Unit Testing
CMPT 276
Dr. Rob Cameron

Adapted from slides of Dr. Brian Fraser

Topics

1) What are common types of testing?
    a) Testing like a user: through the UI.
    b) Testing like a developer: through the code.
2) What makes a good bug report?
3) How can we write code to test code (via JUnit)?
4) How to do effective unit testing?

Types of Testing

Types of Testing

- Test to find bugs and to show a product works.
- How can we test (types of testing)?
  - .. Acceptance Testing
    - Test overall application's features
    - *"Is the program acceptable to customer?"*
  - .. Unit Testing
    - Test each class in isolation
    - *"Does this **class** do anything wrong?"*
  - .. *Integration Testing*
- Human testing (manual) or by code (automated).

Which type of test uses each?

White vs Black Box

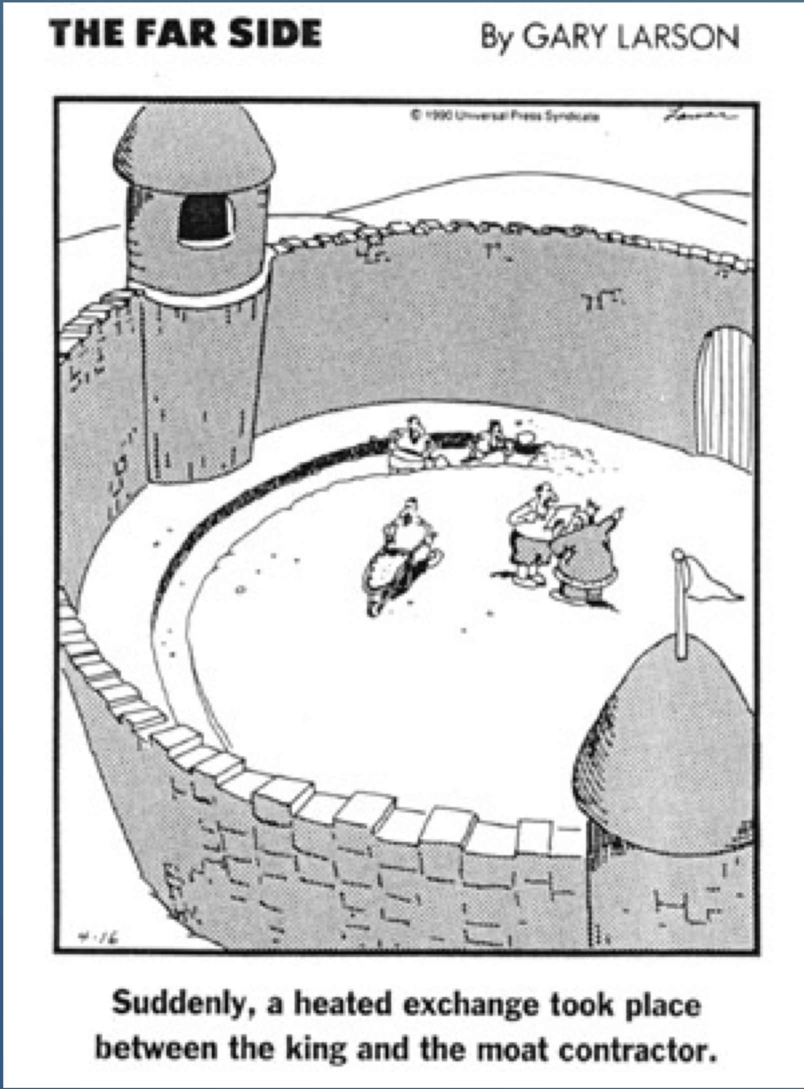
Which is better?

- When creating tests,
  do you have access to the system's code/design?
  - Knowing the code can help you..
    make better, more complete tests.
  - Not knowing the code can help you see the big picture and..find incorrect assumptions in code.
- ..White Box Testing
  - Can see source code when writing tests.
  - Also called clear box or glass box.
- ..Black Box Testing
  - Have no access to system internals.
  - Often for user interface testing.

Acceptance Tests

Acceptance Testing

- Acceptance Testing:.. tests product from customer's perspective
  - Are needed features included?
  - Do the features work as expected?
- Can generate acceptance tests from.. user's requirements.



Suddenly, a heated exchange took place between the king and the moat contractor.

