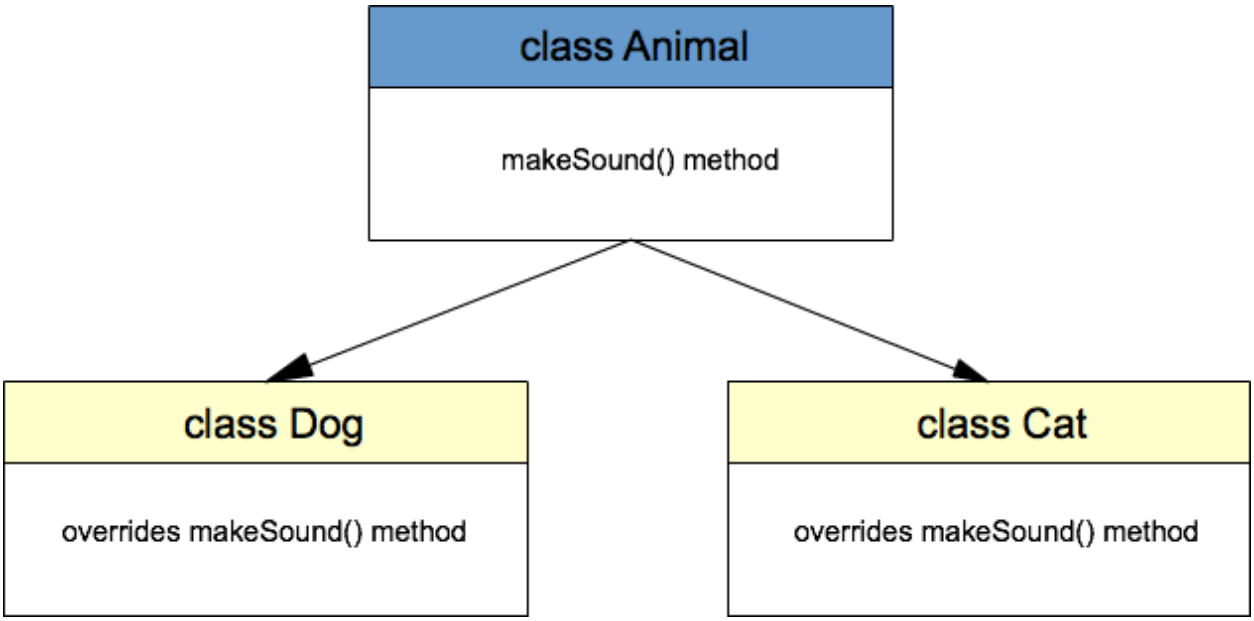
Polymorphism Using Java

What is Polymorphism:
The term Polymorphism comes from the Greek language, and means “many forms”. Polymorphism in Java allows subclasses of a class to define their own unique behaviors and yet share some of the same functionality of the parent class.

Example of Polymorphism:
In this example we will create 3 classes to demonstrate polymorphism and one class to test the concept. Our superclass is called Animal. The successors of the animal class are Dog and Cat classes. Those are animals too, right? That’s what polymorphism is about – you have many forms of the same object with slightly different behavior. To demonstrate this we will use a method called makeSound() and override the output of this method in the successor classes.



Java polymorphism example

The generalized animal class will output some abstract text when we call the makeSound() method:

```
1.  package net.javatutorial;
2.
3.  public class Animal {
4.
5.          public void makeSound() {
6.                  System.out.println("the animal makes sounds");
7.          }
8.
9.  }
```

The `Dog` class, which extends `Animal` will produce slightly different result – the dog will bark. Toachieve this we extend the `Animal` class and override the `makeSound()` method

```
1.  package net.javatutorial;
2.
3.  public class Dog extends Animal{
4.
5.          @Override
6.          public void makeSound() {
7.                  System.out.println("the dog barks");
8.          }
9.
10. }
```

Obviously we have to do the same to our `Cat` class to make the cat meow.

```
1.  package net.javatutorial;
2.
3.  public class Cat extends Animal {
4.
5.          @Override
6.          public void makeSound() {
7.                  System.out.println("the cat meows");
8.          }
9.
10. }
```

Finally lets test our creation.

```
1.  package net.javatutorial;
2.
3.  public class PolymorphismExample {
4.
5.          public static void main(String[] args) {
6.                  Animal animal = new Animal();
7.                  animal.makeSound();
8.                  Dog dog = new Dog();
9.                  dog.makeSound();
10.                 animal = new Cat();
11.                 animal.makeSound();
12.         }
13.
14. }
```

First we create a general `Animal` object and call the `makeSound()` method. We do the same for a newly created `Dog` object. Now note the call to `animal = new Cat()` – we assign a new `Cat` object to an `Animal` object. Cats are animals, remember? So, we can always do this:

```
1.  Animal animal = new Cat();
```

By calling the `makeSound()` method of this object will actually call the overridden `makeSound()` method in the `Cat` class.

Finally, here is the output of the program

```
1.  the animal makes sounds
2.  the dog barks
3.  the cat meows
```

Reference: https://javatutorial.net/java-polymorphism-example

……