Inheritance

Cat
age
weight
breathe()
walk()
climb()

Dog
age
weight
isDangerous
breathe()
walk()
bringStick()

Bird
age
weight
flyingSpeed
breathe()
walk()
Fly()

Cats, dogs and birds have age and weight.

They also can breathe and walk.

Actually, they are animals and every animal has age, weight and can breathe and walk.

Animal
age
weight
breathe()
walk()

Cat
climb()

Dog
isDangerous
bringStick()

Bird
flyingSpeed
fly()
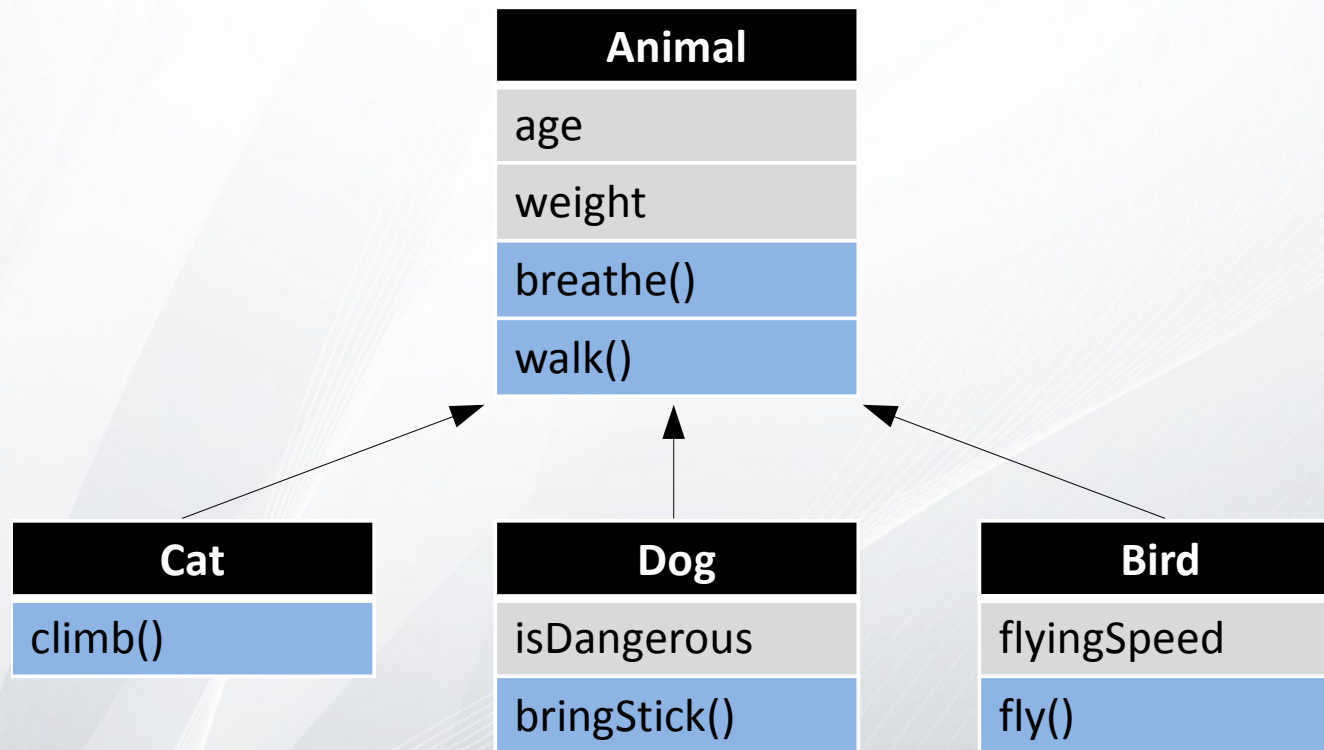
Is this the same?

Yes, but the common logic is only in the class Animal

Инheritance

- Inheritance is one of principles of the OOP

- Inheritance is the ability of a class to implicitly gain some(or all) members of another class

- Inheritance builds a relationship between classes

- The class that gains the functionality is called subclass (child class or derived class). The other class is called base(parent or superclass)

- The main purpose of inheritance is reusability of the code

- Inheritance in java is provided through the keyword *extends* i.e the subclass class extends the superior

- Each class can extend only one class

```java
package lesson09.animalExample;

public class Animal {
    int age;
    double weight;

    void breathe() {
        System.out.println("Breathing...");
    }

    void walk() {
        System.out.println("Walking...");
    }
}
```

```java
public class Cat extends Animal{
    void climb() {
        System.out.println("Climbing...");
    }
}
```

```java
public class Dog extends Animal{
    boolean isDangerous;

    void bringStick() {
        System.out.println("Bringing the stick...");
    }
}
```

```java
public class Bird extends Animal{
    double flyingSpeed;

    void fly() {
        System.out.println("Flying...");
    }
}
```

Keyword extends

'is a' and a 'has a' relationship

- Two of the main techniques for code reuse are class inheritance and object composition

- 'is a' relationship is expressed with inheritance

- 'has a' relationship is expressed with composition

Example using classes House, Building and Bathroom

- Is a: House is a Building

- has a: House has a Bathroom

Inheritance vs Composition

- ## Inheritance is uni-directional:

  House is a Building. But Building is not a House.

