City University

SE 409, Advanced Enterprise Java

Lecture 3 & 4

Supta Richard Philip

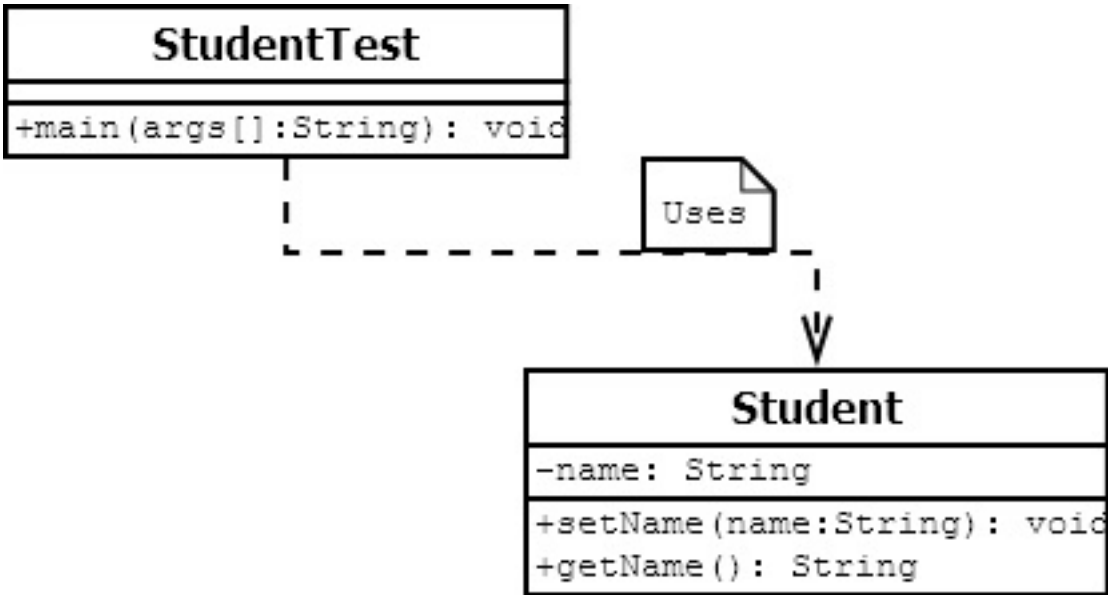
supta.philip@gmail.com

The main goal of OOP is to bind data and code(methods) togather.

OOP provides few concepts

1. Encapsulation

- Encapsulation in Java is a process of wrapping code and data together into a single unit.
- create a fully encapsulated class in Java by making private all the data members of the class.
- use setter and getter methods to set and get the data in it.
- The Java Bean class or POJO class is the example of a fully encapsulated class.
- Class Diagram of Student



Student.java

```
package org.cityU.Encapsulation;
public class Student {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

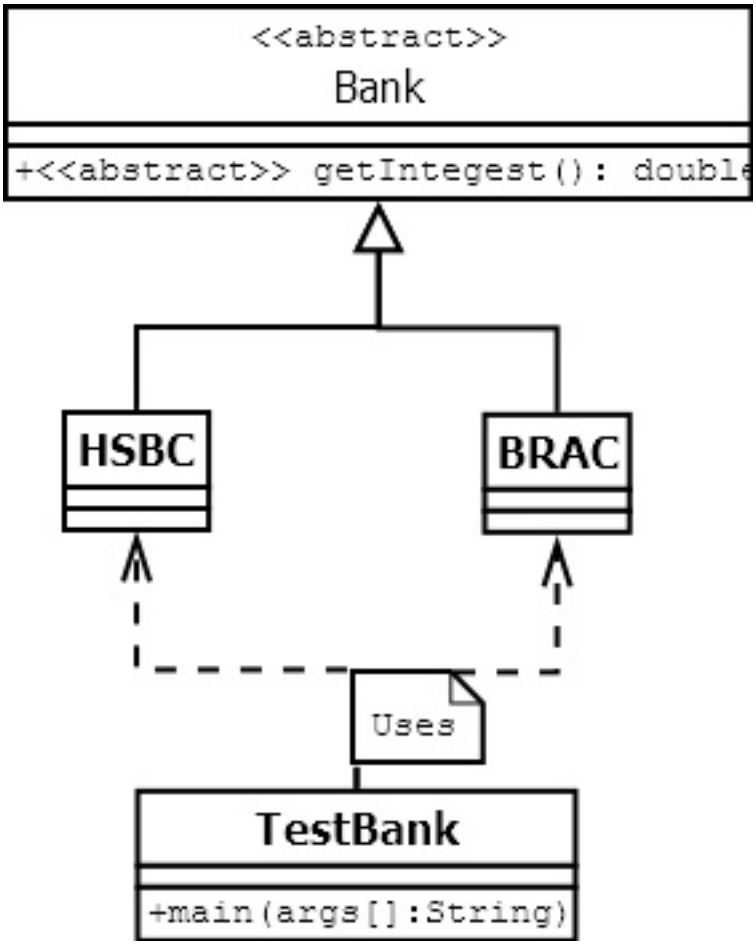
StudentTest.java

