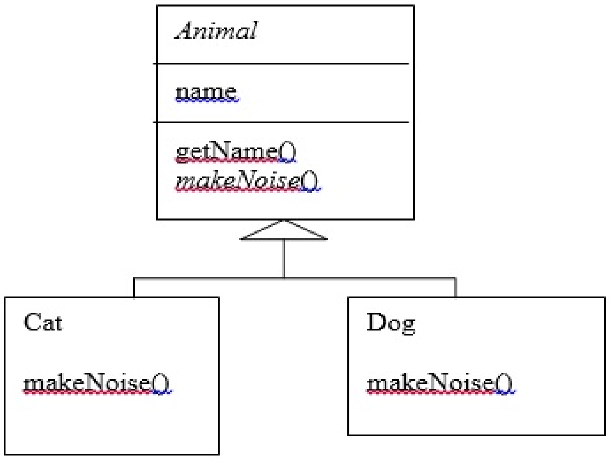
Question 1

Given the following diagram design the Animal, Cat and Dog class:



Question 2

Create a class called `Employee` whose objects are records for an employee. This class will be a derived class of the class `Person` which you will have to copy into a file of your own and compile. An employee record has an employee's name (inherited from the class `Person`), an annual salary represented as a single value of type `double`, a year the employee started work as a single value of type `int` and a national insurance number, which is a value of type `String`.

Your class should have a reasonable number of constructors and accessor methods, as well as an `equals` method. Write another class containing a `main` method to fully test your class definition.

Question 3

A class called `Author` is designed as follows:

- Three `private` member variables: `name` (`String`), `email` (`String`), and `gender` (`char` of either `'m'` or `'f'` - you might also use a `boolean` variable called `isMale` having value of `true` or `false`).
- A constructor to initialize the `name`, `email` and `gender` with the given values. (There is no *default constructor*, as there is no default value for `name`, `email` and `gender`.)
- Public getters/setters: `getName()`, `getEmail()`, `setEmail()`, and `getGender()`. (There are no setters for `name` and `gender`, as these properties are not designed to be changed.)

- A `toString()` method that returns "*`name`* `(`*`gender`*`) at` *`email`*", e.g., "`Tan Ah Teck (m) at ahTeck@somewhere.com`".

Design a `Book` class. Assume that a book is written by one (and exactly one) author. The Book class contains the following members:

- Four `private` member variables: `name` (`String`), `author` (an *instance* of the `Author` class we have just created, assuming that each book has exactly one author), `price` (`double`), and `qty` (`int`).
- The `public` getters and setters: `getName()`, `getAuthor()`, `getPrice()`, `setPrice()`, `getQty()`, `setQty()`.
- A `toString()` that returns "`'book-name' by author-name (gender) at email`". You could reuse the `Author`'s `toString()` method, which returns "`author-name (gender) at email`".

Question 4 - Painting Shapes

In this exercise you will develop a class hierarchy of shapes and write a program that computes the amount of paint needed to paint different objects. The hierarchy will consist of a parent class Shape with three derived classes - Sphere, Rectangle, and Cylinder. For the purposes of this exercise, the only attribute a shape will have is a name and the method of interest will be one that computes the area of the shape (surface area in the case of three-dimensional shapes). Do the following.

1. Write an abstract class Shape with the following properties:
   An instance variable `shapeName` of type String
   An abstract method `area()`
   A `toString` method that returns the name of the shape

2. The file `Sphere.java` contains a class for a sphere which is a descendant of `Shape`. A sphere has a `radius` and its `area` (surface area) is given by the formula 4*PI*radius^2. Define similar classes for a rectangle and a cylinder. Both the `Rectangle` class and the `Cylinder` class are descendants of the `Shape` class. A rectangle is defined by its length and width and its area is length times width. A cylinder is defined by a radius and height and its area (surface area) is PI*radius^2*height. Define the `toString` method in a way similar to that for the `Sphere` class.

3. The file `Paint`.java contains a class for a type of paint (which has a "`coverage`" and a method to compute the amount of paint needed to paint a shape). Correct the return statement in the amount method so the correct amount will be returned. Use the fact that the amount of paint needed is the area of the shape divided by the coverage for the paint. (NOTE: Leave the print statement - it is there for illustration purposes, so you can see the method operating on different types of Shape objects.)

