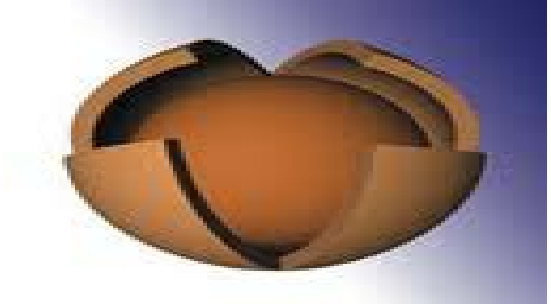
Object Orientation

- Any language is said to be object oriented if it supports following object properties:

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Data abstraction (or interfaces)



Encapsulation

- Encapsulation can be described as the mechanism in which we *"encapsulate"* our code in such a way that it can not be randomly accessed by other code outside the class.
- If we want to include encapsulation in our code then we have to do following things:

1. Always make instance variable private.
2. Always make public accessor methods and force calling code to use these methods instead of directly calling the instance variables.
3. Use naming convention set() and get() for these methods.

Problem scenario without encapsulation

- **public class Duck {**
- **private int size;**
- **public void display()**
- {
- **if (size<0)**
- System.*out.println("incorrect size");*
- **else if(size>10)**
- System.*out.println("bigger duck!!!");*
- **else if (size < 10)**
- System.*out.println("smaller duck!!");*

}

}

- public class Test {
- public static void main(String[] args)
- {
- Duck d = new Duck();

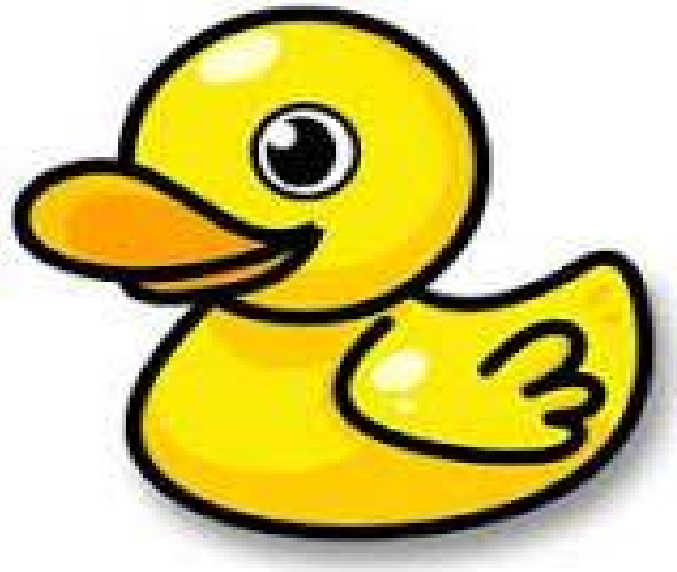
- d.size = 45;
- d.display();
- }
- }

Getters and Setters

- Getters and Setters are nothing but the methods which are used to “set” and “get” the value of instance variables.
- **Setters** **:** Setters catch the “value” of instance variable in it’s parameter and “set” or assign this value to the instance variable. Setters have always parameters and no return.
- **Getters:** Getters returns the value of a instance variable to it’s “caller”. It only returns the already set value of a instance variable. Getters don’t have parameters and always return something.

```
public class Duck {
private int size = 12;

public void setSize(int x)
{
if (x<=0);
else if(x>=25);
else
size = x;
}
public int getSize()
{
return size;
}
public void display()
{
if(size>10)
System.out.println("bigger duck!!!");
else if (size < 10)
System.out.println("smaller duck!!");
}
}
```

```
public class Test {
public static void main(String[] args)
{
Duck d = new Duck();

d.setSize(45);
System.out.println("the encapsulated size" +
d.getSize());
d.display();
}
}
```



Benefits of Encapsulation

- Code becomes more maintainable and flexible.
- In future we can change our code, without breaking some other code, which depends on our code.
- The class have total control over what is going to be stored over it’s fields.
- The user of the class don’t know how class stores the data.

Inheritance

- Inheritance can be defined as the process in which one object acquires the properties of others.
- By using inheritance the information becomes more manageable and in a hierarchical order.
- In other word we can say the Inheritance is the relationship between super class and subclass.

