CS 61B
Spring 2019

Inheritance

Discussion 4: February 11, 2019

JUnit Tests

1.1 Think about the lab you did last week where we did JUnit testing. The following code is a few of these JUnit tests from the lab.

```java
public class IntListTest {

    @Test
    public void testList() {
        IntList one = new IntList(1, null);
        IntList twoOne = new IntList(2, one);
        IntList threeTwoOne = new IntList(3, twoOne);

        IntList x = IntList.list(3, 2, 1);
        assertEquals(threeTwoOne, x);
    }

    @Test
    public void testdSquareList() {
        IntList L = IntList.list(1, 2, 3);
        IntList.dSquareList(L);
        assertEquals(IntList.list(1, 4, 9), L);
    }
}
```

What are the advantages and disadvantages of writing JUnit tests?

- Advantages:
  1. Keeps your code organized - each test corresponds to different building blocks of your program
  2. You can debug your code locally and find which part of your program is not working
  3. Provides documentation that your program actually works
  4. Can reduce the amount of test code you need to write because you can reuse code
- Disadvantages:
  1. Hard to use for higher-level testing

Creating Cats

2.1 Given the `Animal` class, fill in the definition of the `Cat` class so that when `greet()` is called, "Cat says: Meow!" is printed (instead of "Animal says: Huh?"). Cats less than the ages of 5 can say "MEOW!" instead of "Meow!"

```
public class Animal {
    protected String name, noise;
    protected int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
        this.noise = "Huh?";
    }

    public String makeNoise() {
        if (age < 5) {
            return noise.toUpperCase();
        } else {
            return noise;
        }
    }

    public void greet() {
        System.out.println("Animal " + name + " says: " + makeNoise());
    }
}
```

```
public class Cat extends Animal {
```

```
    public Cat(String name, int age) {
        super(name, age);       // Call superclass' constructor.
        this.noise = "Meow!";   // Change the value of the field.
    }

    @Override
    public void greet() {
        System.out.println("Cat " + name + " says: " + makeNoise());
    }
}
```

Raining Cats and Dogs

3.1 Assume that `Animal` and `Cat` are defined as above. What would Java print on each of the indicated lines?

```
1  public class TestAnimals {
2      public static void main(String[] args) {
3          Animal a = new Animal("Pluto", 10);
4          Cat c = new Cat("Garfield", 6);
5          Dog d = new Dog("Fido", 4);
6
7          a.greet();          // (A) Animal Pluto says: Huh?
8          c.greet();          // (B) Cat Garfield says: Meow!
9          d.greet();          // (C) Dog Fido says: WOOF!
10
11         a = c;
12         ((Cat) a).greet();  // (D) Cat Garfield says: Meow!
13         a.greet();          // (E) Cat Garfield says: Meow!
14     }
15 }
16
17 public class Dog extends Animal {
18     public Dog(String name, int age) {
19         super(name, age);
20         noise = "Woof!";
21     }
22
23     @Override
24     public void greet() {
25         System.out.println("Dog " + name + " says: " + makeNoise());
26     }
27
28     public void playFetch() {
29         System.out.println("Fetch, " + name + "!");
30     }
31 }
```

Consider what would happen if we added the following to the bottom of `main` under line 13:

```
a = new Dog("Spot", 10);
d = a;
```

Why would this code produce a compiler error? How could we fix this error?

This code produces a compiler error in the second line. The static type of `d` is `Dog` while the static type of `a` is `Animal`. `Dog` is a subclass of `Animal`, so this assignment will fail at compile time because not all `Animals` are `Dogs`. Use casting to address the problem.

```
d = (Dog) a;
```

This represents a promise to the compiler that at runtime, a will be bound to an object that is compatible with the Dog type.

Note: The @Override tag specifies that the function overrides a parent class's function. Note 2: You can only call *one* other constructor from a constructor, and the call *has* to be on the first line. This call is "super" which means the superclass' constructor. You can use "this(...)" to call a different constructor defined in the same class.) Note 3: A runtime error would occur if we lie during casting. (That if "a" was not *actually* a Dog object, and instead was a Cat or something else, the code would fail at runtime.)

An Exercise in Inheritance Misery *Extra*

4.1 Cross out any lines that cause compile-time errors or cascading errors (failures that occur because of an error that happened earlier in the program), and put an X through runtime errors (if any). Don't just limit your search to main, there could be errors in classes A,B,C. What does `D.main` output after removing these lines?

```
class A {
    public int x = 5;
    public void m1() {      System.out.println("Am1-> " + x);              }
    public void m2() {      System.out.println("Am2-> " + this.x);         }
    public void update() {  x = 99;                                        }
}
class B extends A {
    public void m2() {      System.out.println("Bm2-> " + x);              }
    public void m2(int y) { System.out.println("Bm2y-> " + y);             }
    public void m3() {      System.out.println("Bm3-> " + "called");       }
}
class C extends B {
    public int y = x + 1;
    public void m2() {      System.out.println("Cm2-> " + super.x);        }
    \\ public void m4() {      System.out.println("Cm4-> " + super.super.x); } can't do super.super
    public void m5() {      System.out.println("Cm5-> " + y);              }
}
class D {
    public static void main (String[] args) {
        \\ B a0 = new A(); Dynamic type must be B or subclass of B
        \\ a0.m1(); cascading: prev line failed, so a0 can't be initialized
        \\ a0.m2(16); cascading: prev line failed, so a0 can't be initialized
        A b0 = new B();
        System.out.println(b0.x); [prints "5"]
        b0.m1(); [prints "Am1-> 5"]
        b0.m2(); [prints "Bm2-> 5"]
        \\ b0.m2(61); m2 (int y) not defined in static type of b0
        B b1 = new B();
        b1.m2(61); [prints "Bm2y-> 61"]
        b1.m3(); [prints "Bm3-> called"]
        A c0 = new C();
        c0.m2(); [prints "cm2-> 5"]
        \\ C c1 = (A) new C(); Can't assign c1 to an A
        A a1 = (A) c0;
        C c2 = (C) a1;
        c2.m3(); [print Bm3-> called]
        \\ c2.m4(); C.m4() is invalid
        c2.m5(); [print Cm5-> 6]
        ((C) c0).m3(); [print Bm3-> called]
        \\ (C) c0.m3(); NOT RUNTIME ERROR This would case the result of what the method returns and
    it returns void therefore compile-time error
        b0.update();
        b0.m1(); [print Am1-> 99]
    }
}
```