CS 61B
Fall 2020

Inheritance

Discussion 4: September 14, 2020

1 JUnit Tests

(a) 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

(b) Think about the lab you did last week where we did JUnit testing. Fill in the following tests so that they test the constructor and `dSquareList` functions of `IntList`.

```
1  public class IntListTest {
2
3      @Test
4      public void testList() {
5          IntList one = new IntList(1, null);
6          IntList twoOne = new IntList(2, one);
7          IntList threeTwoOne = new IntList(3, twoOne);
8
9          IntList x = IntList.list(3, 2, 1);
10         assertEquals(threeTwoOne, x);
11     }
12
13     @Test
14     public void testdSquareList() {
15         IntList L = IntList.list(1, 2, 3);
16         IntList.dSquareList(L);
17         assertEquals(IntList.list(1, 4, 9), L);
18     }
19 }
```

2 Creating Cats

Given the `Animal` class, fill in the definition of the `Cat` class so that when `greet()` is called, "Cat [name] says: Meow!" is printed (instead of "Animal [name] says: Huh?"). Cats less than the ages of 5 should say "MEOW!" instead of "Meow!". Don't forget to use `@Override` if you are writing a function with the same signature as a function in the superclass.

```
1  public class Animal {
2      protected String name, noise;
3      protected int age;
4
5      public Animal(String name, int age) {
6          this.name = name;
7          this.age = age;
8          this.noise = "Huh?";
9      }
10
11     public String makeNoise() {
12         if (age < 5) {
13             return noise.toUpperCase();
14         } else {
15             return noise;
16         }
17     }
18
19     public void greet() {
20         System.out.println("Animal " + name + " says: " + makeNoise());
21     }
22 }
```

```
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());
    }
}
```

3 Raining Cats and Dogs

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

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

(b) Consider what would happen if we added the following to the bottom of `main` under line 27:

```
a = new Animal("Fluffy", 2);
c = a;
```

Would this code produce a compiler error? What if we set the second line to be `c = (Cat) a` instead?

This code would produce a compiler error. This is because we are trying to assign a variable of static type `Animal` to a variable of static type `Cat`. Since `Animal` is a more generic type than `Cat`, not all `Animals` are `Cats`. This specifically fails at compile time because Java runs a check before executing code to prevent you form assigning things of the wrong type.

If we were set the second line to be `c = (Cat) a` instead, this code will fail at runtime instead. This is because by casting `a` to `Cat`, we are telling the Java compiler that we know that `a` actually contains a `Cat`, even though it is statically typed to be an `Animal`. When Java goes to execute the code, it tries to assign the `Animal` to `c`, which errors.

(c) Consider what would happen if we instead added the following to the bottom of `main` under line 27:

```
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.)

4 An Exercise in Inheritance Misery *Extra*

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?

```
1  class A {
2      public int x = 5;
3      public void m1() {      System.out.println("Am1-> " + x);             }
4      public void m2() {      System.out.println("Am2-> " + this.x);        }
5      public void update() {  x = 99;                                        }
6  }
7  class B extends A {
8      public void m2() {      System.out.println("Bm2-> " + x);             }
9      public void m2(int y) { System.out.println("Bm2y-> " + y);            }
10     public void m3() {      System.out.println("Bm3-> " + "called");      }
11 }
12 class C extends B {
13     public int y = x + 1;
14     public void m2() {      System.out.println("Cm2-> " + super.x);       }
15     \\ public void m4() {      System.out.println("Cm4-> " + super.super.x); } can't do super.super
16     public void m5() {      System.out.println("Cm5-> " + y);             }
17 }
18 class D {
19     public static void main (String[] args) {
20         \\ B a0 = new A(); Dynamic type must be B or subclass of B
21         \\ a0.m1(); cascading: prev line failed, so a0 can't be initialized
22         \\ a0.m2(16); cascading: prev line failed, so a0 can't be initialized
23         A b0 = new B();
24         System.out.println(b0.x); [prints "5"]
25         b0.m1(); [prints "Am1-> 5"]
26         b0.m2(); [prints "Bm2-> 5"]
27         \\ b0.m2(61); m2 (int y) not defined in static type of b0
28         B b1 = new B();
29         b1.m2(61); [prints "Bm2y-> 61"]
30         b1.m3(); [prints "Bm3-> called"]
31         A c0 = new C();
32         c0.m2(); [prints "cm2-> 5"]
33         \\ C c1 = (A) new C(); Can't assign c1 to an A
34         A a1 = (A) c0;
35         C c2 = (C) a1;
36         c2.m3(); [print Bm3-> called]
37         \\ c2.m4(); C.m4() is invalid
38         c2.m5(); [print Cm5-> 6]
39         ((C) c0).m3(); [print Bm3-> called]
```

```
40          \\ (C) c0.m2(); NOT RUNTIME ERROR This would cast the result of what the method returns and
        it returns void therefore compile-time error
41          b0.update();
42          b0.m1(); [print Am1-> 99]
43      }
44  }
```