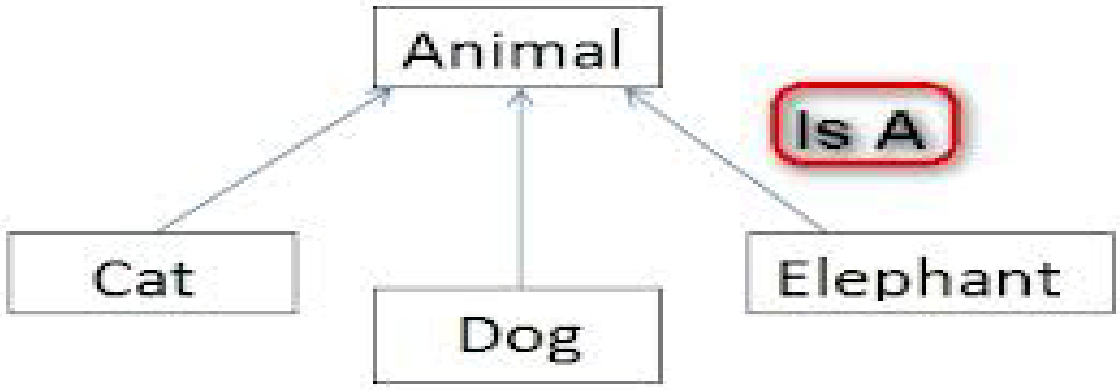
Super class vs. Sub class

- Super class:

This is the class which contains common features of all subclasses.

- Sub class:

This is the class which inherits all the features of super class.



Using Inheritance

- Inheritance relationship is created by using the keyword “extends”.

public class Animal {} // as superclass

- public class Dog extends Animal {} // Dog is subclass of Animal
- public class Cat extends Dog {} // Cat is subclass of Animal

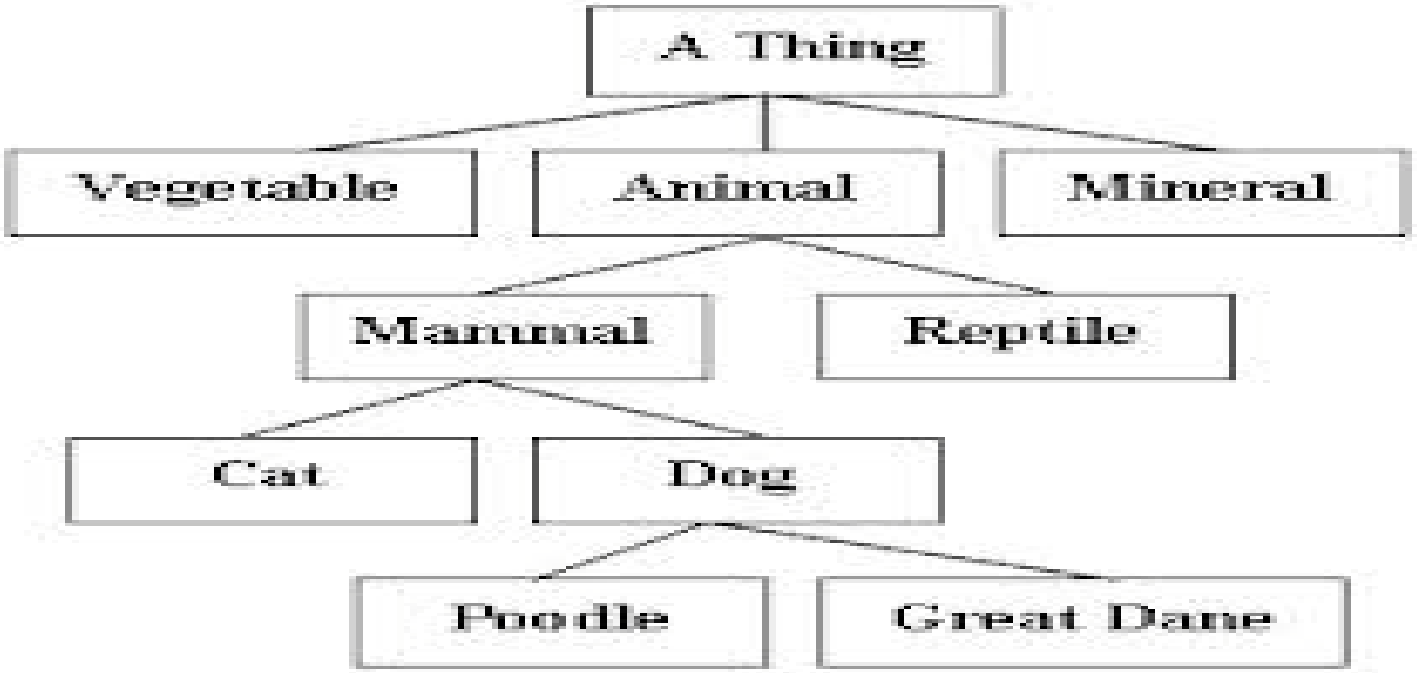
```
public class Animal {

public void eat()
{
System.out.println("eating habit");
}
public void roam()
{
System.out.println("raoming habit");
}
}
```

```
public class Cat extends Animal{

public void sound()
{
System.out.println("meow meow!!!");
}
}
```

```
public class Dog extends Animal{

public void sound()
{
System.out.println("woof woof!!!");
}
}
```

