Java

Agenda

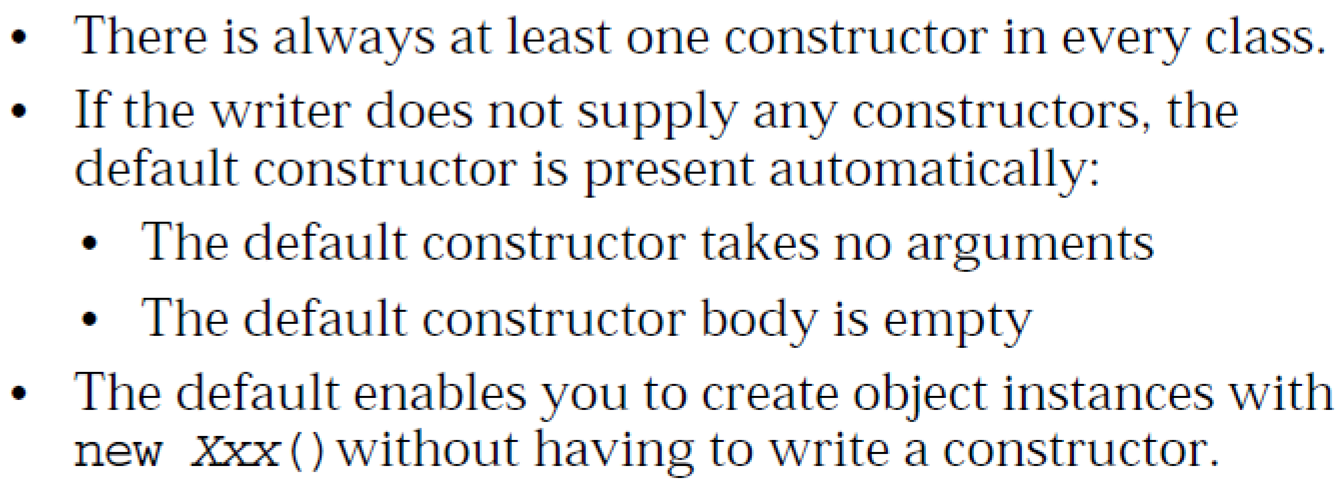
- Constructor Overloading
- Inheritance
- Overriding

Constructor

The Constructor

- A constructor initializes an object when it is created.
- Has the same name as its class
- Have no explicit return type.
- Used to give initial values to the instance variables defined by the class
- All classes have a default constructor that initializes all member variables to zero.
- When own constructor is defined by the programmer, the default constructor is no longer used.

Default Constructor



- There is always at least one constructor in every class.
- If the writer does not supply any constructors, the default constructor is present automatically:
  - The default constructor takes no arguments
  - The default constructor body is empty
- The default enables you to create object instances with `new` *`Xxx`*`()` without having to write a constructor.

Constructor Overloading

- As with methods, constructors can be overloaded. An example is:

```
public Employee(String name, double salary, Date DoB)
public Employee(String name, double salary)
public Employee(String name, Date DoB)
```

- Argument lists *must* differ.
- You can use the `this` reference at the first line of a constructor to call another constructor.

