Lecture – 11 - 15

Inheritance

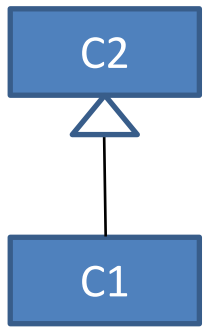
Introduction

- Inheritance is the process by which a new class is created from another class
  - The new class is called a ***derived class***
  - The original class is called the ***base class***
- A derived class automatically has
  - <u>All the instance variables and methods</u> that the base class has
  - <u>And additional methods and/or instance variables as well</u>
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- Advantage is the code reusability

Introduction

- Syntax

```
class Subclass-name extends Superclass-name {
  //methods and fields
}
```



Superclass/Parent Class or Base Class

A class C1 extended from another class C2 is called a subclass/Child class/Extended class, or a Derived class.

- The **extends keyword** indicates that you are making a new class that derives from an existing class.
- The meaning of "extends" is to increase the functionality.

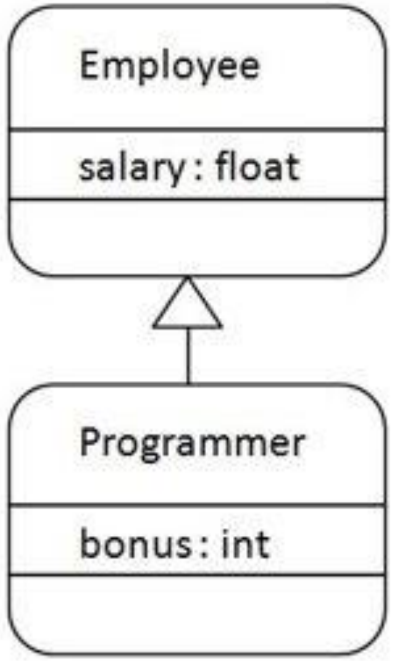
Examples

Superclass	Subclasses
`Student`	`GraduateStudent`, `UndergraduateStudent`
`Shape`	`Circle`, `Triangle`, `Rectangle`
`Loan`	`CarLoan`, `HomeImprovementLoan`, `MortgageLoan`
`Employee`	`Faculty`, `Staff`
`BankAccount`	`CheckingAccount`, `SavingsAccount`

Example

```
class Employee{
 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);
 }
}
```

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

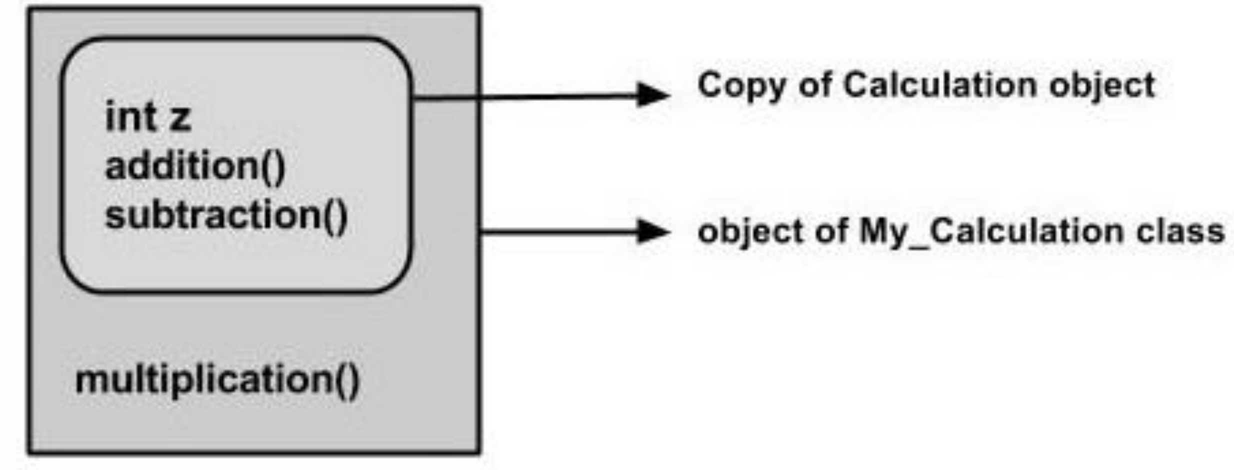


Relationship between two classes
Programmer IS-A Employee

```
public class Calculation {
  int z;
  public void addition(int x, int y) {
    z = x + y;
    System.out.println("The sum of the given numbers:"+z);
  }
  public void Subtraction(int x, int y) {
    z = x - y;
    System.out.println("The difference between the given
numbers:"+z);
  }
}
```

