***Java Directions****: If you are doing this as a Java exercise, you will need to create a class file for each class and test it in a main method. Sample Java main method follows:*

*Public static void main(String[] args) {*

*}*

**The classes you Use in this exercise**

You should have already the classes listed in the diagram below. We will add a few classes in this exercise in order to practice Java:

* Polymorphism
* Abstract Classes
* Interfaces

*Class names are bold, instance variables are the not bold*

**Important Note:** You should have already created the Animal, Mammal, Canine, and Feline classes!!!

**Existing Classes**

**Animal Class**

* Properties/attributes
  + Gender
* Methods
  + getGender and setGender

***Continued on Reverse***

**Mammal Class *(Subclass of Animal)***

1. This class is a subclass of the Animal class.
   1. Attributes:
      1. age
      2. weight
   2. Methods:
      1. getAge()
      2. setAge()
      3. getWeight()
      4. setWeight()
      5. move() - Just print “*Mammal Moves*” in this method.
      6. makeNoise() - Just print “*Mammal Makes a Noise*” in this method.
2. Constructor:
   1. Accepts the values gender, age, and weight as parameters and sets the appropriate attributes.
   2. Prints the values of gender, age, and weight using the appropriate “get” methods.

**Feline class *(Subclass of Mammal)***

1. This class is a subclass of the Mammal class.
   1. Attributes:
      1. breed
      2. isDangerous
   2. Methods:
      1. Getters and setters for all the attributes above.
2. Constructor:
   1. Accepts the values gender, age, weight, breed, and isDangerous as parameters and sets the appropriate attributes.
   2. Prints the values of age, breed, and isDangerous, using the appropriate “get” methods.
3. The makeNoise() method that:
   1. Prints “Feline says purr”
   2. Calls the makeNoise() method of the superclass.

**Canine Class *(Subclass of Mammal)***

1. This class is a subclass of the Mammal class.
   1. Attributes:
      1. breed
      2. isDangerous
   2. Methods:
      1. Getters and setters for all the attributes above.
2. Constructor:
   1. Accepts the values gender, age, weight, breed, and isDangerous as parameters and sets the appropriate attributes.
   2. Prints the values of age, breed, and isDangerous, using the appropriate “get” methods.
3. The makeNoise() method:
   1. Prints “Canine says howl”
   2. Calls the makeNoise() method of the superclass.

**PolyMorphism**

Polymorph – Many Forms (***poly***=many; ***morph***=form)

In the example below, assume that pepperoni inherits from Pizza.  Pizza is the **compile-time type** (declared type)**.**  This code will not compile unless there is a class named Pizza.

Pizza pizzaObj = new Pepperoni

Pepperoni is the **run-time type** (*actual type*). Upon execution, any methods that use the pizzaObj will attempt to find them in the Pepperoni class, before looking in the Pizza class.

In our above example, If a method in the Pizza class calls a method that exists in both Pizza and Pepperoni, the method in Pepperoni executes, because it is defined as the run-time type.

**Before Continuing**

Read the following article and answer the quiz questions:

<http://interactivepython.org/runestone/static/JavaReview/OOBasics/ooPoly.html>

**The PolyTest() Class**

This class receives no parameters into its constructor and contains no instance variables. We will use it to practice one of the best uses of polymorphism.

We will create instances of the Feline and Dog classes, add them to the same ArrayList() and modify (and print) the animal names. Everything for this example can be done in the main method or placed in an appropriately named method.

1. Create a class named PolyTest().
   * Properties/attributes
     1. None
   * Methods
     1. None
   * Constructor: Prints the following:
     1. "Instance of: Animal created."
2. Define a main method. Inside the main method:
   * Create two instances of the Feline() class using “Animal” as the type and cat1 and cat2 as the names of the instance variables.

Animal cat1 = new Feline();

* + Create two instances of the Canine() class using “Animal” as the type and dog1 and dog2 as the names of the instance variables.
  + Define an ArrayList() named animalList that uses Animal as the type.
  + Add your four objects to the list (cat1 cat2 dog1 dog2)
  + Print the message "\n \*\*\* Polymorphic test \*\*\* "
  + Print the message " \* Printing all genders BEFORE MODIFICATION\* "
  + Use a for loop to print the gender of all items in the animalList.
  + Use a for loop to set the gender of all items in the animalList. To “unknown”.
  + Print the message " \* Printing all genders AFTER MODIFICATION\* "
  + Use a for loop to print the gender of all items in the animalList.
  + **Master Coders**
    1. Move all the print logic into a method named printAnimalGenders(). It will receive an ArrayList of the type Animal as a parameter.
    2. How can you call this method?
    3. Do we need to add a modifier to make this work?

