Inheritance
(Chapter 8 of Schilit)

Object Oriented Programming BS (AI/MMG) II

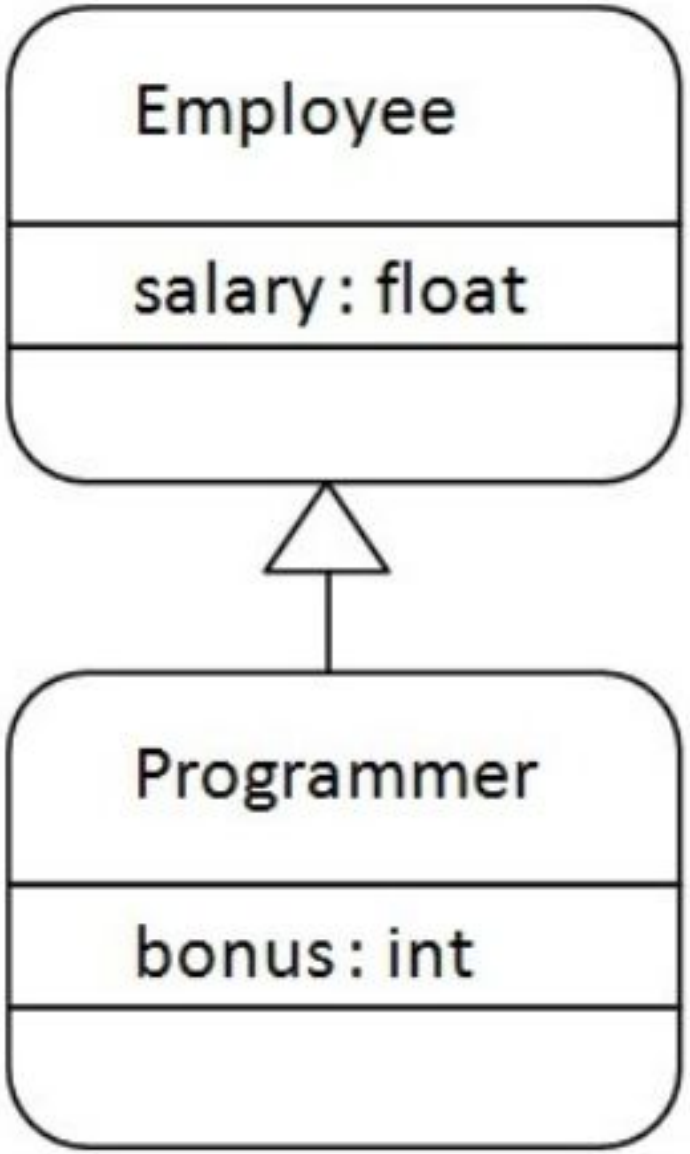
By

Abdul Ghafoor

Inheritance

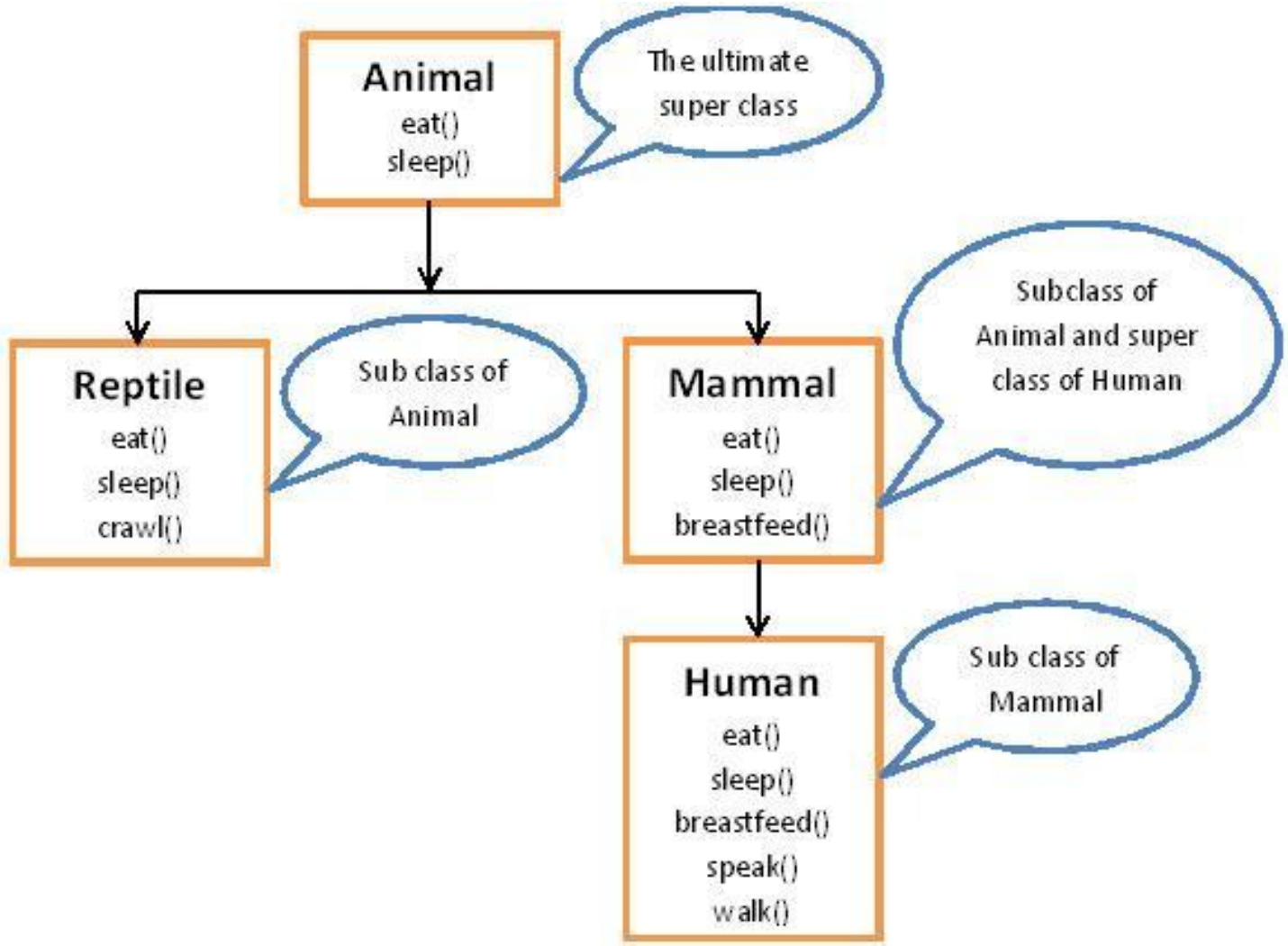
- Define a general class (Super Class)
- Define some special classes (Sub Classes):
  - Contain existing properties of general class, by inheriting from it
  - Also add some unique properties

Example



Example: superclass subclass

Inheritance



Extends keyword

```
class subclass-name extends superclass-name {
  // body of class
}
```