- ## Inheritance uses *extends* key word

```
class Building {
    //...
}


class House extends Building {
    //...
}
```

Keyword extends

- Composition is used when House has a Bathroom. It is incorrect to say House is a Bathroom

- Composition simply means using instance variables that refer to other objects. The class House will have an instance which refers to a Bathroom object

```
class House {
    Bathroom bathroom;

    //...
}
```

Access modifiers in inheritance

- *public, private* and default(package)
- *protected* fields and methods is visible to the classes within the same package and in the child classes

Table with access levels

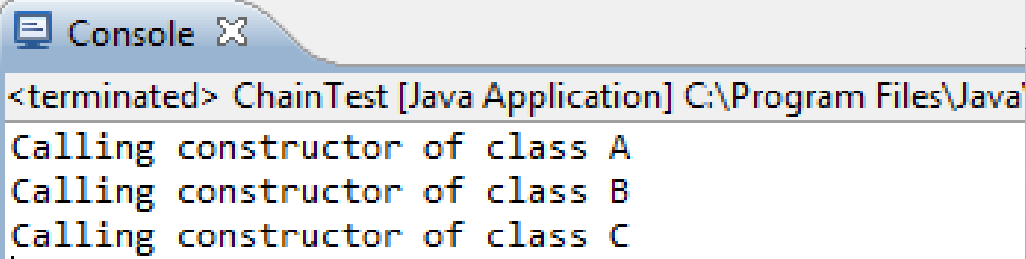|	Class	Package	Subclass (in the same pkg)	Subclass	World
public	Yes	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	Yes	No
default	Yes	Yes	Yes	No	No
private	Yes	No	No	No	No

- Invocation of a superclass constructor must be the first line in the subclass constructor

- Keyword *super* is used for this

- If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the default constructor of the superclass.

- In the previous case, If the super class does not have a default constructor, you will get a compile-time error.
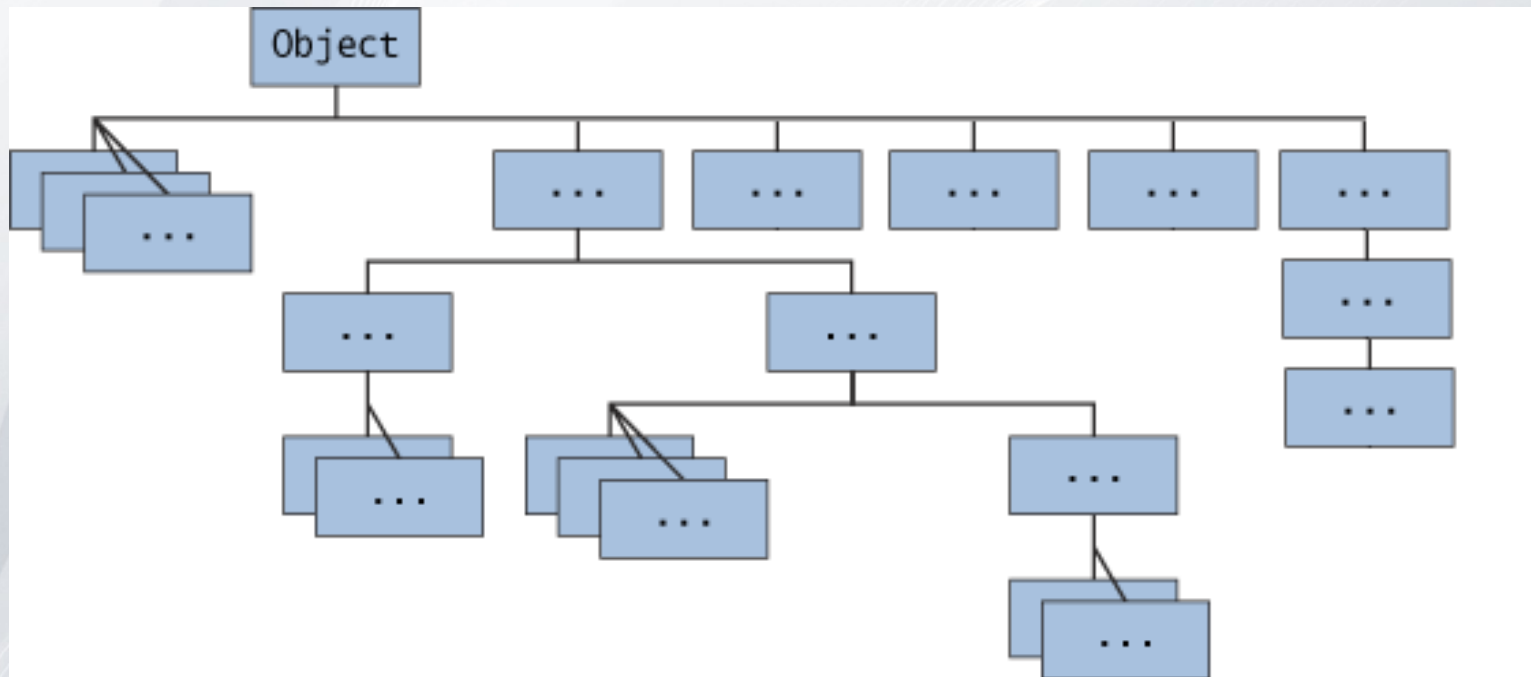
Constructor chaining