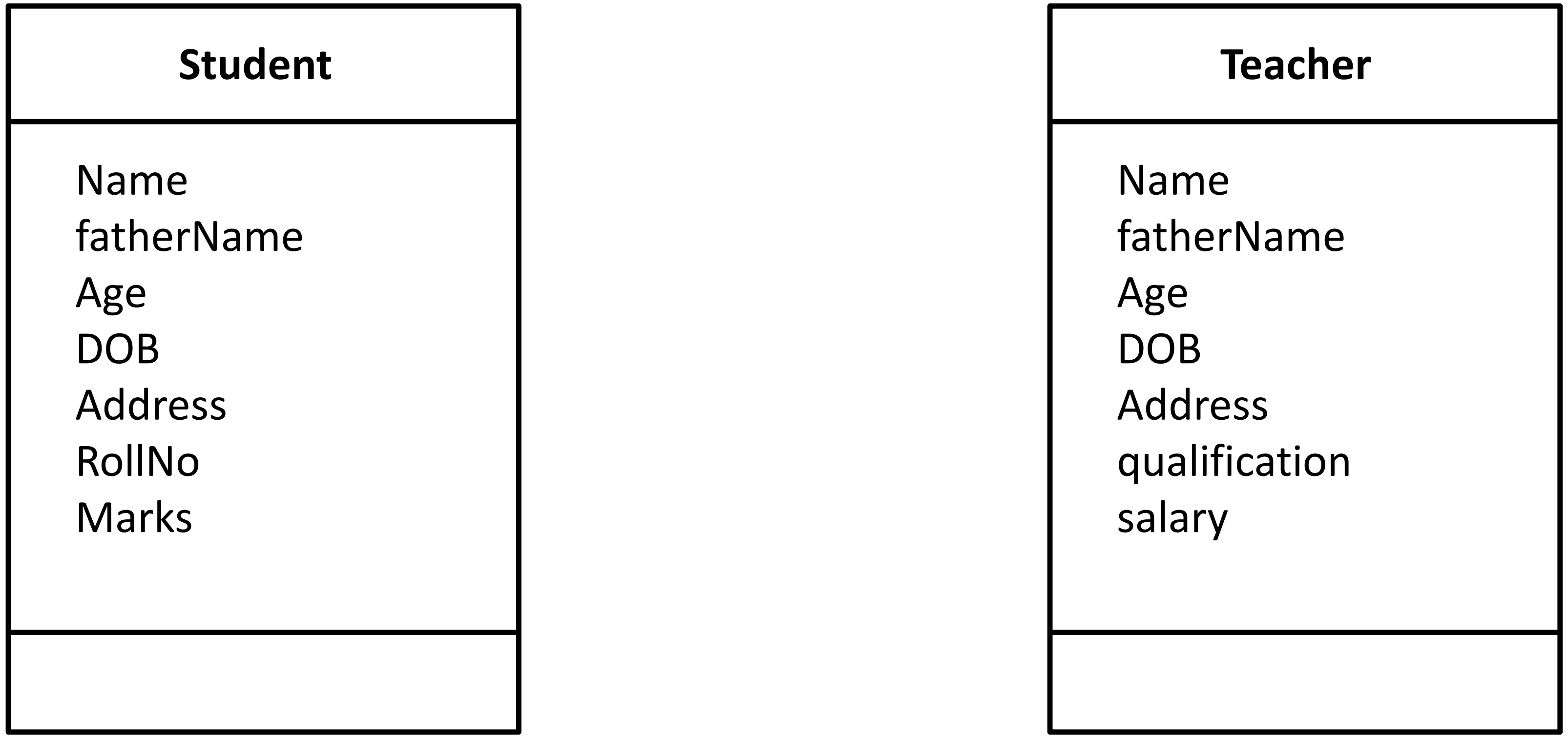
```
public class My_Calculation extends Calculation {
  public void multiplication(int x, int y) {
    z = x * y;
    System.out.println("The product of the given numbers:"+z);
  }
}
```