Ex: Requirements to Acceptance Tests

Requirement

- Scroll bar's slider shows the proportion of how much of the content is shown in the window.
- Scroll bar only visible when all content can not be shown in window at once.

Acceptance Tests

- With enough content to need scroll bar, double amount of content and slider should be half as tall.
- With enough content to need scroll bar, double window height and slider height should double.
- ... etc.

..

Acceptance Testing details

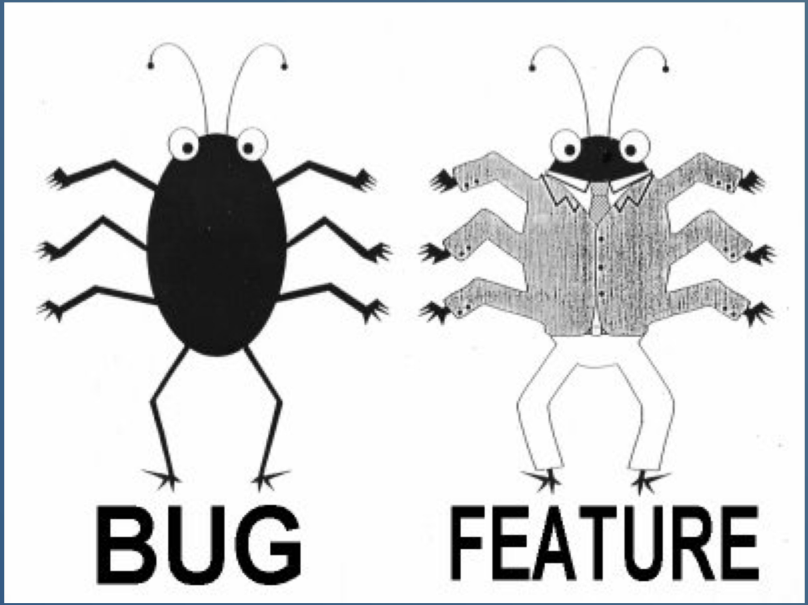
- Acceptance tests often manually done by a tester.

Quality Assurance Tester Job:

- Writing Test Cases and Scripts based on business and functional requirements
- Executing high complexity testing tasks
- Recording and reporting testing task results
- Proactively working with project team members to improve the quality of project deliverables

- Acceptance tests may be part of deploying a product
  - Alpha testing: users try out software at developer's site.
  - Beta testing: software deployed for limited initial testing at customer's site.

Bug reports



Bug Report

- Submit a bug report when a defect is found.

Bug Report Component	**Description**
Summary	Concise, 1 line description of problem.
Component	Which product had error.
Steps to Reproduce	Actions to cause error. Does it always occur, or only occasionally? Create simple example to demonstrate.
Expected vs Actual result	What the steps should do, vs what actually do. Ensure it is actually an error not a feature: "Working as intended"?
Environment	Software version, OS, hardware, drivers, ...

Bug Report Example

Bug Report Component	**Example**
Summary	Upload crashes on MP3 file drag and drop.
Component	File upload window.
Steps to Reproduce	1. Open app to upload window. 2. Select two MP3 files in file explorer. 3. Drag into upload window. 4. Application flashes and crashes. Crash is repeatable.
Expected vs Actual result	Expected “No flashing and no crashing” (files should upload without app crashing)
Environment	ShareFiles 1.2.5, Win10, Dell XYZ, Kaspersky IS 9.

Inspired by an actual bug report submitted by someone I know.

Bug suggestions

- The better the bug report, the more likely the developer is to identify the problem and fix it.
- Example files:
  - For an office application, or a compiler, provide an example file which causes the problem.
- Screenshots:
  - A picture of the problem is great at definitively showing what happened.
  - Developers are often..

    skeptical of problems they cannot reproduce.