When a subclass constructor invokes a constructor of its superclass, either explicitly or implicitly, there will be a whole chain of constructors calls

```java
public class A {
    public A() {
        System.out.println("Calling constructor of class A");
    }
}
    public class B extends A{
        public B() {
            System.out.println("Calling constructor of class B");
        }
    }
        public class C extends B{
            public C() {
                System.out.println("Calling constructor of class C");
            }
        }
```

```
Console ✕
<terminated> ChainTest [Java Application] C:\Program Files\Java
Calling constructor of class A
Calling constructor of class B
Calling constructor of class C
```

- java.lang.Object is parent of all classes in Java

- It defines behavior common to all classes

- When you write class which don't extends other class, it implicitly derive from Object

- Create a new class. Then instantiate it and use Ctrl + Space in eclipse to view Object's methods

- boolean equals(Object obj)

  Indicates whether some other object is "equal to" this one.

- String toString()

  Returns a string representation of the object.

  (later we'll write some equals and toString methods)

Let's create Car hierarchy to demonstrate all described in the previous slides

We can use classes Car and Person from lesson 06

Cars example

Car
- model
- maxSpeed
- currentSpeed
- currentGear
- numberOfDoors
- idNumber
- isSportCar
- owner
- changeOwner(Person newOwner)
- startEngine()
- changeGear(int gear)
- Accelerate(double speed)

SportCar
- isCabriolet
- price
- switchTurbo()

Jeep
- ...
- switch4x4()
- driveOffRoad()

Bus
- numberOfSeats
- placeForLuggage
- ...

Ferrari
- ...

...
- ...

...
- ...

...
- ...
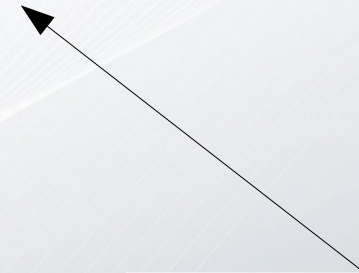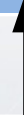
- Subclasses classes can use functionality of the parent class through the keyword *super*

- Super can be used for fields, methods or constructors of the super class

- You can declare a field in the subclass with the same name as the one in the superclass (not recommended)

Try setting isSportCar to true in class SportCar.

What happen if class isSportCar declare its own field isSportCar?

- A subclass can predefine the methods of its parent

- This is called overriding

- To override a method:

  - Use its name as it's in the parent class

  - Use the same number and order of method arguments

  - Use the same return type(or subtype of the type returned by the overridden method. This is called a covariant return type)

Method overriding example

Birds can fly. The eagle and the sparrow are birds and can fly. Although the eagle can fly highly and fast.

```java
public class Eagle extends Bird{

    public void fly(){
        System.out.println("Flying highly like an eagle");
    }
}
```

```java
public class Sparrow extends Bird{

    public void fly(){
        System.out.println("Flying like a sparrow");
    }
}
```

- Methods declared public in a superclass also must be public in all subclasses.

- Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.

- Methods declared without access control (no modifier was used) can't be declared as private in subclasses.

- Methods declared private are not inherited at all, so there is no rule for them.

- Overriding is used for Polymorphism
  (we'll talk about this in the next lesson)

- You can invoke the parent method when overriding it

- Keyword *super* is used

  Try to override method startEngine in class SportCar.

  In the body of the method first invoke startEngine of the parent, and then switch on the turbo.

```java
public class Car {
    ...
    public void startEngine() {
        System.out.println("Starting engine...");
    }
    ...
}
```

> Overriding method startEngine()

```java
public class SportCar extends Car {
    ...
    public void startEngine() {
        super.startEngine();
        switchTurbo(true);
    }
}
```

> Invoke startEngine() of the parent class

Try to override equals(Object obj) and toString() too.

- What is inheritance and how to use it?

- 'Is a' and 'has a' relationship

- Access modifiers in context of inheritance

- Invoking constructors of the superclass and constructor chaining

- Java hierarchy

- How to use *super* keyword?

- What is method overriding?

- What is the constraints when overriding method?