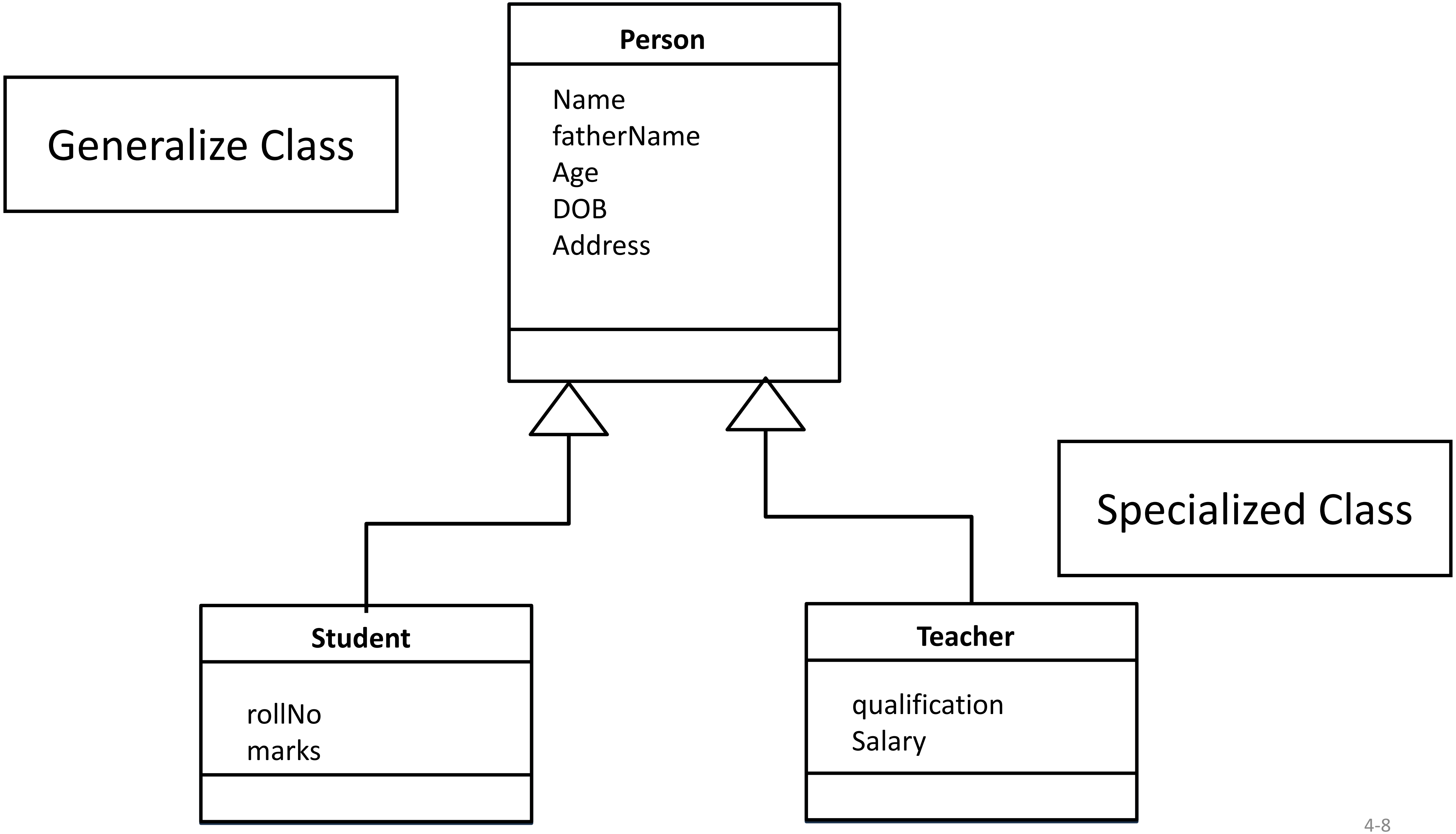
```
   public class Demo{
            public static void main(String args[]) {
      int a = 20, b = 10;
      My_Calculation demo = new My_Calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
    }
   }
```



Introduction

- Generalization and Specialization
  - Inheritance enable you to define a general class (i.e., a superclass) and later extend it to more specialized classes (i.e., subclasses).