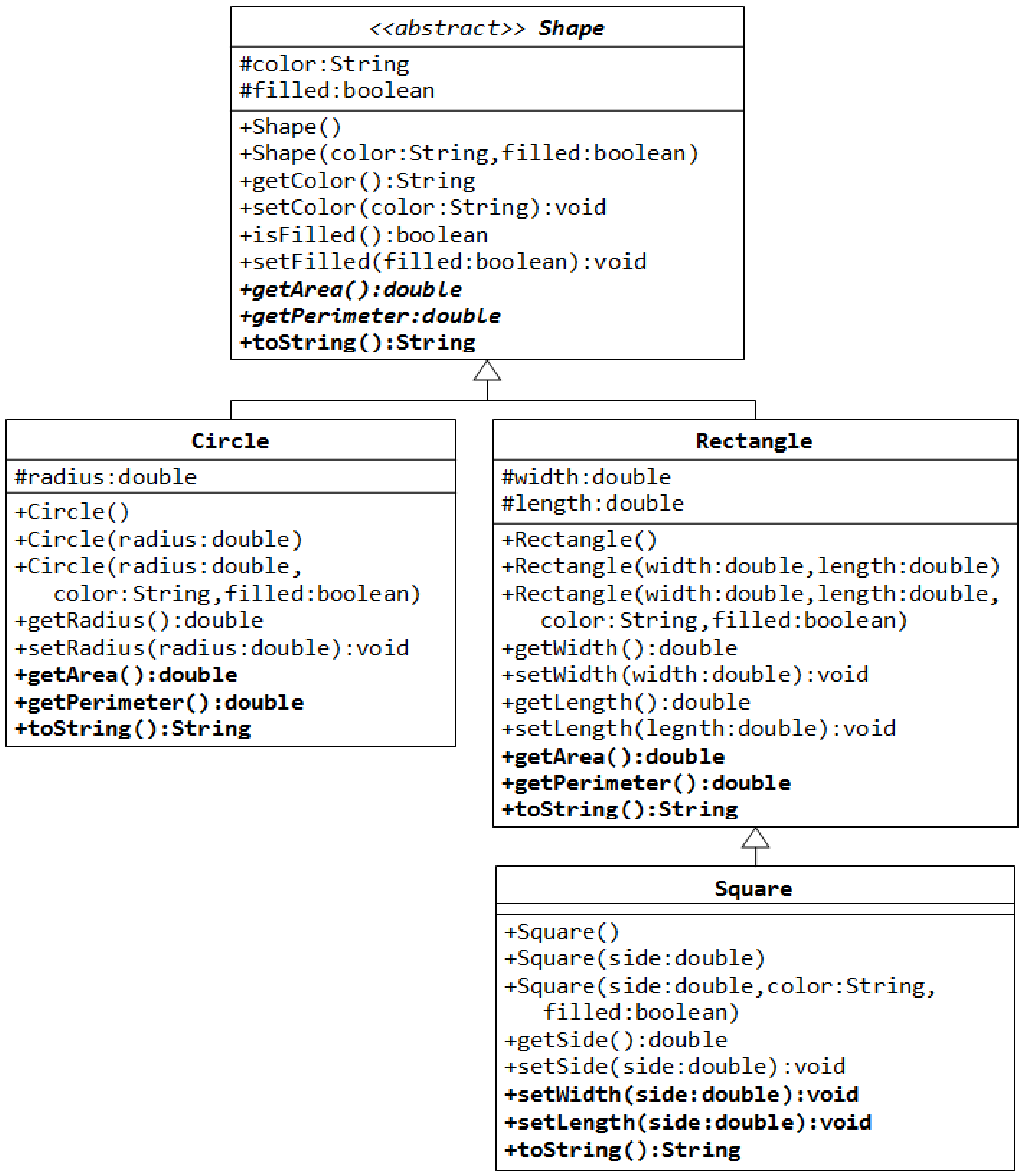
4. The file `PaintThings`.java contains a program that computes the amount of paint needed to paint various shapes. A paint object has been instantiated. Add the following to complete the program:

   Instantiate the three shape objects: deck to be a 20 by 35 foot rectangle, bigBall to be a sphere of radius 15, and tank to be a cylinder of radius 10 and height 30. Make the appropriate method calls to assign the correct values to the three amount variables.

Run the program and test it. You should see polymorphism in action as the amount method computes the amount of paint for various shapes.

Question 5

Write the superclass `Shape` and its subclasses `Circle`, `Rectangle` and `Square`, as shown in the diagram.



In this exercise, `Shape` shall be defined as an `abstract` class, which contains:

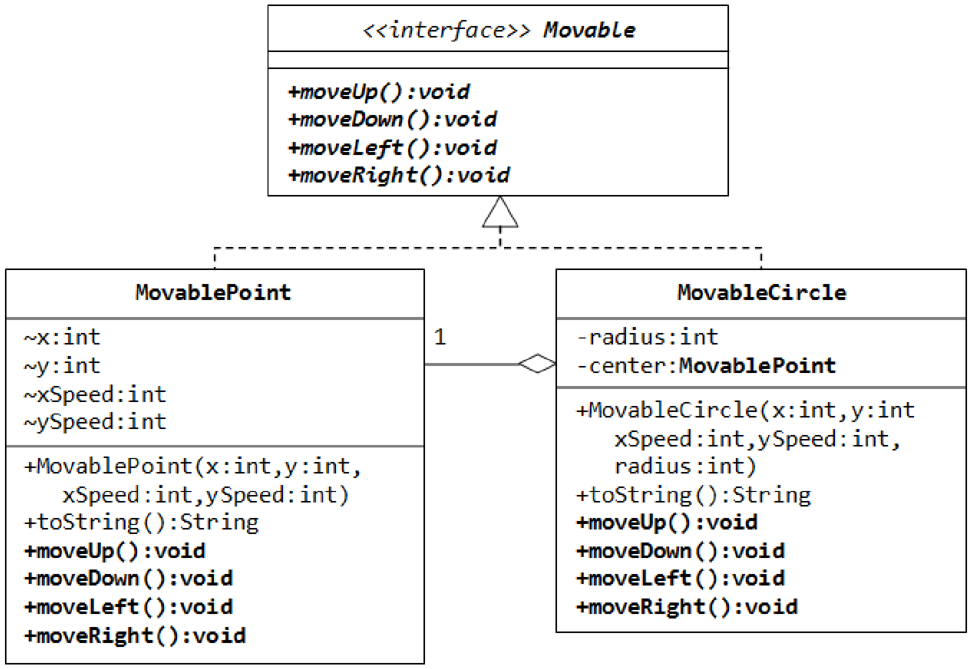
- Two `protected` instance variables `color`(`String`) and `filled`(`boolean`). The `protected` variables can be accessed by its subclasses and classes. They are denoted with a `'#'` sign in the diagram.
- Getter and setter for all the instance variables, and `toString()`.
- Two `abstract` methods `getArea()` and `getPerimeter()` (shown in italics in the diagram).

