1 Creating Cats

Given the Animal class, fill in the definition of the Cat class so that when greet() is called, the label "Cat" (instead of "Animal") is printed to the screen. Assume that a Cat will make a "Meow!" noise, and that this is all caps for cats younger than 5 years old.

```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 {
    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());
    }
}
```
2 Raining Cats and Dogs

Assume that Animal and Cat are defined as above. What will be printed at each of the indicated lines?

```java
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) ______________________
    }
}
```

```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 + ")
    }
}
```

(A) Animal Pluto says: Huh?
(B) Cat Garfield says: Meow!
(C) Dog Fido says: WOOF!
(D) Cat Garfield says: Meow!
(E) Cat Garfield says: Meow!

Consider what would happen if we added the following to the bottom of main:

```java
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.

We can fix that by using a cast:
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.
Cross out any lines that cause compile-time errors, and put an X through runtime errors (if any).

What does the main program (in class D) output after removing these lines? Note: There are many cases covered here and possibly not enough time to finish in discussion. Remember that solutions will be posted online later this week.

```java
3 An Exercise in Inheritance Misery

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) {
        <C> // B a0 = new A(); a0 must be B or a subclass of B.
        <C> // a0.m1(); a0 is invalid
        <C> // a0.m2(16); a0 is invalid
        A b0 = new B();
        System.out.println(b0.x); [5]
        b0.m1(); [Am1-> 5]
        b0.m2(); [Bm2-> 5]
        <C> // b0.m2(61); m2(int y) not defined in static type B
        B b1 = new B();
        b1.m2(61); [Bm2y-> 61]
        b1.m3(); [Bm3-> called]
        A c0 = new C();
        c0.m2(); [cm2-> 5]
        <C> // C c1 = (A) new C(); Can’t assign c1 to an A.
        A a1 = (A) c0;
        C c2 = (C) a1;
        c2.m3(); [Bm3-> called]
        <C> // c2.m4(); C.m4() is invalid
        c2.m5(); [Cm5-> 6]
        ((C) c0).m3(); [Bm3-> called]
        <C NOT R> //((C) c0.m3()); This would cast the result of what the method returns.
        b0.update();
        b0.m1(); [Am1-> 99]}
    }
```