Unit Testing and Code Confidence

July 5, 2017

Midterm Prep

Question(s) 1

- What is an abstract class? How does it differ from a typical class?
- What is the purpose of an abstract class, in contrast to a typical class?
- What is the relationship between an abstract class and an abstract method?

Question(s) the second

```
class Parent {
    protected String egg = "welp, its a start";
    public String improveString() {
        this.egg = this.egg.replace("start", "bart");
    }

    public String getEgg() {
        return this.egg;
    }
}
class Child extends Parent {
    public String improveString() {
        this.egg = this.egg.replace("welp", "help!");
        super.improveString();
        this.egg = this.egg.replace("help!", "😍");
        super.improveString();
    }
}

Child mySweet = new Child();
System.out.println(mySweet.getEgg());
mySweetChild.improveString();
System.out.println(mySweet.getEgg());
```

Question(s) #three

- What is a "generic type" and when would you want to use it?
- Write a class called "Mapper", using generic types, that has one method. The method should take two ArrayLists of generic type, and return a hash map that maps the contents first array to the second.
  (you can assume the ArrayLists are of equal length).
- java.util.HashMap#put(K key, V value)
  Associates the specified value with the specified key in this map.

(Unit) Testing

Problem

- Projects grow bigger than your brain
- If I fix this bug / change this behavior:
    - Am I assuming certain kinds of input won't be provided?
    - Will things regress?
    - Will it break code that relied on the previous behavior?
    - Can I make sure my code is only getting better?
- "Code wackamole"

sentence-parser/* –>

Problem

- Fixing a bug in one place can cause bugs in other places
- We'd like a clear measure for whether a code base is getting better
- Otherwise we're just punching into darkness

Possible Approaches

- **User Tests**
  Pay lots of people to tell us to use our software, tell us if more or less things are breaking

- **Formal Modeling**
  Build tools / proofs in the language that make errors in possible (garbage collection, the type system, etc are minor efforts at this)

Another Approach…

- Write code to make sure our code is coded right
- Tiny tests to check that one small part of our project is working right
    - If every small part of the project is working, then "hopefully" it'll add up to the whole project working correctly.

Unit Testing (in a Nutshell)

1. Break down our large project into the smallest possible parts (units)

2. Provide the method with input we know the correct output to

3. Check that the method returns the right output

4. Do this a LOT of times

5. Automate the whole process

Unit Testing Example

-
  ```
  public static Integer adder(Integer a, Integer b) {
      return a + b;
  }
  ```

- test 1: adder(1, 2) == 3

- test 2: adder(4, 5) == 9

line-count/* –>

Writing Good Unit Tests

- Should be as focused as possible
    - Test only one aspect of a function at a time
    - Write multiple tests for multiple cases
- Should be fast
    - Make sure they actually get run…

Java and Unit Testing

- Minimize state where possible
- Use dependency injection to focus tests
- Push functionality into the classes, state into the receiver

GoodJavaUnitTests.java –>

jUnit

- Extremely popular unit testing library for java
- Libraries to make tests easier to write
- Tool to make it easy to run lots of tests
- http://junit.org/junit4/

command-line-calc/* –>

Limitations of Unit Testing

- Interrelationship between methods can be difficult to test
- Tests that relate to external data (databases, networks, dates, etc.)
- Not always easy to break down larger applications into smaller parts

Continuous Integration

- Build tests into version control
- If tests don't pass, code is rejected
- Ex: https://jenkins.io/index.html
- Ex: https://travis-ci.org/

Test Driven Development

- Decide on the "success conditions"
- Write tests that enforce the success conditions
- Write code to the tests

sentence-parser/* –>

Housekeeping

A. Work on HW4 example together

B. Unit Test / TDD code

C. End early and self-study for Midterm

Rest of Class

- Midterm Questions
  - Group coding?
- sentence-parser and TDD