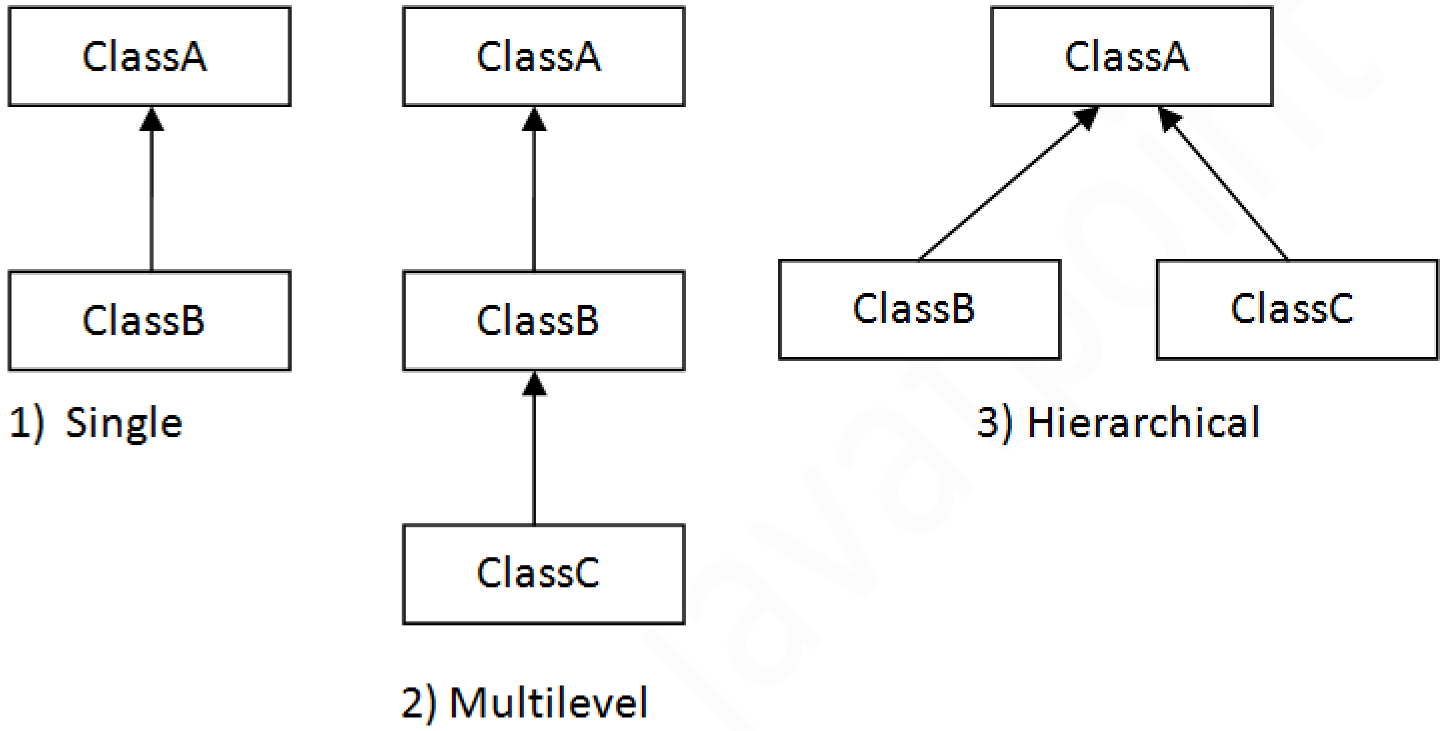
```java
public class Employee {
  private static final double BASE_SALARY = 15000.00;
  private String name;
  private double salary;
  private Date   birthDate;

  public Employee(String name, double salary, Date DoB) {
    this.name = name;
    this.salary = salary;
    this.birthDate = DoB;
  }
  public Employee(String name, double salary) {
    this(name, salary, null);
  }
  public Employee(String name, Date DoB) {
    this(name, BASE_SALARY, DoB);
  }
  // more Employee code...
}
```