Life-cycle of a bug

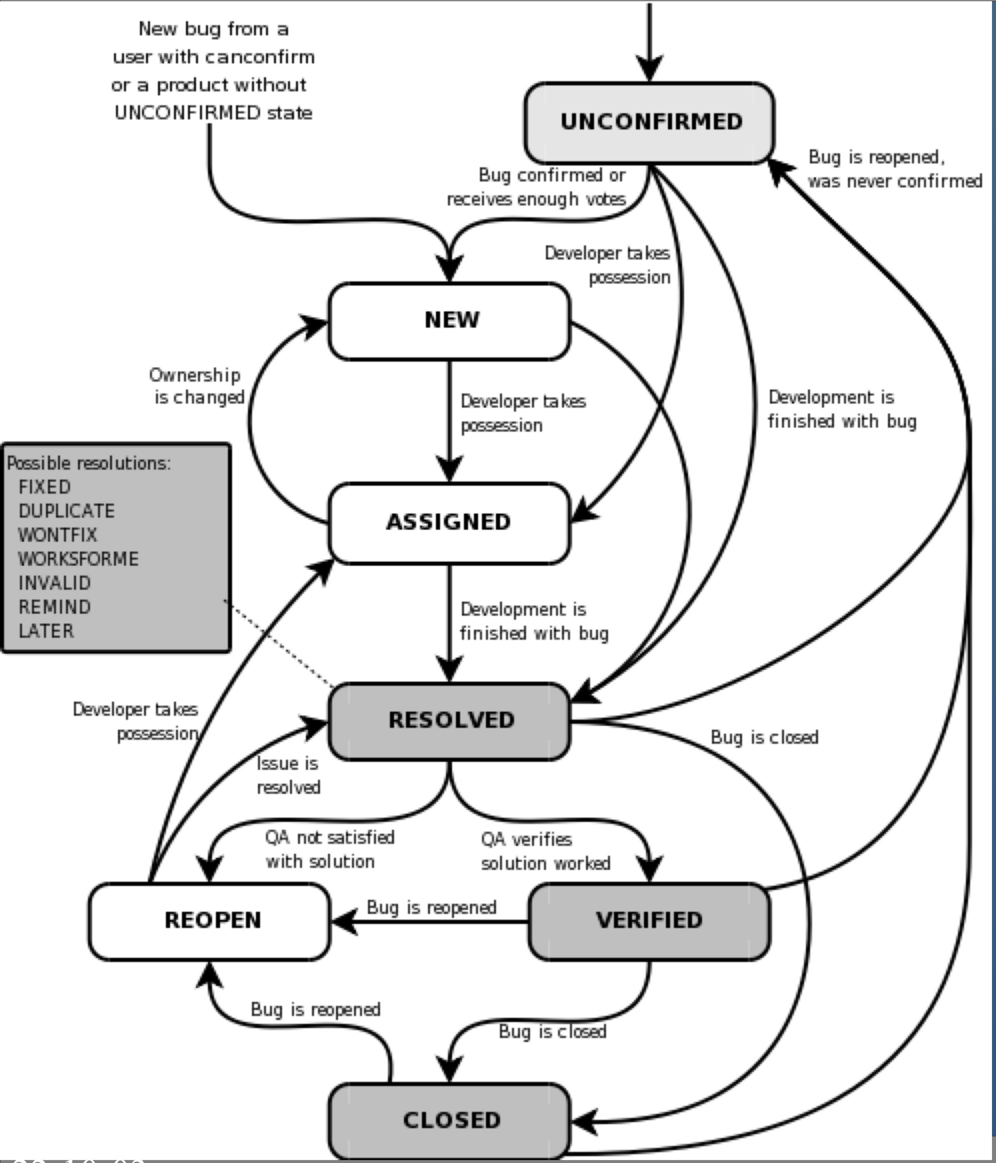

New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
ASSIGNED
Possible resolutions: FIXED DUPLICATE WONTFIX WORKSFORME INVALID REMIND LATER
Development is finished with bug
Developer takes possession
RESOLVED
Issue is resolved
Bug is closed
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

Image Source: Bugzilla – lifecycle.

- Some resolutions:
  - Fixed
  - Duplicate
  - Won't Fix
  - Cannot Reproduce
  - Working as intended
    - "ID-10-T"
    - "PLBKAC"
  - Enhancement / feature request

BUGS HAVE FEELINGS TOO

IF YOU FIND A BUG:
REPORT IT

BUGS DON'T LIKE
TO BE FORGOTTEN

IF YOU FIND A BUG:
GET TO KNOW THEM

BUGS LIKE TO BE
UNDERSTOOD

IF YOU FIND A BUG:
TAKE A PHOTO

BUGS LIKE TO KEEP MEMORIES
OF THE OCCASION

IF YOU FIND A BUG:
GET TO KNOW THEIR MATES

BUGS ARE SOCIALITES

IF YOU FIND A BUG:
REPORT IT QUICK

OTHERWISE BUGS SETTLE IN AND
MAKE A HOME FOR THEM SELVES

IF YOU FIND A BUG:
BE HONEST

BUGS DON'T LIKE
GOSSIPS

IF YOU FIND A BUG:
NOTE HOW YOU
MEET THEM

BUGS ARE ROMANTICS

IF YOU FIND A BUG:
DON'T IGNORE IT

BUGS CAN BITE IF
NOT APPRECIATED

AG

Unit testing
with JUnit

JUnit Unit Testing

- Unit Tests..