**Abstract Classes**

An abstract class:

* Cannot be instantiated
  + They exist only to be subclassed
* Often have at least one abstract method (a method that has the keyword abstract in the declaration and no method body)
  + This is not always the case.
* Can have constructors, fields, and methods with bodies (non-abstract methods)
  + Can have real, executable, code.
* Abstract methods can only be defined within abstract classes.

**Before Continuing**

Read the following article and answer the quiz questions:

<http://interactivepython.org/runestone/static/JavaReview/OOBasics/ooAbstract.html>

**The AbstractPizza() Class**

This class is defined as abstract. It receives no parameters into its constructor and contains no instance variables.  We will use it to practice abstract classes.

1. Create a class named AbstractPizza().
   1. Properties/Attributes
      1. Size (possible values: 8, 12, and 16)
      2. Crust (possible values: “thin”, “hand tossed”, “deep dish”)
2. Methods
   1. None
3. Constructor:
   1. Accepts the instance variables size and crust.
   2. Prints "Instance of: AbstractPizza created and displays the values for size and crust."
4. Create setters and getters for the size and crust attributes.
5. Create the following abstract methods:
   1. buildPizza
      1. employeeId int
      2. buildTimestamp double
   2. deliverPizza
      1. employeeId int
      2. deliveredTimestamp double

**The Pepperoni() Class**

This is a typical class that will inherit from AbstractPizza() and, eventually InterfacePizza().

1. Create a constructor for the Pepperoni() class that:
   1. Accepts and sets values for size and crust
   2. Calls the constructor for the AbstractPizza() class.
2. Create the buildPizza and deliverPizza methods. Each method should print a message that:
   1. Describes itself
   2. Displays the parameter values for the method.
3. Define a main method. Inside the main method:
   1. Create an instance of the Pepperoni() class.
   2. Call the buildPizza method using the following values:
      1. employeeID 100
      2. buildTimeStamp 21.50
   3. Call the deliverPizza method using the following values:
      1. employeeID 999
      2. deliveredTimeStamp 22.15
4. Print the message “Transaction Complete - Pizza baked and delivered.”

**Java Interfaces**

A Java **interface** is a special type of class. The only type of methods it can contain are **public abstract methods**. An **abstract** method is one that only has a method header and no body (no code). You define interfaces to define what a class needs to be able to do to **implement** an interface.

# **Interfaces are like Contracts**

You can also think of an interface as a contract. Classes that implement the interface agree to provide code for the methods that are defined in the interface.

**Before Continuing**

Read the following article and answer the quiz questions:

<http://interactivepython.org/runestone/static/JavaReview/ListBasics/listInterface.html>

**The InterfaceBake() Class**

This interface contains some constants used when baking pizzas. It sets cook time and temperature.

**Constants**

DEEP\_DISH\_COOK\_TIME = 11.30

DEEP\_DISH\_COOK\_TEMP = 450

HAND\_TOSSED\_COOK\_TIME = 9.0

HAND\_TOSSED\_COOK\_TEMP = 400

THIN\_COOK\_TIME = 7.5

THIN\_COOK\_TEMP = 375

**The bake Method**

Four input variables (cookTime, cookTemp, startTimestamp, endTimestamp)

***Continued on next page***

**Modify the Pepperoni() class**

1. Add the code that includes the InterfaceBake() interface in the Pepperoni() class.
2. Test your class
   1. Did it work?
3. Add a bake method to your Pepperoni() class. This method should:
   1. Accept the four variables (cookTime, cookTemp, startTimestamp, endTimestamp)
   2. Print the values of the values of the variables.
   3. Print the message “baking”
   4. Print the message “Ready for delivery”
4. Add a call to the bake method within your main method just after the buildPizza method call.
   1. Make up your own Timestamp values when calling the bake method.
5. Test your class again using constants for cookTime and cookTemp.
   1. Did it work this time?