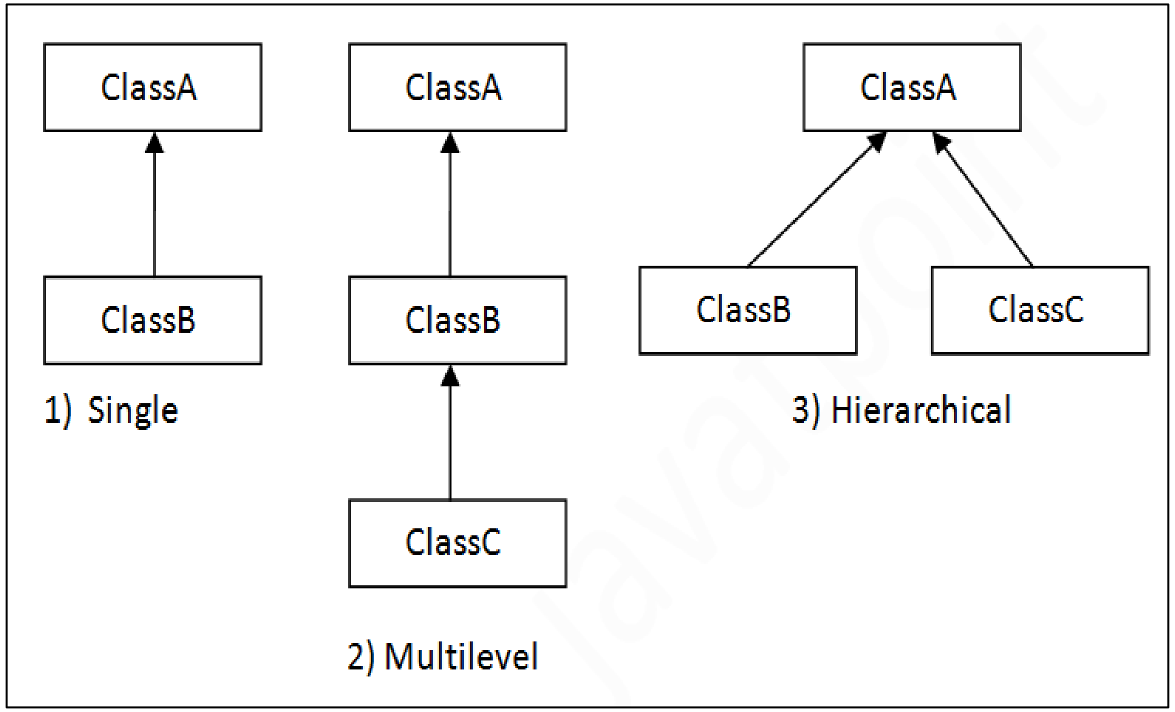


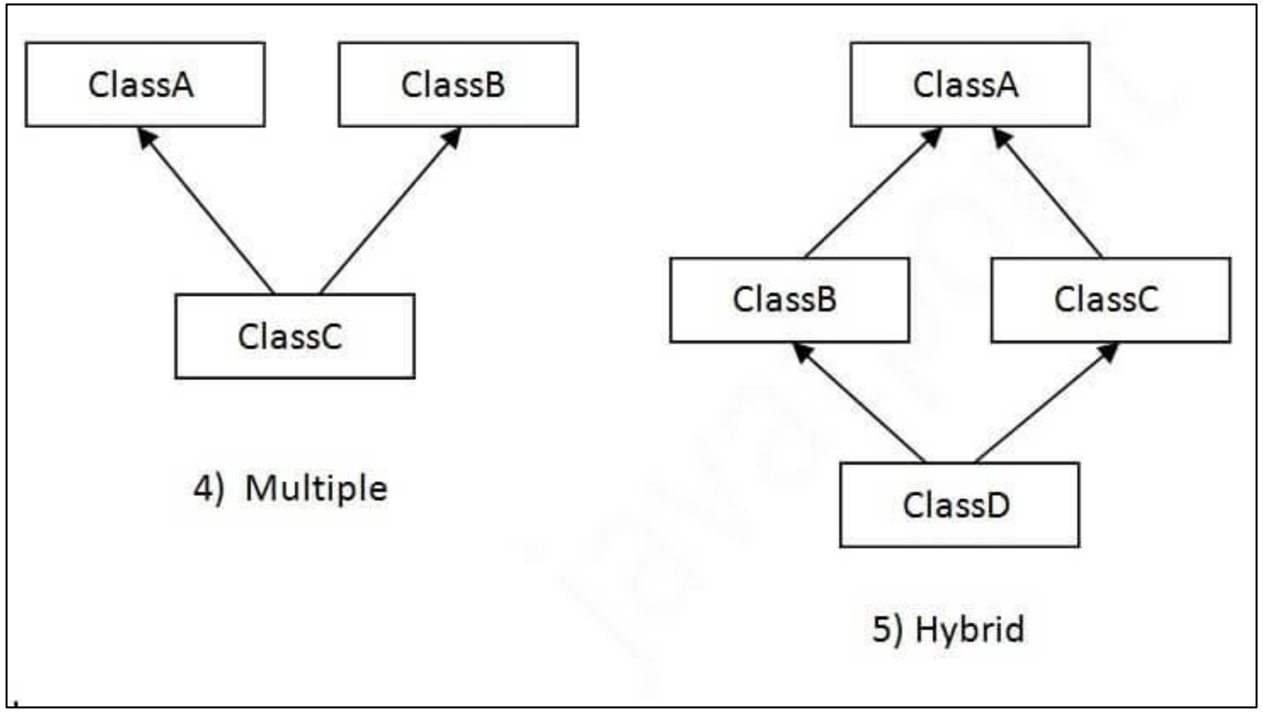

Generalize Class
Person
Name
fatherName
Age
DOB
Address
Student
rollNo
marks
Teacher
qualification
Salary
Specialized Class

What is Inherited in Subclass

- All non-private **fields**(Instance Variables) and **Methods** (declared with any access modifier `public`, `protected`, or `default`) of the superclass are inherited by the subclass
  - Subclass can override methods or hide fields with its own implementation
- Constructors are not inherited by subclasses
- A subclass can call a constructor of the superclass using the **super()** keyword

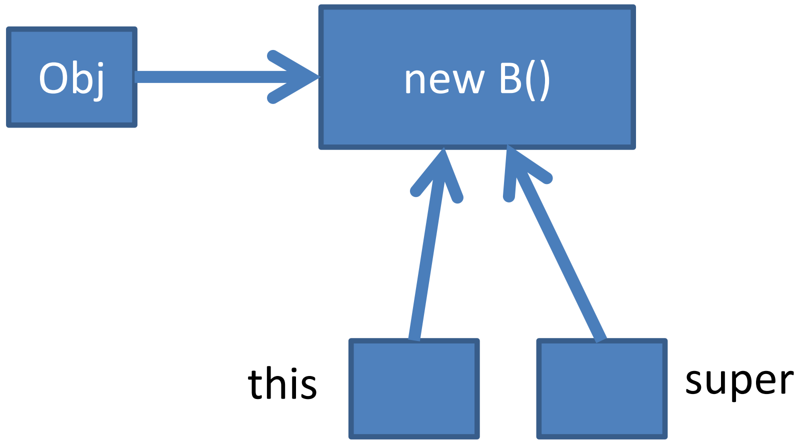
Types of inheritance in java





Super keyword

- In inheritance, subclass object, when call an instance member function of subclass only
  - The function contains two reference variables **this** and **super** referring to current object (object of subclass)
- The only difference in **this** and **super** is only **type**
  - ***This*** reference variable is of subclass type
  - ***Super*** reference variable is of superclass type



The super keyword

- **super** can be used to refer immediate parent class instance variable.

```
super.variable;
```

- **super** can be used to invoke immediate parent class method.

```
super.method();
```

- It is used to **invoke the superclass** constructor from subclass.

```
super(values);
```

Constructors in Inheritance

- Constructors are not inherited in inheritance
- Subclass constructors invokes constructor of the super class
- Implicit call and explicit call to the super class constructor
  - Implicit call – java automatically place call by placing super
  - Explicit call – use super() in subclass constructor and it must be the first line in the subclass constructor.

