does not override parent class method, compiler generates error message
  - Information for the compiler to suppress warnings (@SuppressWarnings)

Tests are Methods
- Mark your testing method with @Test
  - From org.junit.Test
  ```java
  public class CalculatorTest {
      @Test
      public void add() {
          ...}
  ```
- Convention: Method name describes what you’re testing

Assert Methods
- Variety of assert methods available
- If fail, throw an exception
- All static void
- Example:
  ```java
  @Test
  public void add() {
      assertEquals(4, calculator.add(3, 1));
  }
  ```

Set Up/Tear Down
- May want methods to set up objects for every test in the class
  - Called fixtures
  - If have multiple, no guarantees for order executed
  ```java
  @Before
  public void prepareTestData() { ... }
  @Before
  public void setupMocks() { ... }
  @After
  public void cleanupTestData() { ... }
  ```

Set Up/Tear Down For Class
- May want methods to set up objects for set of tests
  - Executed once before any test in class executes
  ```java
  @BeforeClass
  public static void setupDatabaseConnection() { ... }
  @AfterClass
  public static void teardownDatabaseConnection() { ... }
  ```
Eclipse can help make our job easier

- Automatically execute tests (i.e., methods)
- We can focus on coming up with tests

## Using JUnit in Eclipse

- In Eclipse, go to your MediaItems project
- Create a new JUnit Test Case (under Java)
  - Use JUnit 4
    - Add junit to build path
  - Put in package media.tests
  - Name: DVDTest
  - Choose to test DVD class
    - Select setUp and tearDown
    - Select methods to test
- Run the class as a JUnit Test Case

## Unit Testing & JUnit Summary

- Unit Testing: testing smallest component of your code
  - For us: class and its methods
- JUnit provides framework to write test cases and run test cases automatically
  - Easy to run again after code changes
- JUnit Resources available from Course Page’s “Resource” Link, under Java
  - API
  - Tutorials

## Example

- Test method that gets the length of the DVD
  - Revise: Add code to setUp method that creates a DVD

## Notes

- Replaying all the test cases: right click on package
- FastView vs Detached
- Hint: CTL-Spacebar to get auto-complete options

## Project 1: Testing Practice

- Due next Friday
- Given: a Car class that only has enough code to compile
- Your job: Create a good set of test cases that thoroughly/effectively test Car class
  - Find faults in my faulty version of Car class
  - Start: look at code, think about how to test, set up JUnit tests
  - Written analysis of process