Study questions set #7

1. Fill out the full access modifier table below. Columns and rows are not in the same order as shown in lecture.

Modifier	Package	World	Subclass	Class
Protected				
Private				
Public				
Package-private				

2. True or false:
   a. Final methods can only be declared in final classes.
   b. Abstract methods can only be declared in abstract classes.
   c. Classes can extend multiple super-classes.
   d. Classes can implement multiple interfaces.

3. Consider the following class:

```
public class Animal {

   double x, y;
   double health, age;

   public Animal(double x, double y, double health) {
         this.x = x; this.y = y; this.health = health;
         this.age = 0;
   }
   public void move(double x, double y) {
         this.x = x;
         this.y = y;
   }
   public void tick() {
         age += 1.0;
         health -= 1.0;
   }

}
```

Implement a `Mouse` class that extends `Animal`. `Mouse` should have the following fields:

- A named constant *`INITIAL_HEALTH`* equal to 10.0
- An instance field `babies` which is an `ArrayList` of `Mouse` objects
- An instance field `cheese` whose data-type is `double`

Mouse should also have two constructors:

- Three parameter constructor: two coordinates and a cheese amount. This constructor should invoke the `super` constructor, using *`INITIAL_HEALTH`*, and initialize the `cheese` field.
- Zero parameter constructor: This constructor should use the `this` keyword to invoke the three parameter constructor, with initial coordinates (0, 0) and a cheese amount of 10.0.

Finally, mouse should have the following instance methods:

- `Scavenge`(): increases cheese by 1.0
- `Eat`(): decreases cheese by 1.0 and increases health by 1.0
- `Tick`(): invokes the overridden method of the super-class, `scavenges`, and `eats`.
- `haveBabies`(`int numBabies`): Constructs the given number of babies, at the same position as this `Mouse`, with this `Mouse's` cheese distributed evenly among them, and adds them to this `Mouse's` babies list.

4. Implement an immutable class `Pair` that implements the `Comparable<Pair>` interface. This class should have two instance fields: `fst` (of type `Integer`) and `snd` (of type `String`). In addition, it should have getter methods for each field and a constructor that initializes each field. It should implement `compareTo`(`Pair other`) by comparing corresponding fields lexographically.
5. Repeat question 3, but replace `Integer` and `String` with two distinct type parameters: `T`, which implements `comparable<T>`, and `U`, which implements `comparable<U>`.