Inheritance Tree

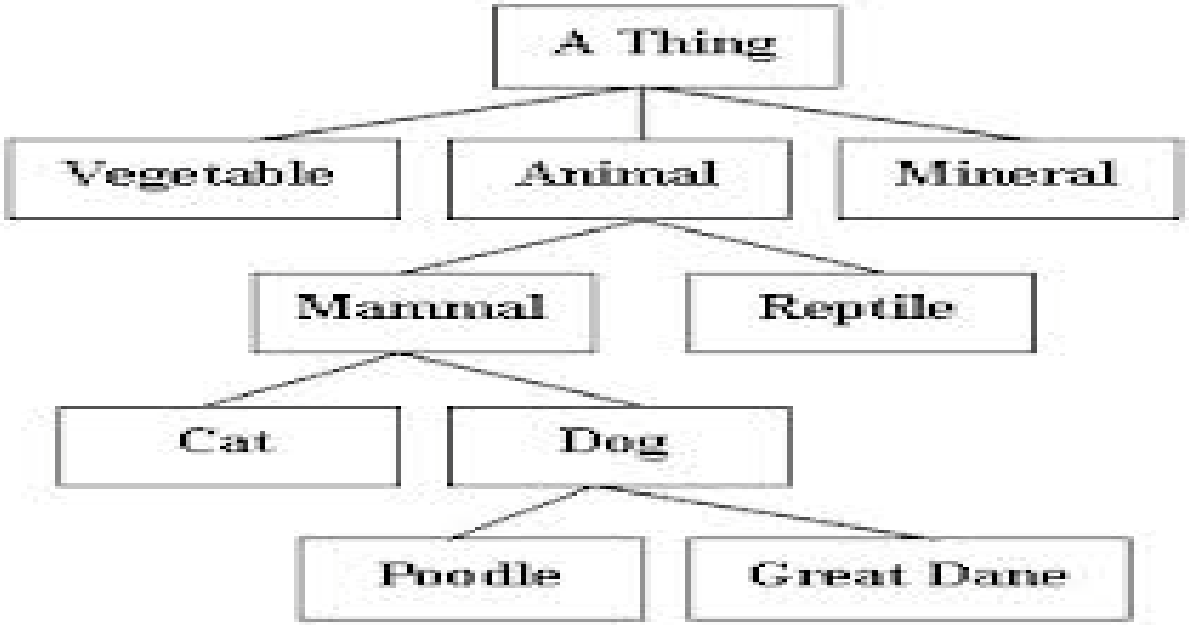
- Inheritance Tree is basically a tree structure (upside down) that maps inheritance hierarchies of the classes.



Method calling in Inheritance tree

- In inheritance tree the method calling is started from the lowest class in the tree.

e.g. if we call eat() method of “poddle” class then it starts searching from “Poddle” to “A Thing”.



Method Overriding

- Method Overriding is basically a ability to define a behavior which is specific to that sub class.
- Method Overriding RULES:

1. The arguments and the return type must be same as of superclass method.
2. The access level can’t be decreased.