Constructors in Inheritance

- Scenarios
  - Implicit constructors in superclass and subclass (default constructor)
  - Implicit constructor in subclass and explicit constructor in superclass (no-arg constructor)
  - Implicit constructor in superclass and explicit constructor in subclass
  - Explicit constructor in superclass and subclass
  - For parameterized constructor in superclass, subclass must use super(x) in its constructor to call superclass constructor

Find Errors

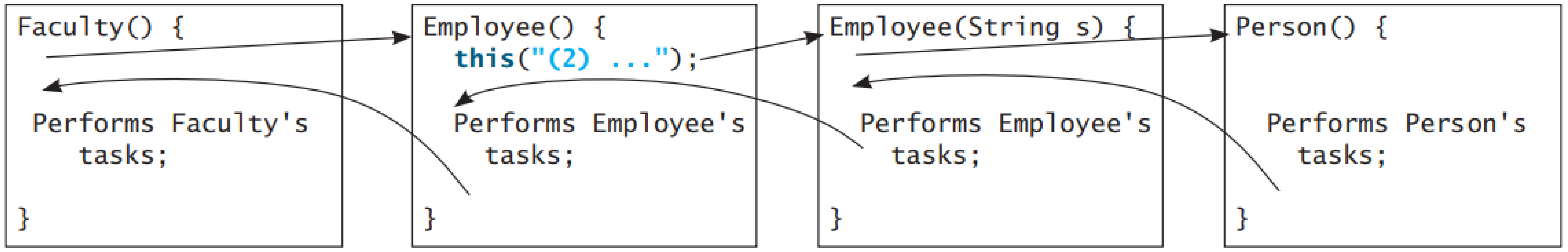
```
public class Apple extends Fruit{
}
class Fruit{
        public Fruit(String name){
                System.out.println("Fruit's constructor is invoked");
        }
}
```

Since `Apple` has no constructor, therefore no-arg constructor is defined implicitly. `Apple` is subclass of `Fruit`, `Apple` default constructor invokes `Fruit’s` no-arg constructor. However `Fruit` does not have no-arg constructor, because it has an explicit constructor defined

You should provide a no-arg constructor for every class to make the class easy to extend and to avoid errors

Constructor Chaining

- Constructor can call other constructors of the same class or superclass
- Constructor call from a constructor must be the first step(call should appear in the first line)
- Such series of invocation of constructors is known as constructor chaining



```java
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }
  public Faculty() {
    System.out.println("Faculty's no-arg constructor is invoked");
  }
}
class Employee extends Person {
  public Employee() {
     System.out.println("Employee's no-arg constructor is invoked");
  }
  public Employee(String s) {
    System.out.println(s);
  }
}
class Person {
  public Person() {
    System.out.println("Person's no-arg constructor is invoked");
  }
}
```

Overloading Methods

- If two methods of a class (whether both declared in the same class, or both inherited by a class, or one declared and one inherited) have the same name but different signatures, then the method name is said to be overloaded
- Method overloading is a way to implement polymorphism

Example

```
class A{
		public void f1(int x){
			System.out.println("A");
		}
}
class B extends A{
		public void f1(int x, int y){
			System.out.println("B");
		}
}
public class Example{
		public static void main(String[] args){
			B obj = new B();
			obj.f1(3);
			obj.f1(3,4);
		}
}
```

Overriding Methods

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- Provide the specific implementation of a method which is already provided by its superclass.
- Rules for Java Method Overriding
  - The method must have the same name as in the parent class
  - The method must have the same parameter as in the parent class.
  - There must be an IS-A relationship (inheritance).

