Object Oriented Programming

Week 3 Part 3
Polymorphism

Lecture

- What is Polymorphism
- Examples of Polymorphism

What is Polymorphism?

Polymorphism

- Polymorphism lets programs use the same name for different functions.
  - It comes from the Greek poly (many) morph (form)
  - The name was adapted from biology where it means the same animal with different forms

Types of Polymorphism

- There are three types of Polymorphism
  - **Ad Hoc Polymorphism**: the meaning of the function is determined by the type of a functions parameters and its return type.
  - **Sub-typing Polymorphism**: the meaning of the function is determined by the type of the object on which is is called
  - **Parametric Polymorphism**: writing a function or method generically so it can manage different types.
    - These are called *Generic* and will be dealt with later.

Ad Hoc Polymorphism

- The simplest type of Polymorphism determines the behavior of a function or operator based on its operands
  - E.g. Addition of ints in the sum of the two values; additions of two strings is their concatenation
    - 1 + 2 == 3; “1” + “2” == “12”
  - We may use the same method name for different functions if the parameters are different.
  - We may use this do provide different Constructors:

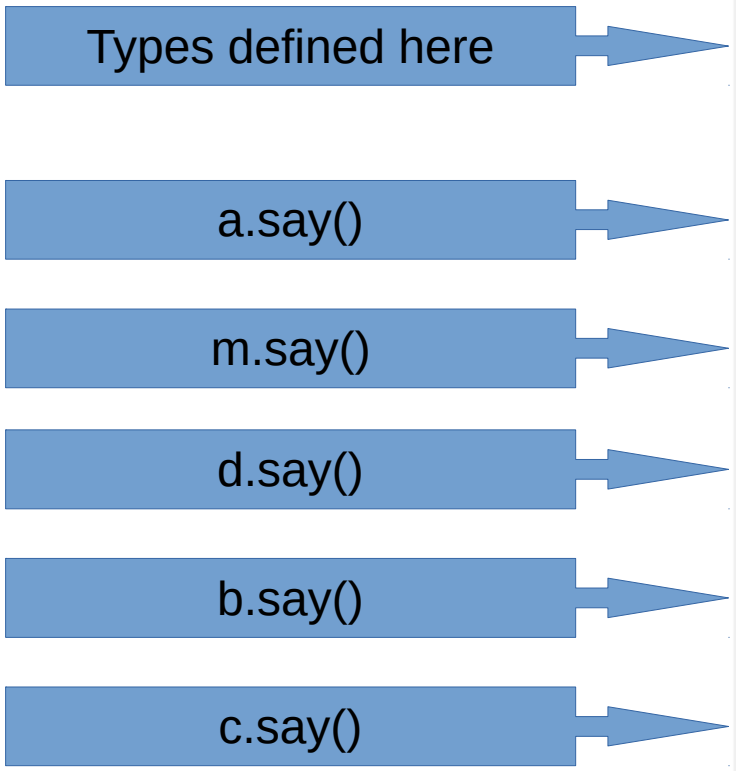
```
public Dog(String food) {
    super(food);
}
```

```
public Dog() {
    super("Meat");
}
```

Sub-type Polymorphism

- Subclasses inherit methods, which will behave the same way regardless of the subclass.
- Subclasses may override methods, which causes them to behave differently.
- The behavior is determined by the type.
- We have seen this in our Test file.

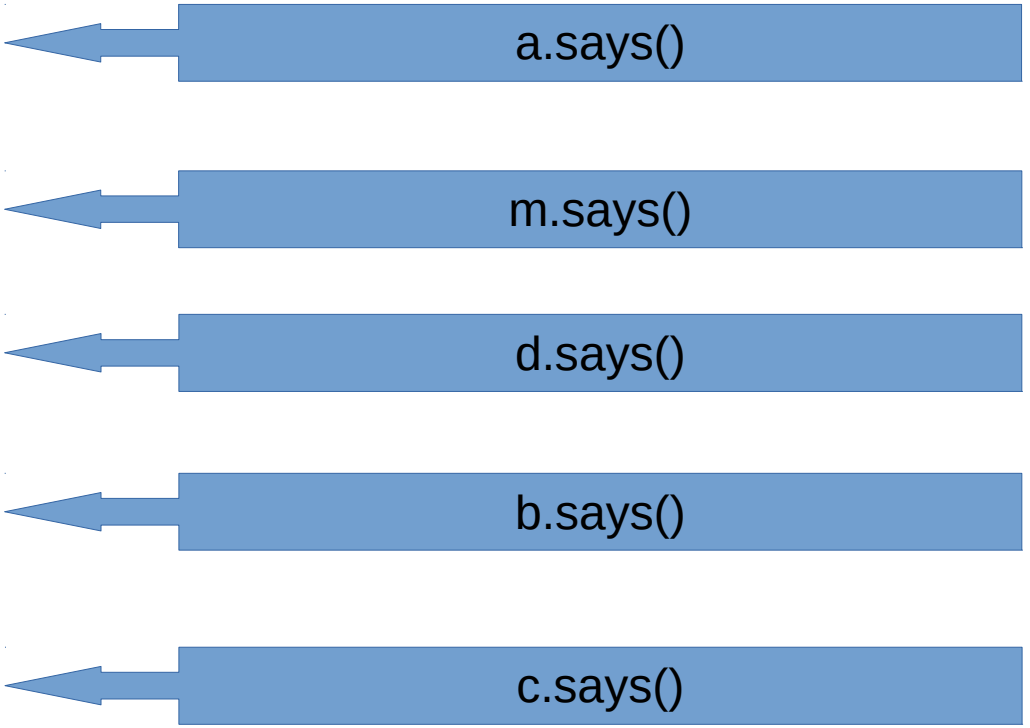
Sub-type Polymorphism: Test




```
package animals;

public class Test {

    public static void main(String[] args) {
        Animal a = new Animal("Food");
        Mammal m = new Mammal("Milk");
        Dog d = new Dog("Meat");
        Bird b = new Bird("Food");
        Crow c = new Crow("Seeds");

        System.out.println("Animals eat " + a.getFood());
        System.out.println(a.says());
        System.out.println("Mammals eat " + m.getFood());
        System.out.println("Mammal young are " + m.getOffspring());
        System.out.println(m.says());
        System.out.println("Dogs eat " + d.getFood());
        System.out.println("Dog young are " + d.getOffspring());
        System.out.println(d.says());
        System.out.println("Birds eat " + b.getFood());
        System.out.println("Bird young are " + b.getOffspring());
        System.out.println(b.says());
        System.out.println("Crows eat " + c.getFood());
        System.out.println("Crow young are " + c.getOffspring());
        System.out.println(c.says());
    }

}
```