```
package org.cityU.Encapsulation;

public class StudentTest {
```

```
    public static void main(String[] args) {

        Student s = new Student();
        s.setName("Richard");

        System.out.println(s.getName());
    }
}
```

2. Abstraction

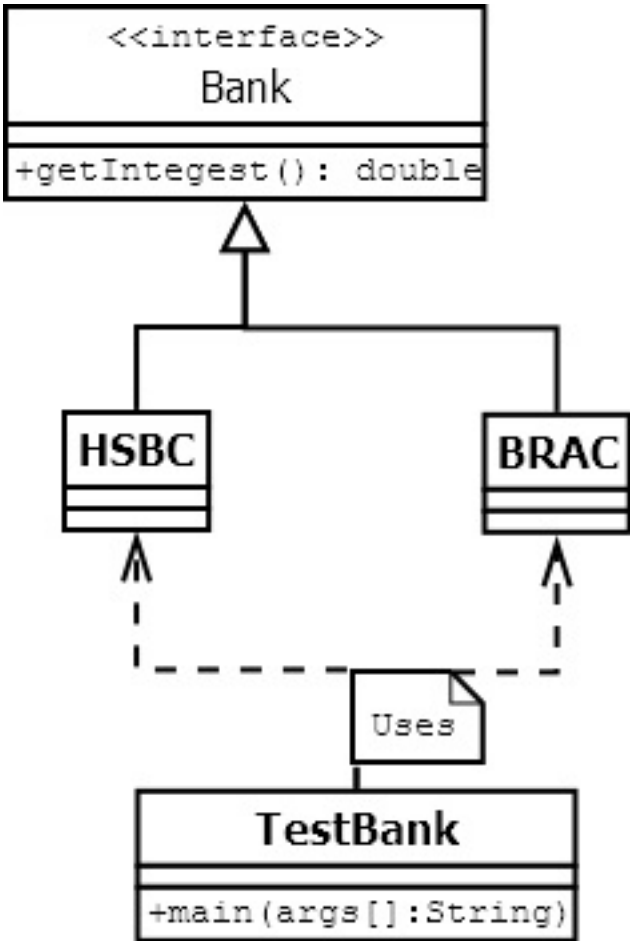
- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- There are two ways to achieve abstraction in java
    - Abstract class (0 to 100%)
    - Interface (100%)




```
abstract class Bank {
    abstract int getRateOfInterest();
}
class HSBC extends Bank {
    int getRateOfInterest() {
        return 7;
    }
}
class BRAC extends Bank {
    int getRateOfInterest() {
        return 8;
    }
}
```