Example

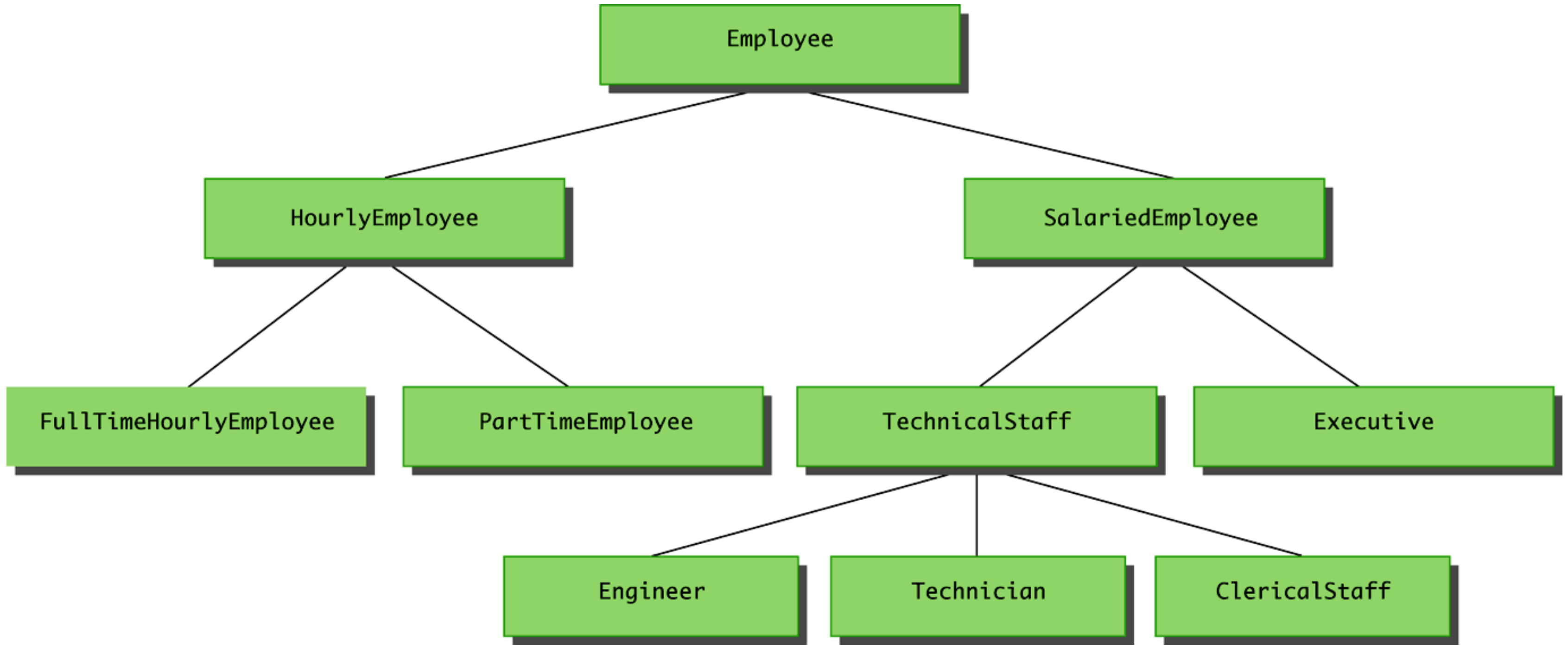
```
class A{
        public void f1(int x){
                System.out.println("Class A");
        }
}
class B extends A{
        public void f1(int x){
                System.out.println("Class B");
        }
}
public class ExampleOverRiding{
        public static void main(String[] args){
                B obj = new B();
                obj.f1(8);
        }
}
```

Overriding and Access-Modifiers

- The access modifier for an overriding method can allow more, but not less, access than the overridden method
- For example
  - A protected instance method in superclass can be made public, but not private, in the subclass.
  - Doing so will generate compile time error

Inheritance hierarchy for Employee

- Each arrow in the hierarchy represents an *is-a* relationship
- Not every class relationship is an inheritance relationship



Employee - Example

- Class **`Employee`** defines the instance variables **`name`** and **`hireDate`** in its class definition
- Class **`HourlyEmployee`** also has these instance variables, but they are not specified in its class definition
- Class **`HourlyEmployee`** has additional instance variables **`wageRate`** and **`hours`** that are specified in its class definition
- The class **`HourlyEmployee`** inherits the methods **`getName`**, **`getHireDate`**, **`setName`**, and **`setHireDate`** from the class **`Employee`**