JUnit Tests

(a) What are the advantages and disadvantages of writing JUnit tests?

(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.

```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(_________________);
        assertEquals(_______________, _______________);
    }

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

```java
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 {

}
```
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?

```java
public class Dog extends Animal {
    public Dog(String name, int age) {
        super(name, age);
        noise = "Woof!";
    }

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

    public void playFetch() {
        System.out.println("Fetch, " + name + "!");
    }
}

public class TestAnimals {
    public static void main(String[] args) {
        Animal a = new Animal("Pluto", 10);
        Cat c = new Cat("Garfield", 6);
        Dog d = new Dog("Fido", 4);
        a.greet(); // (A) --------------------------
        c.greet(); // (B) --------------------------
        d.greet(); // (C) --------------------------
        a = c;
        ((Cat) a).greet(); // (D) --------------------------
        a.greet(); // (E) --------------------------
    }
}
```
(b) Consider what would happen if we added the following to the bottom of main under line 27:

```java
    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?

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

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

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

```java
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); }
    public void m5() { System.out.println("Cm5-> " + y); }
}

class D {
    public static void main (String[] args) {
        B a0 = new A();
        a0.m1();
        a0.m2(16);
        A b0 = new B();
        System.out.println(b0.x);
        b0.m1();
        b0.m2();
        b0.m2(61); 
        B b1 = new B();
        b1.m2(61);
        b1.m3();
        A c0 = new C();
        c0.m2();
        C c1 = (A) new C();
        A a1 = (A) c0;
        C c2 = (C) a1;
        c2.m3();
        c2.m4();
        c2.m5();
        ((C) c0).m3();
        (C) c0.m2();
    }
}
```
Inheritance

b0.update();
b0.m1();
}
}