```
class TestBank {
    public static void main(String args[]) {
        Bank b;
        b = new HSBC();
        System.out.println("Rate of Interest is: " + b.get
RateOfInterest() + " %");
        b = new BRAC();
        System.out.println("Rate of Interest is: " + b.get
RateOfInterest() + " %");
    }
}
```

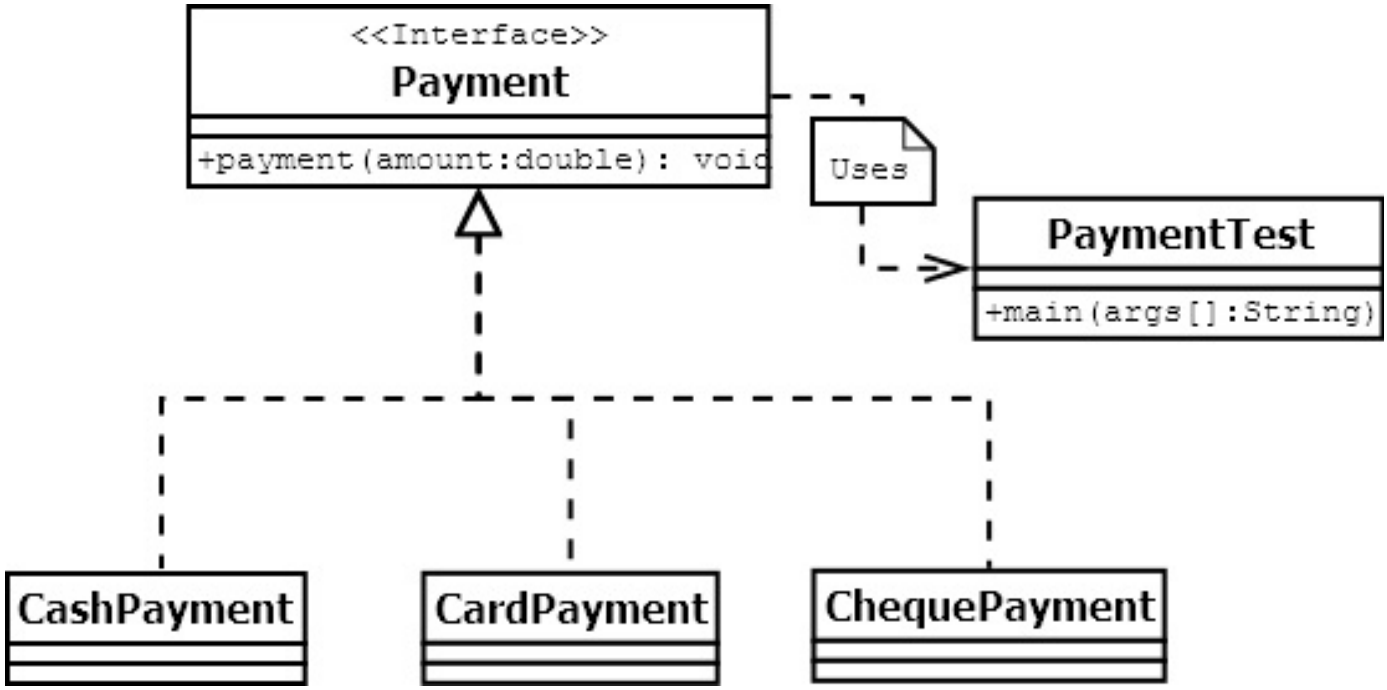
Same examples you can solve with interface.



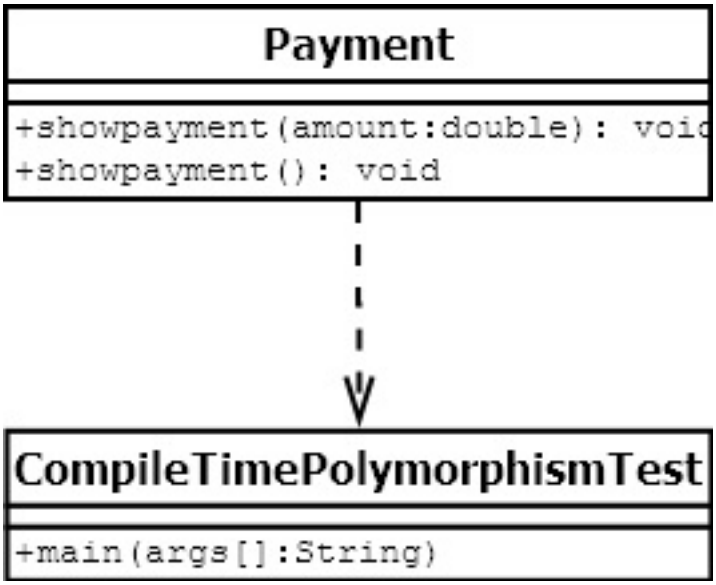
3. Polymorphism