The subclasses `Circle` and `Rectangle` shall *override* the `abstract` methods `getArea()` and `getPerimeter()` and provide the proper implementation. They also *override* the `toString()`.

Question 6

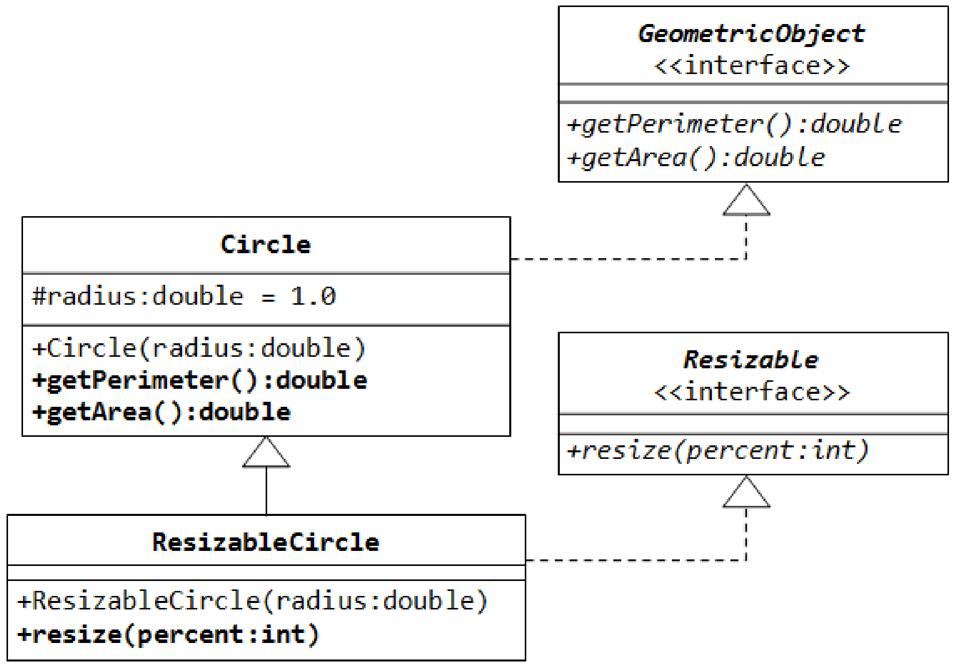
Suppose that we have a set of objects with some common behaviors: they could move up, down, left or right. The exact behaviors (such as how to move and how far to move) depend on the objects themselves. One common way to model these common behaviors is to define an *interface* called `Movable`, with `abstract` methods `moveUp()`, `moveDown()`, `moveLeft()` and `moveRight()`. The classes that implement the `Movable` interface will provide actual implementation to these `abstract` methods.

For the `MovablePoint` class, declare the instance variable `x`, `y`, `xSpeed` and `ySpeed`. For the `MovableCircle` class, use a `MovablePoint` to represent its center (which contains four variable `x`, `y`, `xSpeed` and `ySpeed`). In other words, the `MovableCircle` composes a `MovablePoint`, and its `radius`.



Question 7

1. Write the `interface` called `GeometricObject`, which declares two `abstract` methods: `getParameter()` and `getArea()`.
2. Write the implementation class `Circle`, with a protected variable `radius`, which implements the interface `GeometricObject`.
3. Write a test program called `TestCircle` to test the methods defined in `Circle`.
4. The class `ResizableCircle` is defined as a subclass of the class `Circle`, which also implements an interface called `Resizable`, as shown in diagram. The interface `Resizable` declares an `abstract` method `resize()`, which modifies the dimension (such as `radius`) by the given percentage. Write the interface `Resizable` and the class `ResizableCircle`.
5. Write a test program called `TestResizableCircle` to test the methods defined in `ResizableCircle`.


GeometricObject
<<interface>>
+getPerimeter():double
+getArea():double
Circle
#radius:double = 1.0
+Circle(radius:double)
+getPerimeter():double
+getArea():double
Resizable
<<interface>>
+resize(percent:int)
ResizableCircle
+ResizableCircle(radius:double)
+resize(percent:int)