

Warm-ups solutions

1.
 - A static function is a type of function that you invoke without creating an instance of the class. For example, we have a class named MyClass with a static function `public static void foo()`. Then, we can call it by `MyClass.foo()`.
 - An instance function can only be called by an instance of the class. For example, you have an instance `MyClass c = new MyClass()` and a instance function `public void bar()`. Then, we can call it by `c.bar()`.
2. A static function cannot access/modify instance variables of the class, but it can access/modify static variables of the class.
3. It marks a variable to be unchangeable after being set.
- 4.

```
public class Animal { }  
public class Bird extends Animal { }  
public class Chick extends Bird { }  
public class Penguin extends Bird { }
```

5. The keyword is `super`. Depending on your constructor(s) defined in the parent class, `super` can have different numbers of parameters.
6. *one* and *three*.
7. By using the `@Override` label and having the same function declaration of the parent function.
8. The function call will throw an incompatible type error: because a Pet is not necessarily a Penguin.
9. Upcasting is referring to an object to its supertypes. A few examples:

```
Penguin p = new Penguin();  
Bird b = new Penguin(); // upcasting to Bird  
Animal a = new Penguin(); // upcasting to Animal  
Object o = new Penguin(); // upcasting to Object
```

The functions/variables available for the object to use is determined by the type it is cast to. For example, `a` cannot access functions/variables uniquely defined in the Penguin class because it has type Bird.

10. Yes, because `b` has no access to function `swim()` as it has type Bird.
11. No, because we downcast `b` to be Penguin and so it can access Penguin class function. This is doable for `b` because it is an instance of Penguin at the first place (although we upcast it to Bird type first).
12. If the object comparing to is `null`; if it is an instance of the current object; anything else you think that should be equal.
13. For example:

```
public class Animal {  
    // any variables defined  
  
    public Animal() { }  
  
    public Animal(Animal other) {  
        // copy over the variables "other" has  
    }  
}
```

```
}
```

Combining it all together!

1.

```
public class Course {
    private String name;
    private int attendance;
    private String professor;

    public Course(String setName, int setAttd, String prof) {
        name = setName;
        assert setAttd >= 0 : "attendance needs to be non-negative.";
        attendance = setAttd;
        professor = prof;
    }
}
```

2.

```
public class CS10025 extends Course {
    private double gpa;

    public CS10025(String setName, int setAttd, String prof, double setGPA) {
        super(setName, setAttd, prof);
        gpa = setGPA;
        System.out.println("This is Amazhingu teaching!");
    }
}
```

3.

```
public class CS10025 extends Course {
    //...

    @Override
    public boolean equals(Object other) {
        if (other == null || !(other instanceof CS10025)) {
            return false;
        }
        CS10025 c = (CS10025) other;
        return name.equals(c.name) && professor.equals(c.professor);
    }
}
```

4.

```
public class CS10025 extends Course {
    //...

    @Override
    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        // Notice, even if "other" is of type CS10025,
        // it is still an instance of Course, as CS10025
        // extends Course. So this if statement is sufficient
    }
}
```

```
    if (other instanceof Course) {  
        Course c = (Course) other;  
        return name.equals(c.name);  
    }  
}  
}
```