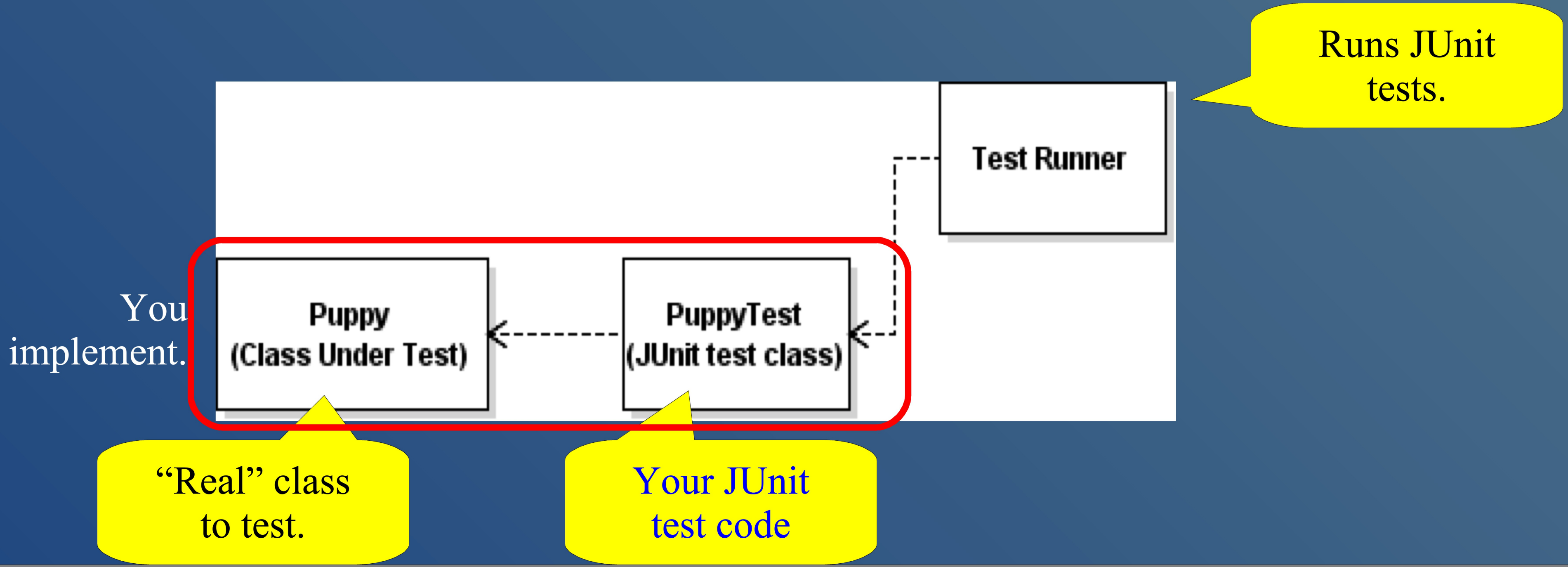
  Test a class in isolation.
- Purpose:

  For *you* to “know” *your* code works.
  - Should test ~100% of a class.
  - Helps improve quality of code.
  - Supports aggressive refactoring because you can..