- Used to provide inheritance in java
- A extends B; means A is a subclass(specialized version) of B and B is super class
- Extends functionality

```
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Test it Now

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

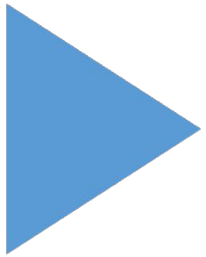
Output:

Demo

Inheritance

A subclass can not access private members of a superclass

A private member remains private to that class

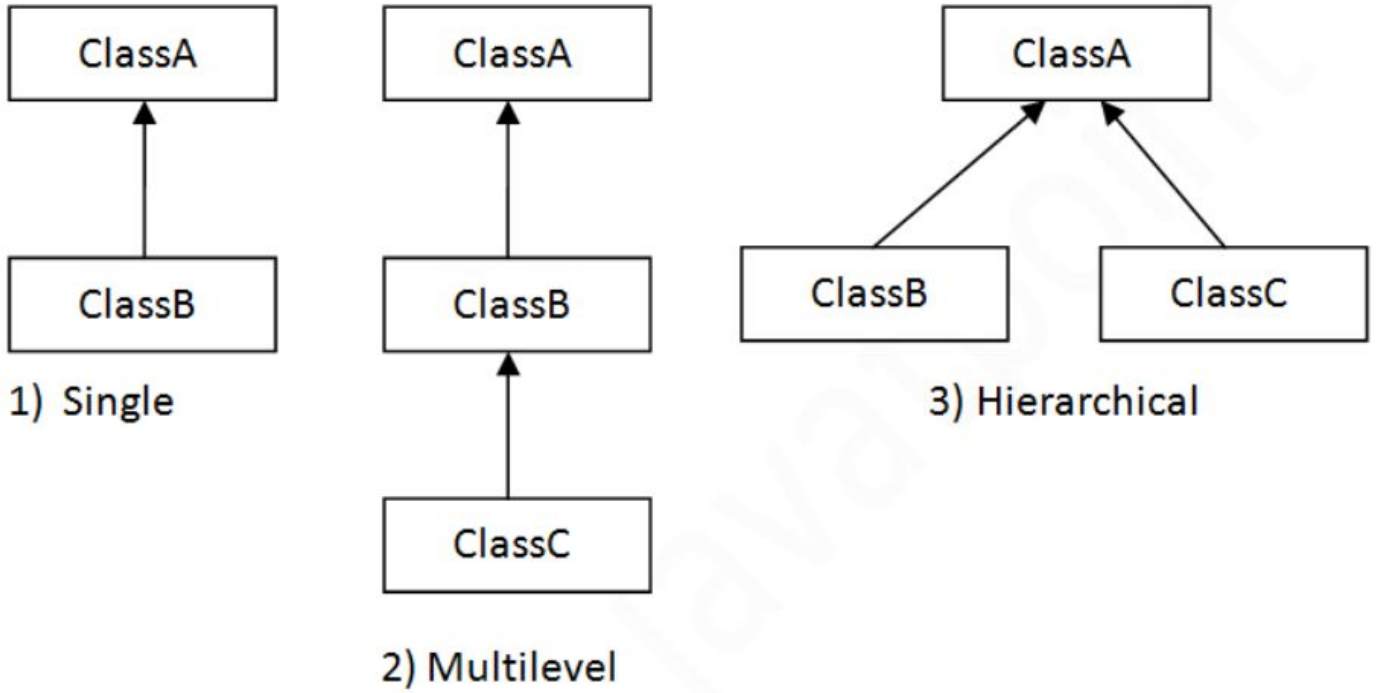


Demo

Demo

- Extending Box Class to create:
  - BoxWeightClass
  - BoxColorClass

Types


ClassA
ClassB
1) Single
ClassA
ClassB
ClassC
2) Multilevel
ClassA
ClassB
ClassC
3) Hierarchical

Inheritance

In java multiple inheritance is not supported

Whereas Multilevel Inheritance is supported

Inheritance

- Member access
- Example: private: no access

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Inheritance

- A reference variable of superclass can be assigned:
  - The reference variable of any child class that is derived from that superclass
  - Demo

Inheritance

- Any variable that is pointing to child class object:
  - When it is assigned to a reference variable which is of parent type
  - When a subclass reference is assigned to a superclass variable
  - Child c=new Child();
  - Parent p=new Parent();
  - p=c
  - You will have access to only those parts which are defined by parent class:
    - Because parent has no knowledge of what a child adds to its implementation

Super keyword

When a child wants to make reference to its immediate parent:

- It can do so by **super** keyword

Two Forms:

- First to call superclass or base class constructor
- Second to access a member of superclass

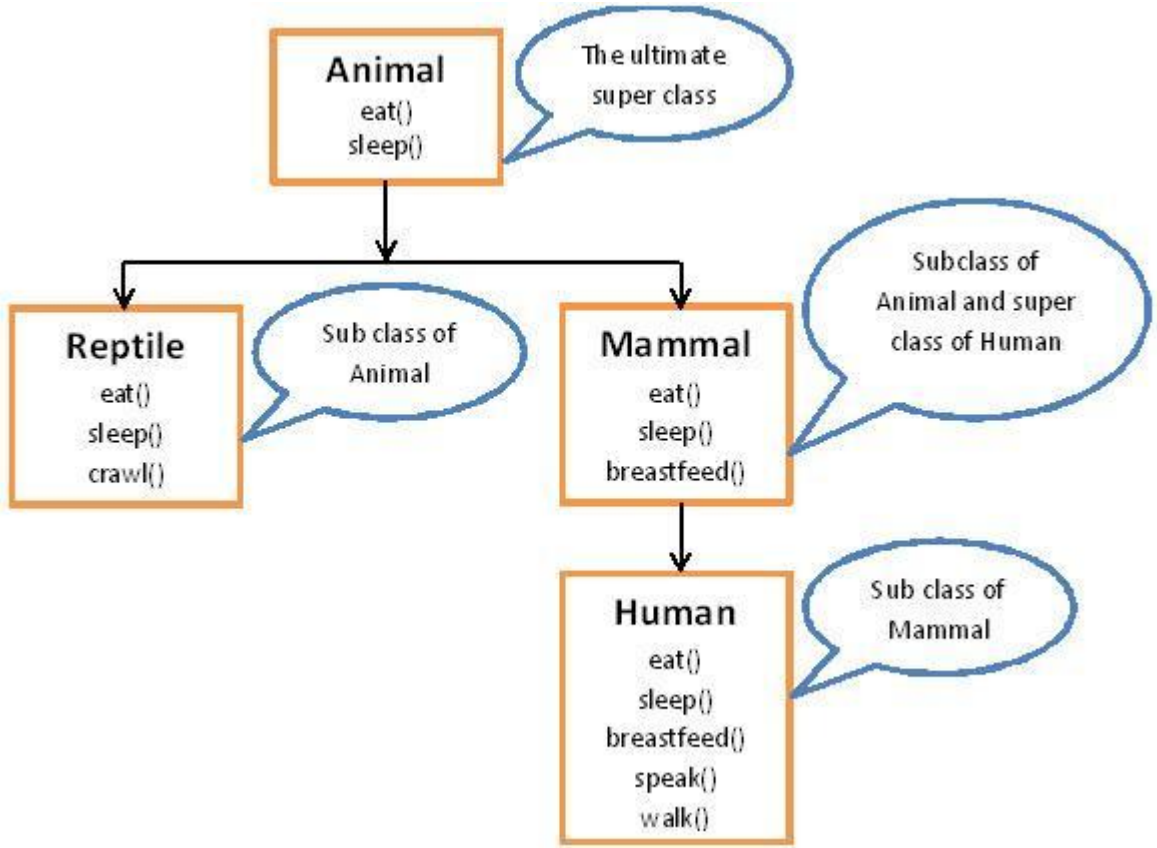
Using Super to call Base class Constructor, Demo

Using super with Data Members

- Prevents Name Hiding:
  - Parent class and child class have same data member name
  - Demo

Inheritance

- Multilevel hierarchy



