Recap:

- Inheritance allows for creating a hierarchy of classes where all of the general behaviors are pulled out into the super class.

- Each subclass then adds additional specializations:
  - New attributes
  - New methods
  - Overriding existing methods
Making things Final

- In some cases we want to prevent a subclass from overriding a method.
  - To do this we declare the method as `final`.
  - For example, we may not want debit or credit methods to be changed outside of our BankAccount class for security reasons.

- In other cases we may declare an entire class as `final`.
  - This will prevent any subclasses to be made from the class.

An Example

- Now on to a new example
- Suppose we wanted to create a class `Animal` are animals with have the following attributes:
  - `name`
  - `Age`

- We want our animals to be able to respond to the following messages:
  - `What is your name?`
  - `How old are you?`
  - `Speak.}`
So we implement the methods:
- public String getName() – returns its name.
- public int getAge() – returns its age.

But how do we implement public String speak() in the Animal class? It depends on the what type of animal it is.

There are situations that you may want to define class and its interface, without providing a complete implementation of each and every method in the class.

We use the abstract keyword to do this.

This ensures that any subclasses are required to be implemented in the subclass, but not in the super class.
Because abstract classes are not fully implemented, and have no constructors you cannot instantiate them.

The purpose is to create subclasses that fully implement the general behaviors.

This allows for the use of polymorphism.

An example:

```java
public abstract class Animal{
    private String name;
    private int age;

    public abstract String speak();

    public String getName(){
        return name;
    }

    public int getAge(){
        return age;
    }

    protected void setName(String aName){
        name = aName;
    }

    protected void setAge(int aAge){
        age = aAge;
    }
}
```
public class Cat extends Animal
{
    public Cat(String itsName, int itsAge) {
        setName(itsName);
        setAge(itsAge);
    }

    public String speak() {
        return "Meow";
    }
}

The example in the book uses a Shape class.

- In this example all Shapes are expected to be able to draw themselves.
  (pp 145-147)

- He creates an abstract class Shape, with an abstract draw method.
Contract

- The **public interface** is defined within the API.

- The public interface includes all method signatures that **must be implemented** in the class.

- When using an **abstract** class, the methods in the interface are identified and defines what another developer is responsible for when **implementing** a subclass.

Abstract Classes in UML

- Abstract classes and methods can be identified in UML by **italics**.
In summation...

- Abstract classes by design take advantage of inheritance and polymorphism.

- We will implement the general behaviors that are in common with all subclasses in the abstract class.

- We identify the methods that subclasses must implement, but leave that specialized implementation to be completed in the subclasses.