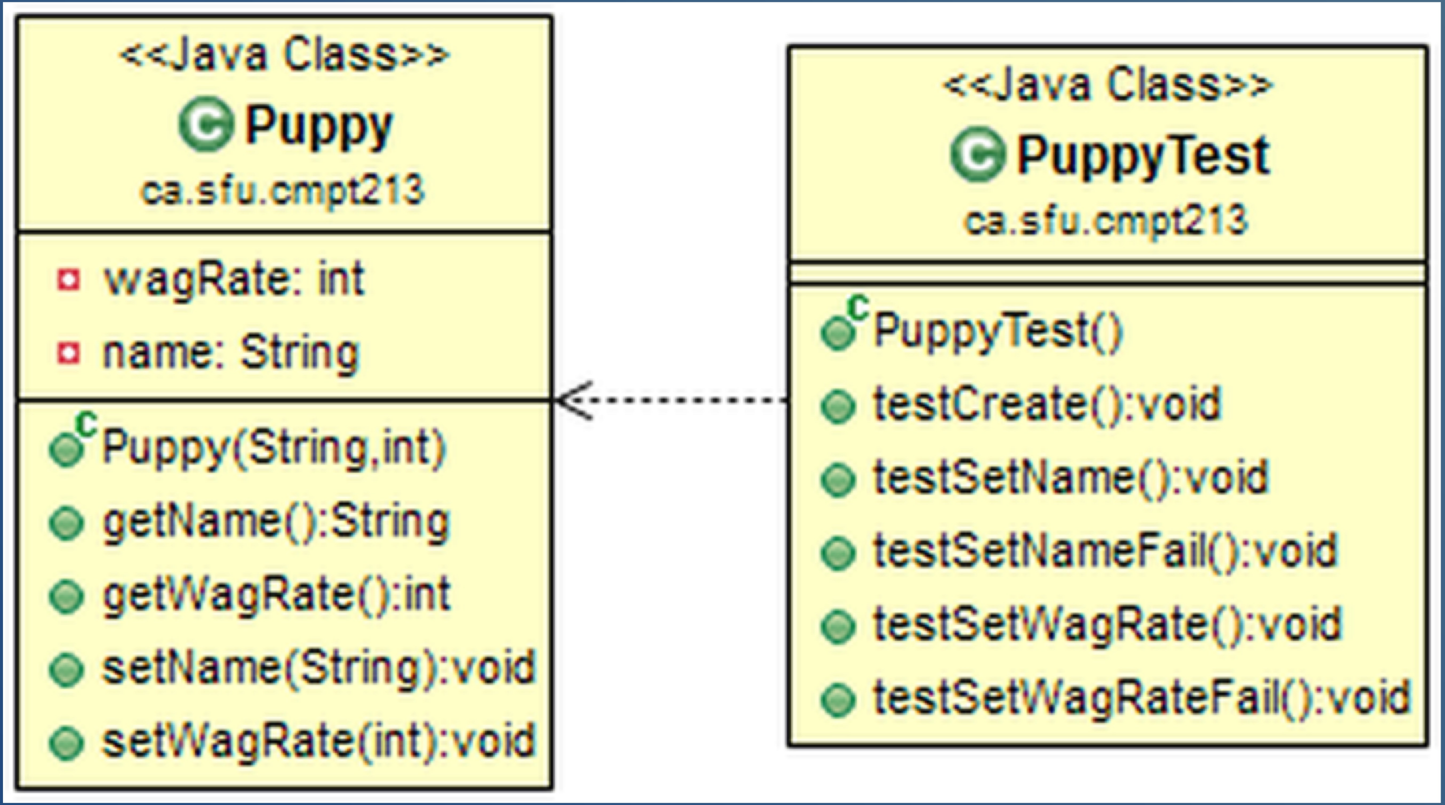
    quickly check your code is correct.

JUnit Context

- You create a test class which is.. paired with the class you want to test.
- JUnit test runner executes your test class.



Basic JUnit Architecture



JUnit 4:
"Test Runner"
executes
methods with..

@Test
annotation

JUnit 4 Example

```
package ca.sfu.cmpt276;
import org.junit.Test;
import static org.junit.Assert.*;
public class PuppyTest {
	@Test
	public void testCreate() {
		Puppy rover = new Puppy("Rover", 100);
		assertEquals("Rover", rover.getName());
		assertEquals(100, rover.getWagRate());
	}

	@Test
	public void testSetName() {
		Puppy rover = new Puppy("Rover", 100);
		rover.setName("Fluffy");
		assertEquals("Fluffy", rover.getName());
	}

	//... more tests omitted.
}
```

