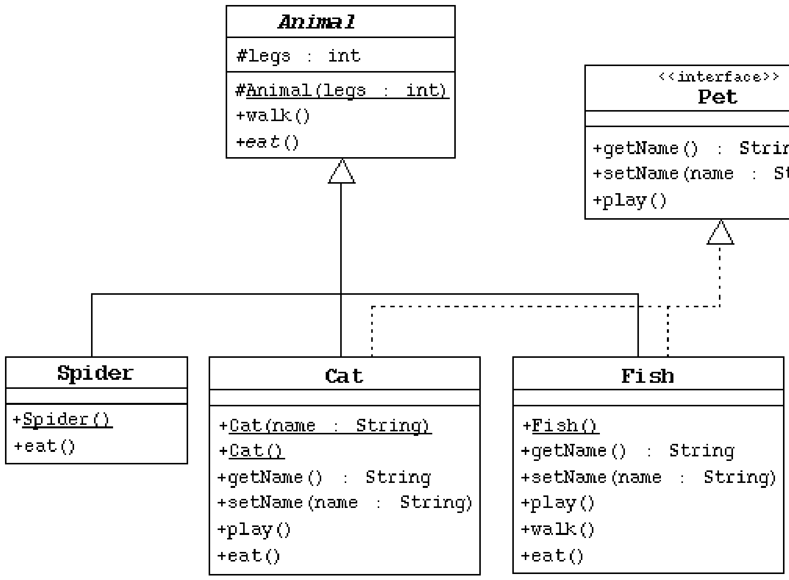
Lab 6: Abstract classes and interfaces

Lab Objective: After this lab, the student should be able to:

- Use abstract classes in Java
- Use interfaces in Java
- Design a system with interfaces and/or abstract classes

<u>Part 1:</u> Hierarchy of animals

Consider the hierarchy of animals in the following figure that is rooted in the abstract class `Animal`. Some of the animal classes implement the interface called `Pet`.



1. Create the `Animal` class, which is the **abstract** superclass of all animals.
   1. Declare a protected integer attribute called `legs`, which records the number of legs for this animal.
   2. Define a protected constructor that initializes the `legs` attribute.
   3. Declare an **abstract method** `eat`.
   4. Declare **a concrete method** `walk` that prints out a string about how the animals walks (include the number of legs).
2. Create the `Spider` class.
   1. The `Spider` class extends the `Animal` class.
   2. Define a default constructor that calls the superclass constructor to specify that all spiders have eight legs.
   3. Implement the `eat` method.
3. Create the `Pet` interface.
4. Create the `Cat` class that extends `Animal` and implements `Pet`.

   1. This class must include a `String` attribute to store the name of the pet.
   2. Implement the `Pet` interface methods.
   3. Implement the `eat` method.
5. Create the `Fish` class. Override the `Animal` methods to specify that fish can't walk and don't have legs.
6. To simulate multiple inheritance, we add a new interface `Hunter` with a method `hunts`. Make the classes `Spider, Cat and Fish` implement the interface Hunter and update their implementations so that the spider hunts insects, the cat hunts mice and the fish hunts another smaller fish
7. Create the TestAnimals program. Have the main method create and manipulate instances of the classes you created above. Start with:

```
Fish d = new Fish();
Cat c = new Cat("Fluffy");
Animal a = new Fish();
Animal e = new Spider();
Pet p = new Cat();
```

   Experiment by: a) calling the methods in each object,
   b) casting objects.

<u>Part 2:</u> Borrowing books

We want to develop system that records the books that have been borrowed from a library. There are three kinds of books: regular books, reference books, and audio books.

Reference books can be taken out for just two days, while other kinds of books may be borrowed for two weeks. The overdue fees are 50 dinars per day for reference books and regular books, and 70 dinars per day for audio books.

Audio books and regular books have both authors and titles; reference books only have titles.

- Design the interfaces/abstract classes and classes that can represent the library borrowing system.
- Design the method daysOverdue that consumes the number that represents today in the library date-recording system and produces the number of days this book is overdue. If the number is negative, the book can still be out for that many days.
- Design the method isOverdue that produces a boolean value that informs us whether the book is overdue on the given day.
- Design the method computeFine that computes the fine for a book, if it is returned on the given day.

For all methods, think carefully whether they should be designed being implemented solely in the abstract class, implemented solely in the concrete classes, or implemented in the abstract class and then overridden in some of the concrete classes.

Part 3: Drawing Shapes

Reconsider Part 2 from Lab5. Provide a better design of classes by making use of abstract classes and interfaces.

Use the AWT and Swing packages to create a graphic panel for drawing the basic shapes (circle, square, etc.).

Prompt the uses to insert a number of basic shapes which you store in the same array then draw the array content in the panel using polymorphism.