- Polymorphism in Java is a concept by which we can perform a single action in different ways.
- There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.

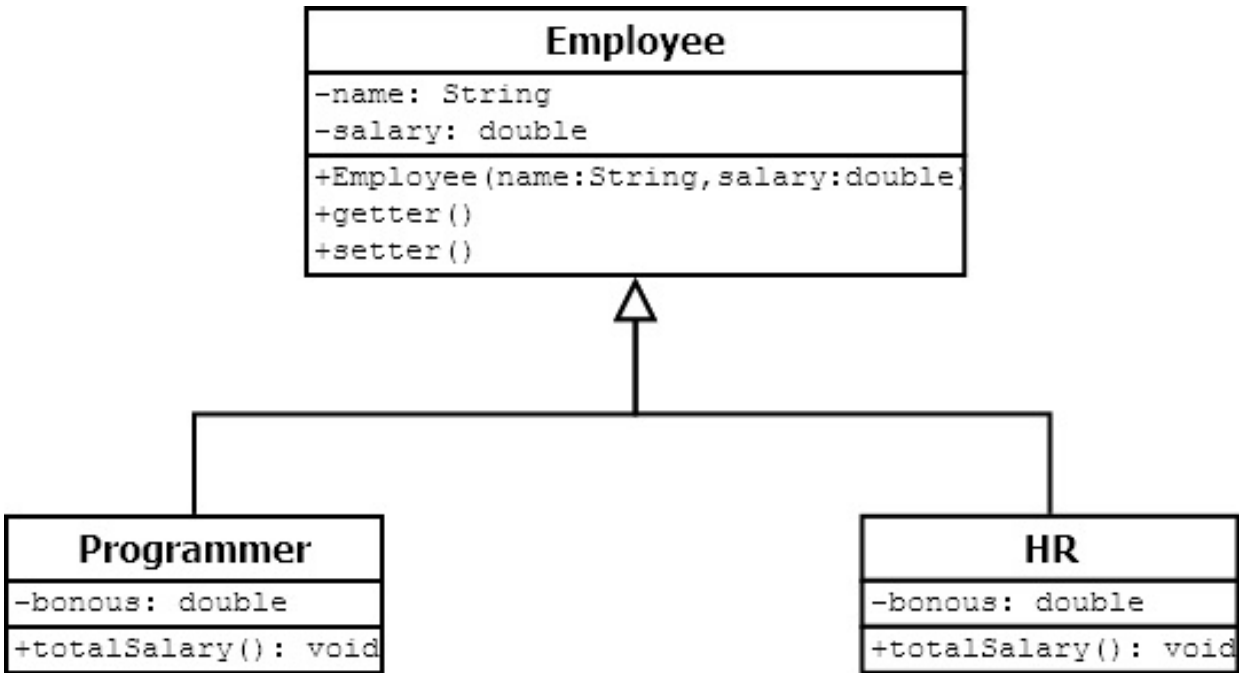
Runtime polymorphism using interface and method overriding



Compiletime polymorphism using method overloading



4. Inheritance

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- create new classes that are built upon existing classes.
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.
- 