Test runner executes all methods with @Test annotaiton

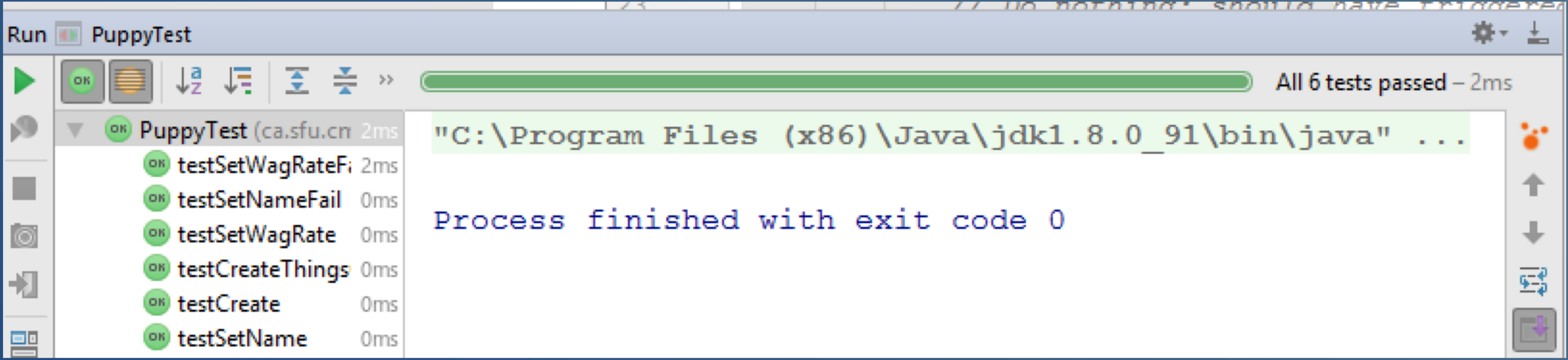
Tests are done using.. JUnit's asserts.

New instance of PuppyTest created for each JUnit test method:

Behaviour of one.. does not affect the others.

Test Runner

- Test runner executes @Test methods in test class.
- Displays results & coloured bar
  - Green-bar.. All tests successful.



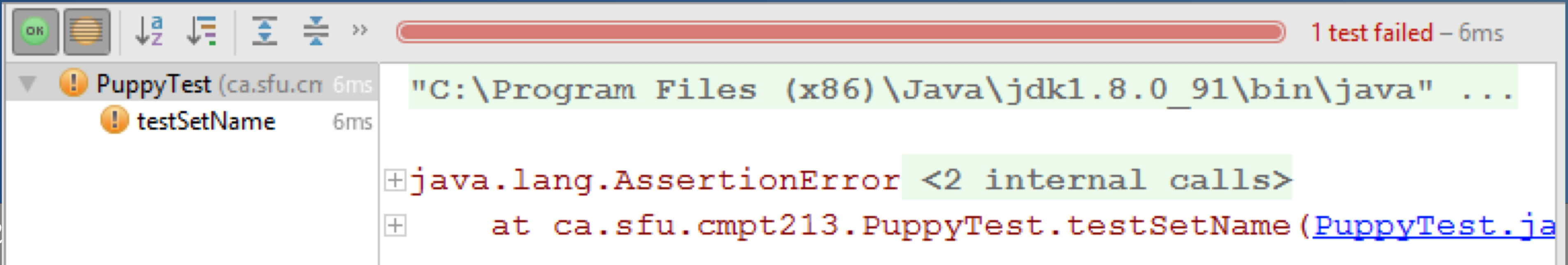
Run PuppyTest

All 6 tests passed – 2ms

```
PuppyTest (ca.sfu.cn  2ms
    testSetWagRateFi  2ms
    testSetNameFail   0ms
    testSetWagRate    0ms
    testCreateThings  0ms
    testCreate        0ms
    testSetName       0ms
```

```
"C:\Program Files (x86)\Java\jdk1.8.0_91\bin\java" ...

Process finished with exit code 0
```

  - Red-bar.. Test(s) failed.



1 test failed – 6ms

```
PuppyTest (ca.sfu.cn  6ms
    testSetName       6ms
```

```
"C:\Program Files (x86)\Java\jdk1.8.0_91\bin\java" ...

java.lang.AssertionError <2 internal calls>
    at ca.sfu.cmpt213.PuppyTest.testSetName(PuppyTest.ja
```

JUnit 4 Asserts: Basics

```
public class JUnitAssertTest {
	@Test
	public void demoAssertEquals() {
		String name = "Dr. Evil";
		assertEquals("Dr. Evil", name);
	}
	@Test
	public void demoOtherAsserts() {
		int i = 10;
		assertEquals(10, i);
		assertTrue(i == 10);
		assertFalse(i == -5);
	}
	@Test
	public void demoAssertEqualsOnDouble() {
		double weight = (1 / 10.0);
		assertEquals(0.1, weight, 0.000001);
	}
	// Array support: assertArrayEquals()
}
```

