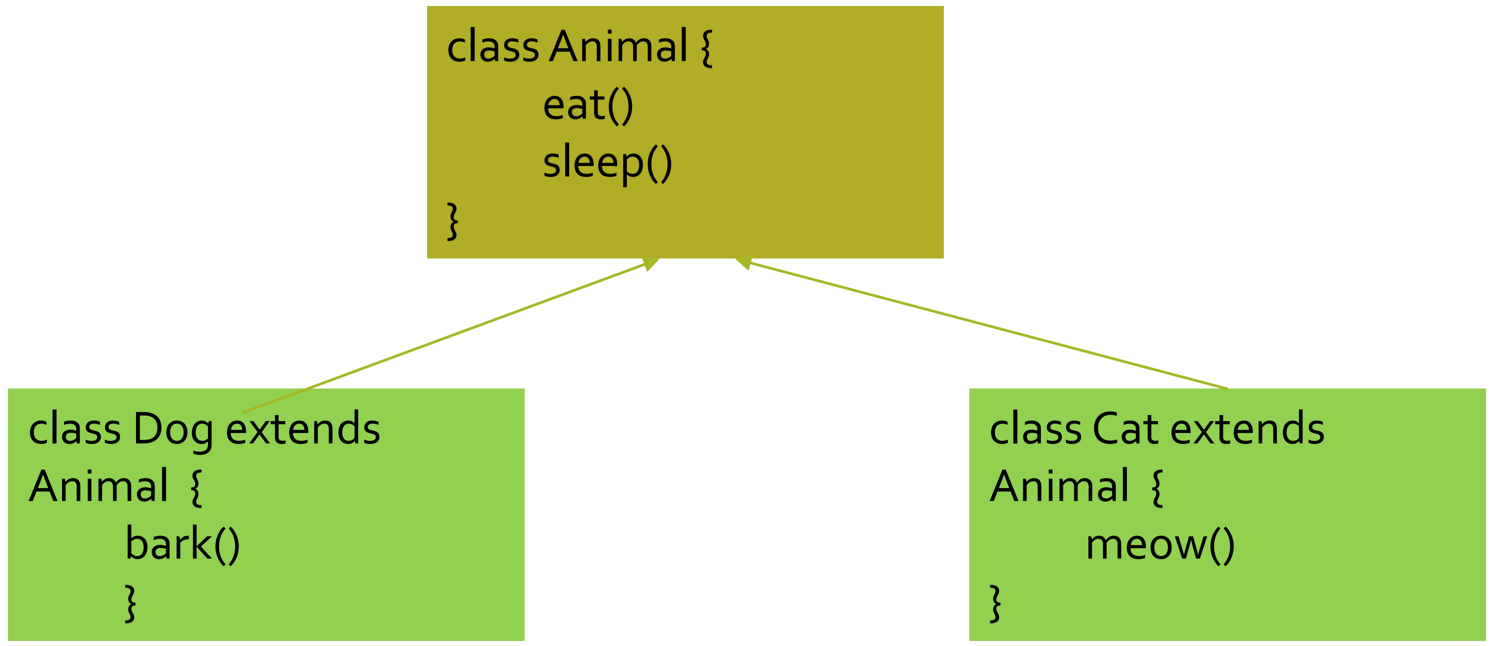
CHAPTER 8 – INHERITANCE AND POLYMORPHISM

Object Oriented Programming

Inheritance

- Inheritance allows us to define a new class from an existing class.
- We use “extends” keyword to inherit from a class



```
class Animal {
        eat()
        sleep()
}
```

```
class Dog extends
Animal  {
        bark()
        }
```

```
class Cat extends
Animal  {
        meow()
}
```

Terms

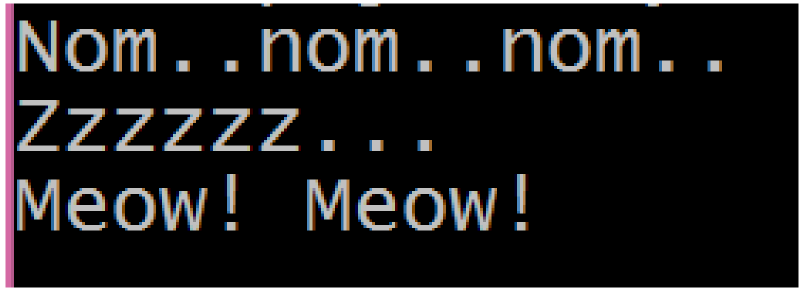
- Parent class, superclass, and base class refer to the class that another class inherits from
- Child class, subclass, and derived class refer to a class that inherits from another class

Save as MyPet.java

```
class Animal {
   public void eat() {
      System.out.println("Nom..nom..nom..");
   }
   public void sleep() {
      System.out.println("Zzzzzz...");
   }
}

class Cat extends Animal {
   public void meow() {
      System.out.println("Meow! Meow! ");
   }
}
```

Save as MyPet.java

```
class MyPet {
   public static void main(String[] args) {

     Cat garfield = new Cat();
      garfield.eat();
      garfield.sleep();
      garfield.meow();

   }
}
```



```
Nom..nom..nom..
Zzzzzz...
Meow! Meow!
```

What is method overriding?