```
public class Employee {
```

```
    private String name;
    private double salary;
    public Employee(String name, double salary) {
        super();
        this.name = name;
        this.salary = salary;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

```
public class Programmer extends Employee{
    private double bonous;
    public Programmer(String name, double salary) {
        super(name, salary);
        this.bonous=bonous;
    }
```

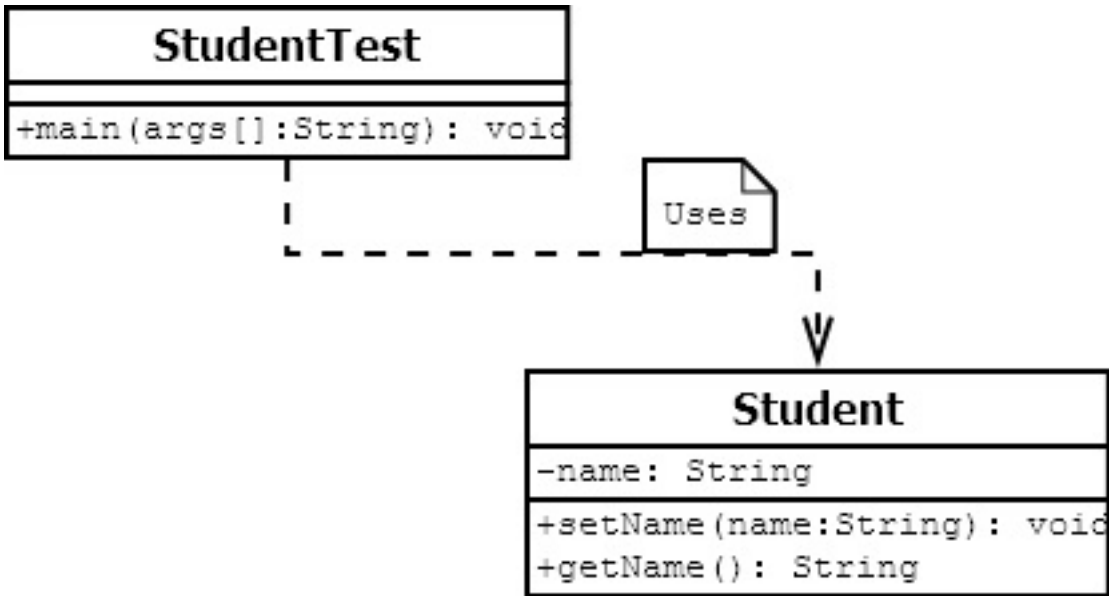
```
    public double getBonous() {
        return bonous;
    }
    public void setBonous(double bonous) {
        this.bonous = bonous;
    }
}
```

> *Watch the video tutorials*
>
> 

> *Apart from these, there are some other concepts which are used in Object-Oriented design:*