INHERITANCE

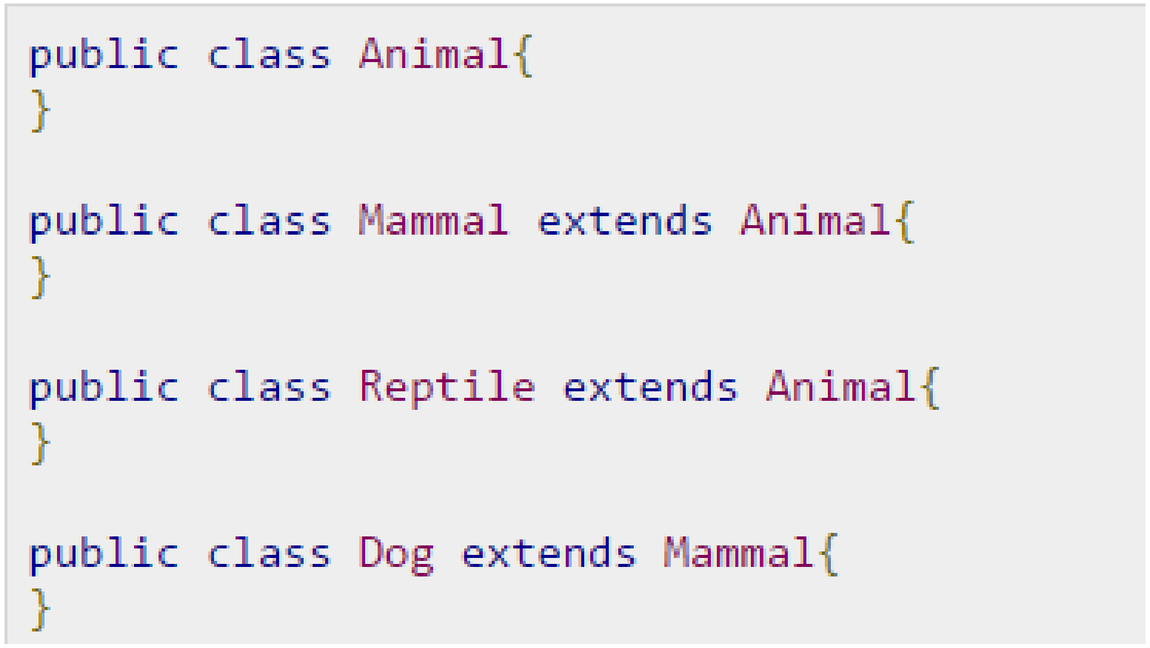
INHERITANCE

- extends ,implements - keyword
- Single , Multilevel, Hierarchical – supported

Types Of Inheritance



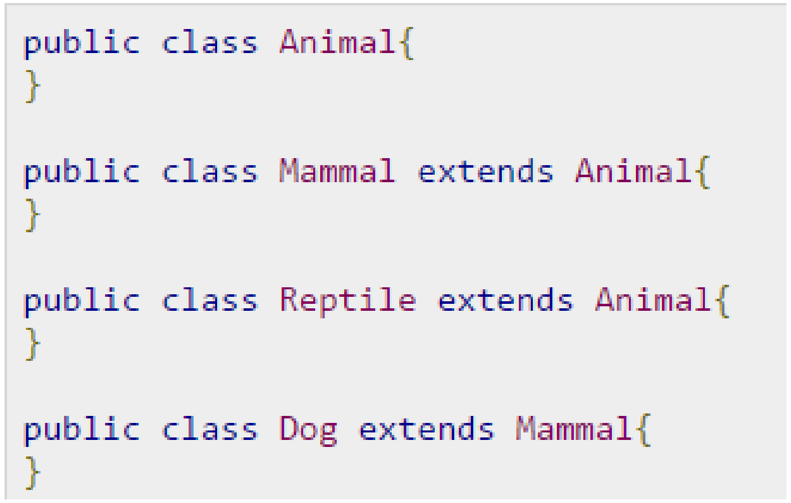
Extends Keyword



```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

IS-A Relationship

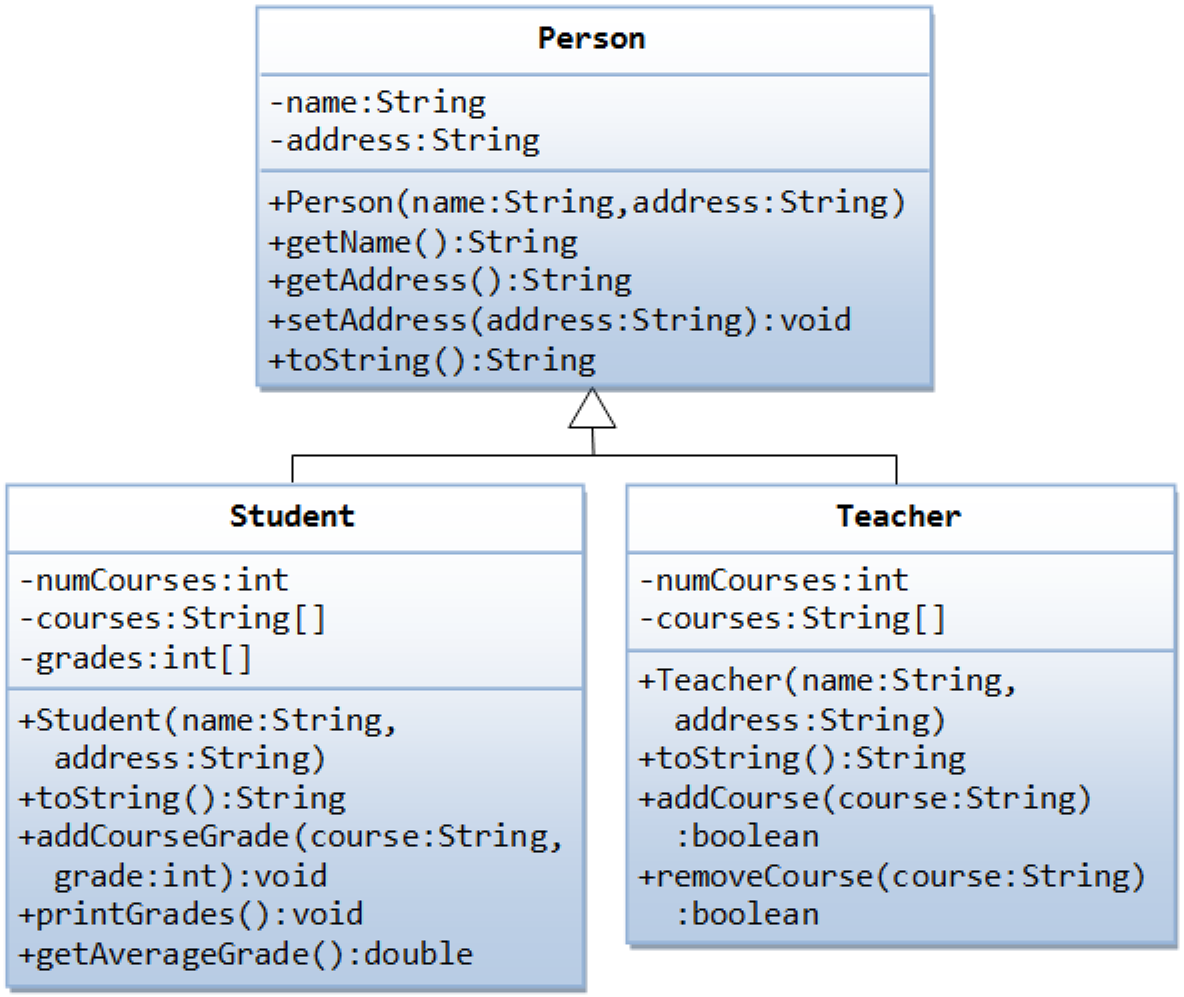
- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.



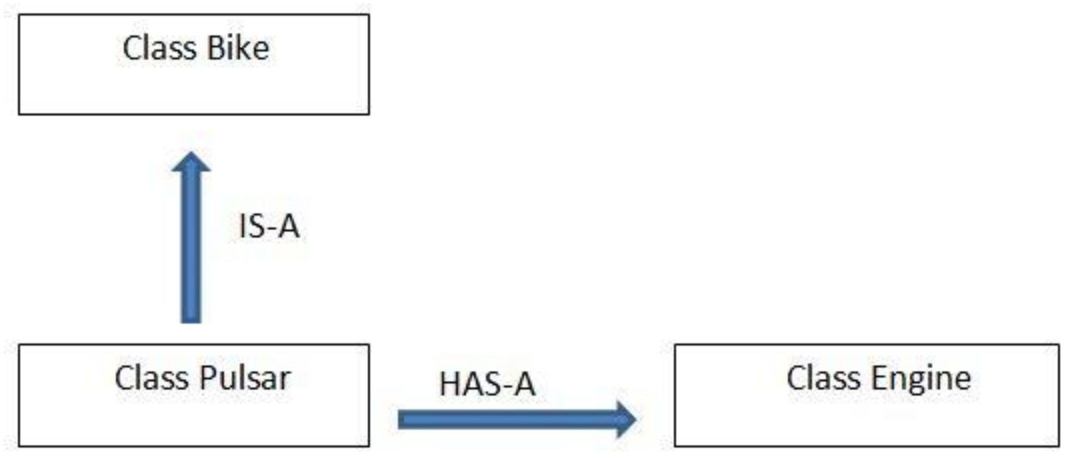
```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