Doubles have limited precision. 3rd arg is the “delta” to tolerate

JUnit 4 Asserts: Exceptions

```
public class JUnitAssertTest {
    public void throwOnNegative(int i) {
        if (i < 0) {
            throw new IllegalArgumentException();
        }
    }

    @Test (expected = IllegalArgumentException.class)
    public void testThrows() {
        throwOnNegative(-1);
    }

    @Test
    public void testNoThrows() {
        throwOnNegative(1);
    }
}
```

Code likely in class under test
(shown here for simplicity)

Use to test exception throwing..

Test fails unless it throws
IllegalArgumentExecption

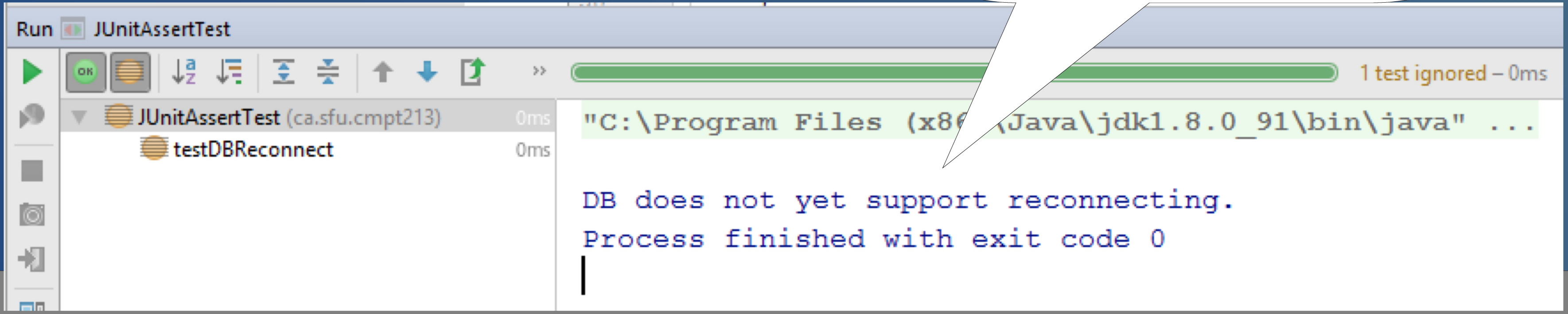
JUnit 4 Asserts: Ignore

```
public class JUnitAssertTest {


    @Ignore("DB does not yet support reconnecting.")
    @Test
    public void testDBReconnect() {
        // ... put your JUnit tests of the not-yet implemented code....
    }

}
```

Ignore the test so "to-be-done" style tests do not break testing.

Why would @Ignore be better than commenting out?

Gives warning message to highlight that some tests not yet enabled.



```
"C:\Program Files (x86)\Java\jdk1.8.0_91\bin\java" ...

DB does not yet support reconnecting.
Process finished with exit code 0
```

JUnit with Android Studio

1) Create JUnit Test Class:
   1) Open class under test,
   2) Click class name, alt-enter --> *Create Test*
   3) Select *JUnit 4*, click *OK*
   4) Select *...\app\**src**\test\java\\.....* folder
2) Execute Tests:
   1) *Run --> Run... (alt-shift-F10)*
   2) Select your JUnit test class.
3) Run app: *Run --> Run...*; select "*app*"

IntelliJ JUnit Video Tutorials:
Basics: https://www.youtube.com/watch?v=Bld3644bIAo&t
More: https://www.youtube.com/watch?v=xHk9yGZ1z3k&t

Unit Testing Discussion

Effective unit tests

- Unit testing should be.. automated!

How?

- Test 'class under test' for:
  - Works for expected normal inputs.
  - Works for extreme or invalid inputs.
- Testing strategies
  - Partition testing:
    - group input values which are "similar"
    - test based on these groupings.
  - Guideline-based testing:
    - use guidelines to choose test cases.
    - guidelines cover common programming errors.