Lets create a Multi-Level Hierarichy

Demo

When constructors are executed?

Executed in order of derivation:

- Inside each constructor immediate super() is called automatically

Demo

Method Overriding

- A method in subclass has:
  - Same name
  - Type
  - Number of Parameters
  - It overrides the parent class method

- When overridden method called by subclass:
  - Parent method is hidden

Demo

Accessing Parent class show method in subclass show method

```
class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

  void show() {
    super.show(); // this calls A's show()
    System.out.println("k: " + k);
  }
}
```

Overloading vs Overriding

- Changing the number of type of parameters will cause the method to be overloaded
- For Overriding we need Inheritance, while Overloading can still occur in one class

Demo

- Overloading parent method

Dynamic Method Dispatch

- Calls to methods are resolved at run-time:
  - Way to Achieve Run-Time Polymorphism
- Calls are resolved by examining:
  - Which object made the call to the mthod

Demo

Why Overriding?

- Provides us Run-Time Polymorphism

Demo

- A Practical Example of Overriding (Run-Time Polymorphism)

Using abstract classes

Use of final keyword with Inheritance

- To prevent method overriding
- To prevent class inheritance

Using final to prevent inheritance

```
final class A {
  //...
}

// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
  //...
}
```

Using final to prevent Overriding

```
class A {
  final void meth() {
    System.out.println("This is a final method.");
  }
}

class B extends A {
  void meth() { // ERROR! Can't override.
    System.out.println("Illegal!");
  }
}
```

Object Class

- A special class
- Which is parent of all the classes in java, and all other classes are subclasses of this class

Object Class Methods

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled. (Deprecated by JDK 9.)
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.