Test Output

```
Animals eat Food
Animals say many different things
Mammals eat Milk
Mammal young are Live
Mammals say many different things
Dogs eat Meat
Dog young are Live
Dog goes woof
Birds eat Food
Bird young are Eggs
Bird says tweet
Crows eat Seeds
Crow young are Eggs
Crow goes caw
```



Each of the says() method produces a different output based on the type for which it is a method.

Subtype Polymorphism without vars

- This is not that different from ad hoc polymorphism
  - You could think of the variables against which the method is being called as another parameters
- However, you do not need to have a variable of that type.
  - Subclasses can be assigned to arrays of its superclass
  - The type of the object in the array determines the action of the method, not the type of the list

Advanced Subtype Polymorphism

List of Animals

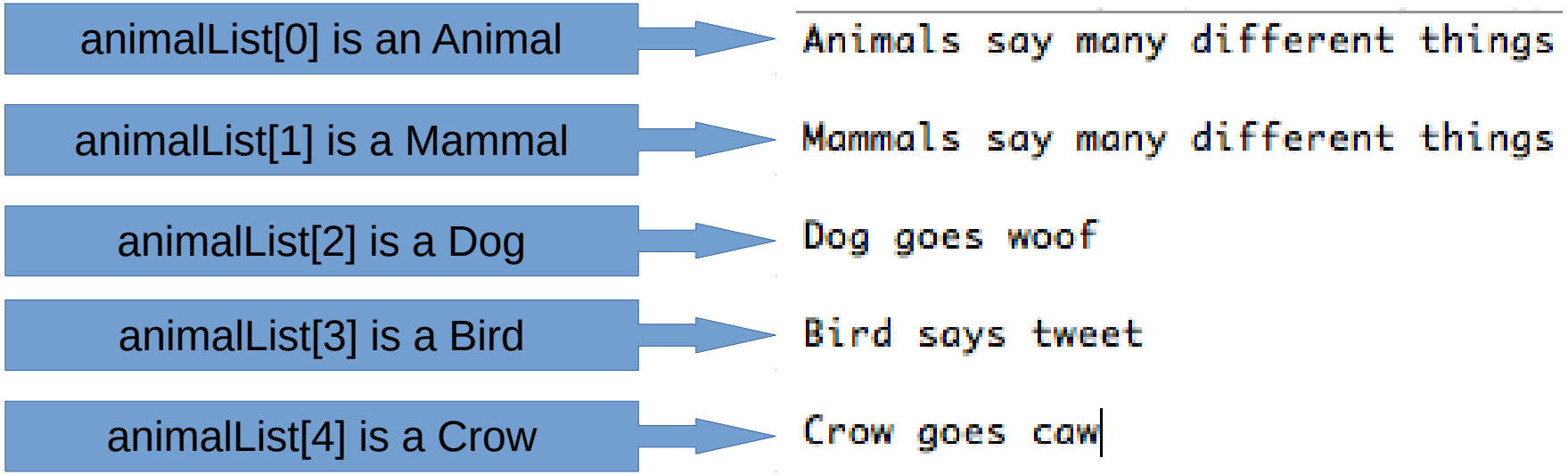
Initialized to different types

Constructor called

Elements of list call says()

```
package animals;

public class PolymorphismDemo {

    private Animal animalList[] = new Animal[10];

    public PolymorphismDemo() {
        animalList[0] = new Animal("Food");
        animalList[1] = new Mammal("Food");
        animalList[2] = new Dog("Meat");
        animalList[3] = new Bird("Worms");
        animalList[4] = new Crow("Seeds");
    }

    public Animal[] getAnimalList() {
        return animalList;
    }

    public static void main(String args[]) {

        PolymorphismDemo p = new PolymorphismDemo();
        for (int i = 0; i < 5; i ++) {
            System.out.println(p.getAnimalList()[i].says());
            System.out.println("");;
        }
    }

}
```

PolymorphismDemo Output



Each element of the list calls the appropriate says() method.