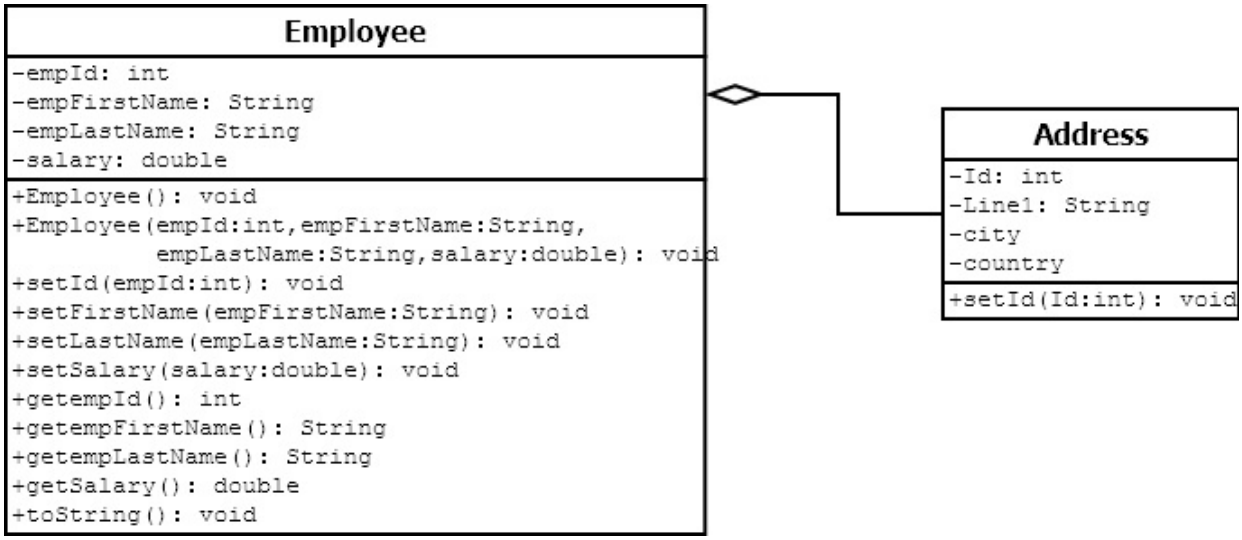
5. Association

- Association refers to the relationship between multiple objects.



5.1. Aggregation

- Aggregation is a weak association.
- An association is said to be aggregation if both Objects can exist independently.



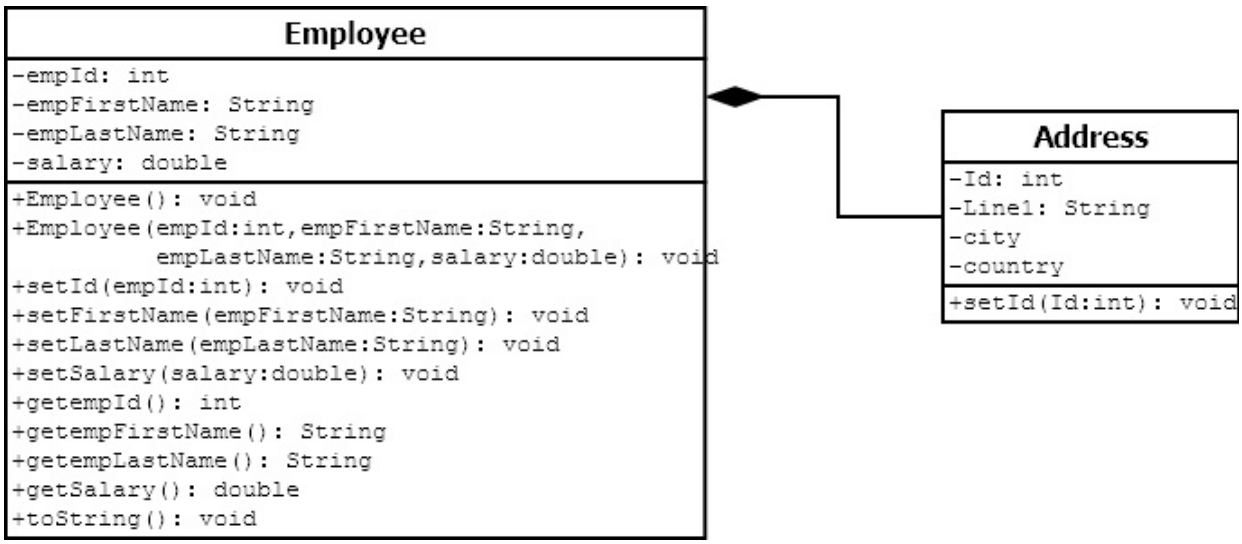

```
//Address.java
public class Address {
    private int id;
    private String line;
    private String city;
    private String country;
```

```
    //getter, setter, toString
}

//Employee.java
public class Employee {
    private long empId;
    private String empName;
    private Address address;// Aggregation
    private double salary;
    //getter, setter, toString
    }

//AggregationTest.java
public class AggregationTest {
public static void main(String[] args) {
    //Address object
    Address home = new Address();
    home.setId(1);
    home.setLine("Honda Goli");
    home.setCity("Dhaka");
    home.setCountry("Bangladesh");
    //Employee Object
    Employee e = new Employee();
    e.setEmpId(100);
    e.setEmpName("Richard");
    e.setSalary(10000);
    e.setAddress(home);
    System.out.println(e);
```

```
    //assign employee to null
    e = null;
    //Still exist Adsress in memory. This is called soft r
elationship.
    System.out.println(home);
}
}
```

5.2. Composition

- The composition is the strong type of association.
- An association is said to composition if an Object owns another object and another object cannot exist without the owner object.