Partition testing

- Identify groups, or regions of values in the input data and output results which... should behave similarly
- Ex: Multiplying two integers.
  - Input: Positive vs negative input values
  - Output: Positive vs negative result.
- Each of these groups is an... equivalence class:
  - Program behaves in an equivalent way for each group member.
- Test cases should be chosen from each partition.
  - test the extremes of the partitions.. (min/max)
  - test a middle value of the partition

Equivalence Classes

- Identify the equivalence classes (partitions):

```
/** Return a grade based on the percent:
 *   50 to 100 = 'P'
 *   0 to <50 = 'F'
 *   otherwise throw an exception.
 */
char assignGrade(int percent);
```

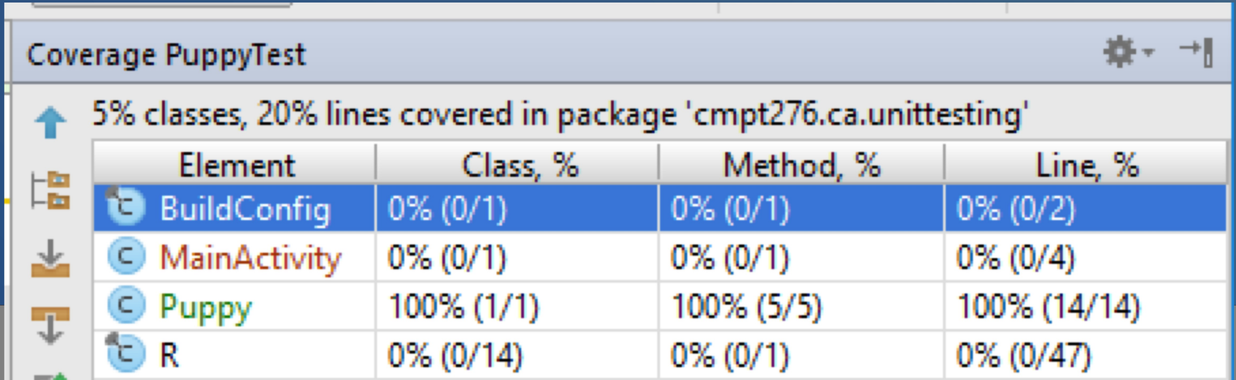
General testing guidelines

Choose test inputs to:

- ..Generate all error messages;
- Cause buffers to overflow;
- Force calculation result to be too large (or small): (overflow & underflow).
- Testing With Arrays:
  - Different # elements. Ex.. 0, 1, 2, 100, 32000...
  - Put desired element.. first, last, middle.

Code Coverage

- Code Coverage:..
  % of class under test's lines executed by tests
- Want ~100% Code Coverage
  - All lines of code executed at least once.
  - Quite hard to achieve (complex error cases, asserts, ..)
  - This should almost be the *bare minimum*:
    Tests run.. each line perhaps only once!
- Demo (Android Studio or IntelliJ)
  Run --> Run PuppyTest with Coverage



Coverage PuppyTest

5% classes, 20% lines covered in package 'cmpt276.ca.unittesting'

Element	Class, %	Method, %	Line, %
BuildConfig	0% (0/1)	0% (0/1)	0% (0/2)
MainActivity	0% (0/1)	0% (0/1)	0% (0/4)
Puppy	100% (1/1)	100% (5/5)	100% (14/14)
R	0% (0/14)	0% (0/1)	0% (0/47)

Test Code Quality

- Unit tests are integral part software development:
  ..Write tests to same code quality standards
  as the rest of the project.
  - Only possible if you don’t think of tests as throw-away or beneath your coding skill.

- Good code quality makes maintenance easier
  - Keeps tests current and relevant
  - Poor code makes tests obsolete fast (and useless)!
  - Unreliable tests cause developers to lose trust.

Finding Many Bugs

- If you find a function which is quite buggy, don’t debug it:
  .. rewrite the function!
  - Good unit testing only finds.. ~30% of defects.
  - A hacked together routine indicates poor understanding of its requirements:
    - If many bugs are discovered now,
      then many bugs will be encountered later!
- More tests cannot solve this problem:

  Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often.

McConnel, 2004

Summary

- White-box knowledge of internals;
  Black-box uses external interface only.
- Test Types
  - Acceptance for checking features in product.
  - JUnit for detailed unit testing (white-box):
    asserts, @Test, ignore, exceptions.
- Bug reports include
  - Description, steps to reproduce, environment info.
- Good JUnit tests
  - Partition testing using equivalence classes.
  - High-quality test code: maintain it!