- If the same method is defined in both the superclass and subclass, **the method in the subclass overrides the method in the superclass.**

Change the Cat class from previous example with this

```
class Cat extends Animal {
   public void meow() {
      System.out.println("Meow! Meow! ");

   }
   public void eat() {
      System.out.println("Meat and fish only, please!");
   }
}
```

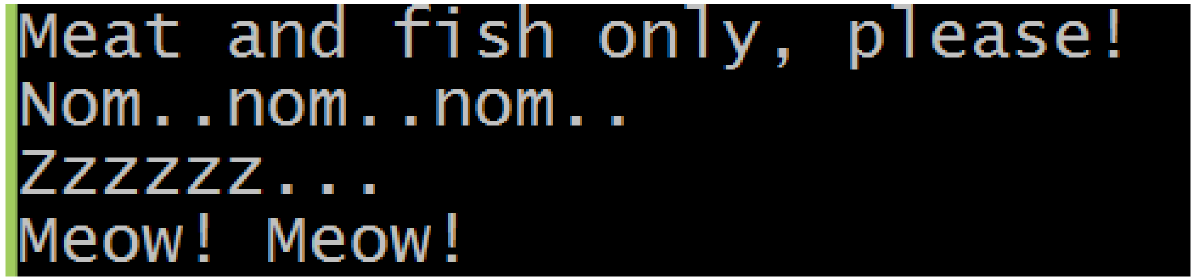
The Super keyword

- To call methods of the superclass that is overridden in the subclass.
- To access attributes (fields) of the superclass if both superclass and subclass have attributes with the same name.
- To explicitly call superclass constructor from the subclass constructor.

Example 1 : Calling methods of the superclass that is overridden in the subclass

In class Cat, change the codes to method eat:

```
public void eat() {
     System.out.println("Meat and fish only, please!");
      super.eat();
  }
```



```
Meat and fish only, please!
Nom..nom..nom..
Zzzzzz...
Meow! Meow!
```

Example 2 : Accessing attributes of superclass if both superclass and subclass have attributes with the same name

Add an attribute in class Animal and class Cat:

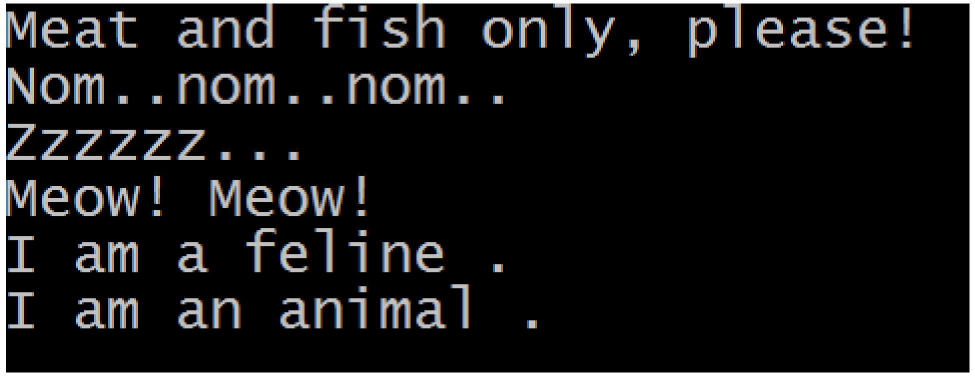
```
class Animal {
   protected String type="animal";
   public void eat() {
      System.out.println("Nom..nom..nom..");
   }
   public void sleep() {
      System.out.println("Zzzzzz...");
   }
}
```

Example 2 : Accessing attributes of superclass if both superclass and subclass have attributes with the same name

```
class Cat extends Animal {
   protected String type= "feline";
   public void meow() {
      System.out.println("Meow! Meow! ");

   }
   public void eat() {
      System.out.println("Meat and fish only, please!");
          super.eat();
   }
    public void printType() {
    System.out.println("I am a " + type  +".");
    System.out.println("I am an " + super.type +".");
  }

}
```

Example 2 : Accessing attributes of superclass if both superclass and subclass have attributes with the same name

```
class MyPet {
   public static void main(String[] args) {

      Cat garfield = new Cat();
         garfield.eat();
         garfield.sleep();
         garfield.meow();
         garfield.printType();

   }
}
```



```
Meat and fish only, please!
Nom..nom..nom..
Zzzzzz...
Meow! Meow!
I am a feline .
I am an animal .
```

Example 3: explicitly call superclass constructor from the subclass constructor

```
class Animal {
     Animal() {
    System.out.println("I am an animal.");
     }
     Animal(String type) {
    System.out.println("Type: "+ type);
  }
}
```

Example 3: explicitly call superclass constructor from the subclass constructor

```
class Cat extends Animal {
   Cat(){
      super("mammal");
    System.out.println("I am a feline.");
  }
}
class MyPet {
   public static void main(String[] args) {
     Cat garfield = new Cat();
    }
}
```

Polymorphism

- Polymorphism simply means more than one form. The same entity (method or operator or object) can behave differently in different scenarios.
- In Java, Polymorphism can be divided into two types:
  - Run-time Polymorphism

    Run-time polymorphism can be achieved through method overriding
  - Compile-time Polymorphism
    - The compile-time polymorphism can be achieved through method overloading and operator overloading in Java.

Example of method overriding

```
abstract class Animal {
   public abstract void Talks();
}
class Cat extends Animal {
   public void Talks() {
      System.out.println("Meow meow meow.");
   }
}
class Bird extends Animal {
   public void Talks() {
      System.out.println("Chirp chirp chirp.");
   }
}
class MyPet {
   public static void main(String[] args) {
    Cat garf = new Cat();
     garf.Talks();
       Bird tweety = new Bird();
       tweety.Talks();
   }
}
```

Example of method overloading

```
class Demo {
 public void displayPattern(){
   for(int i = 0; i < 10; i++) {
     System.out.print("*");
   }
 }
 public void displayPattern(char symbol) {
   for(int i = 0; i < 10; i++) {
     System.out.print(symbol);
   }
 }
}
class Main {
 public static void main(String[] args) {
   Demo d1 = new Demo();
   d1.displayPattern();
   System.out.println("\n");
   d1.displayPattern('#');
 }
}
```