SE 350 - Object-Oriented Software Development

Test-Driven Development

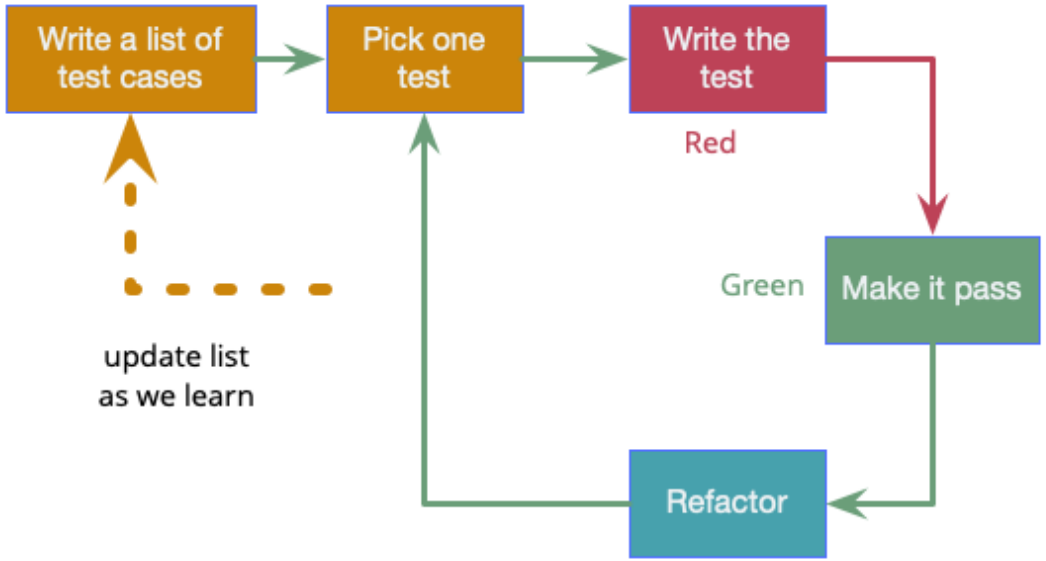
Instructor: Stefan Mitsch

Learning Objectives

- Think in terms of public interfaces
- Understand writing unit tests

Test-Driven Development

- Technique for developing software, guided by tests
- Kent Beck
- Martin Fowler
- Forces us to think about interface first
- Makes refactoring and bug fixes convenient (no hidden regression)



Unit Tests Put Interfaces First

- Unit tests: at the scale of classes (vs. integration tests)
- Promotes thinking about how to use (instead of how to implement)
- Promotes decomposing functionality into reusable pieces
- Helps chosing access modifiers (abstraction and encapsulation)

Test-Driven Development with JUnit

- JUnit is a Java library to automate unit testing
- `@Before`, `@BeforeEach` use to initialize test object before each test
- `@BeforeClass` called when the test class is initialized, use for shared data between test cases
- `@After`, `@AfterEach` clean up resources after each test case
- `@AfterClass` after the entire test class
- `@Test` marks a single test

```
public class ThreeDigitsTests {
   // reference to object under test
  private ThreeDigits td;

  @Before
  public void setUp() { td = new ThreeDigits(); }

  @After
  public void tearDown() { td = null; }

  @Test
  public void testSetDigits() {
    // implement the test
  }

}
```

Junit Expectations

- `assertEquals` checks that a result is equal to an expected result
- `assertArrayEquals` checks content equality of arrays
- `assertThrows` checks that an expected exception is thrown
- `assertTrue`, `assertFalse`, `assertNull`
- `assertThat` checks a predicate

```
public class ThreeDigitsTests {
  private ThreeDigits td;

  // ... test setup and tear-down

  @Test
  public void testSetDigits() {
    td.setDigits(0, 1, 2);
    assertArrayEquals(new int[] {0, 1, 2}, td.getDigits());
  }

  @Test
  public void testSetTooManyDigits() {
    Throwable e = assertThrows(
      IllegalArgumentException.class,
      () -> td.setDigits(0, 1, 2, 3)
    );
    assertEquals("Expected 3 digits, but got 4", e.getMessage());
  }
}
```

Summary

Test-driven development

- promotes specifying interfaces before implementation
- enables refactoring (more about this in a later class)
- helps prevent regression (reproduce bug before fixing it)

Exercises

- Implement test cases for a class that holds 3 digits; functionality: change all digits at once, or one digit at a time, print to string
- What's the bug in the program below? Write a unit test that reproduces the bug and then fix it.

```
public class Fibonacci {
  public static long fib(int n) {
    if (n == 1) return 1;
    else return fib(n-1) + fib(n-2);
  }
}
```