IS-A RELATIONSHIP

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence : **Dog IS-A Animal as well**


Person
-name:String
-address:String
+Person(name:String,address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
Student
-numCourses:int
-courses:String[]
-grades:int[]
+Student(name:String, address:String)
+toString():String
+addCourseGrade(course:String, grade:int):void
+printGrades():void
+getAverageGrade():double
Teacher
-numCourses:int
-courses:String[]
+Teacher(name:String, address:String)
+toString():String
+addCourse(course:String) :boolean
+removeCourse(course:String) :boolean

Has-A relationship



Overriding

Overriding

- To override the functionality of an existing method in parent class.

```
class Animal{

   public void move(){
      System.out.println("Animals can
     move");
   }
}

class Dog extends Animal{

   public void move(){
      System.out.println("Dogs can walk
     and run");
   }
}
```

```
public class Test{

   public static void main(String args[]){
      Animal a = new Animal(); // Animal
     reference and object
      Animal b = new Dog(); // Animal
     reference but Dog object

      a.move();// runs the method in
     Animal class

      b.move();//Runs the method in Dog
     class
   }
}
```

Super Keyword

```
class Animal{

  public void move(){
    System.out.println("Animals can move");
  }
}

class Dog extends Animal{

  public void move(){
    super.move(); // invokes the super class method
    System.out.println("Dogs can walk and run");
  }
}

public class TestDog{

  public static void main(String args[]){

    Animal b = new Dog(); // Animal reference but Dog object
    b.move(); //Runs the method in Dog class

  }
}
```

Rules for method overriding

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level. For example: if the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.