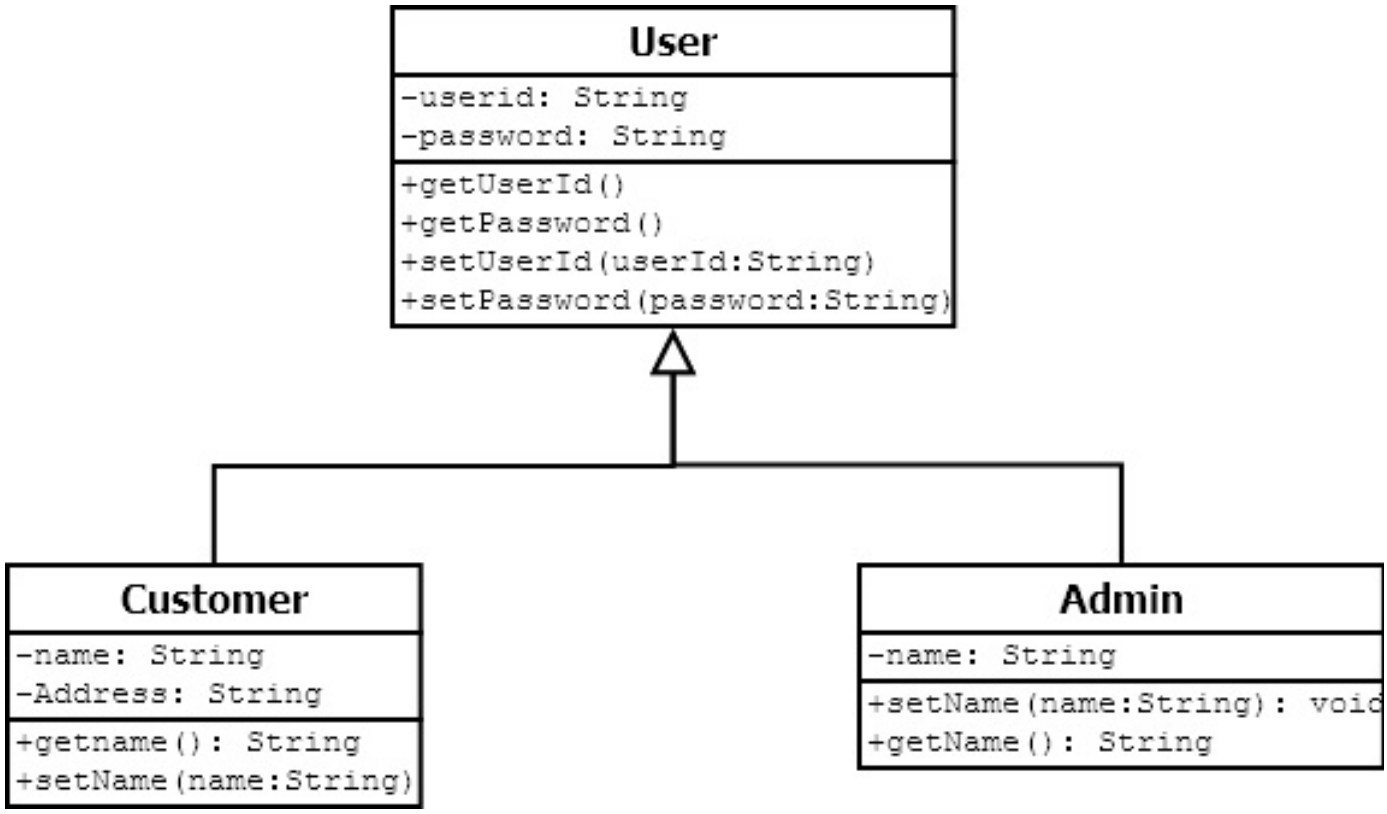



```
//Address.java
public class Address {
    private int id;
    private String line;
    private String city;
    private String country;
    //getter, setter, toString
```

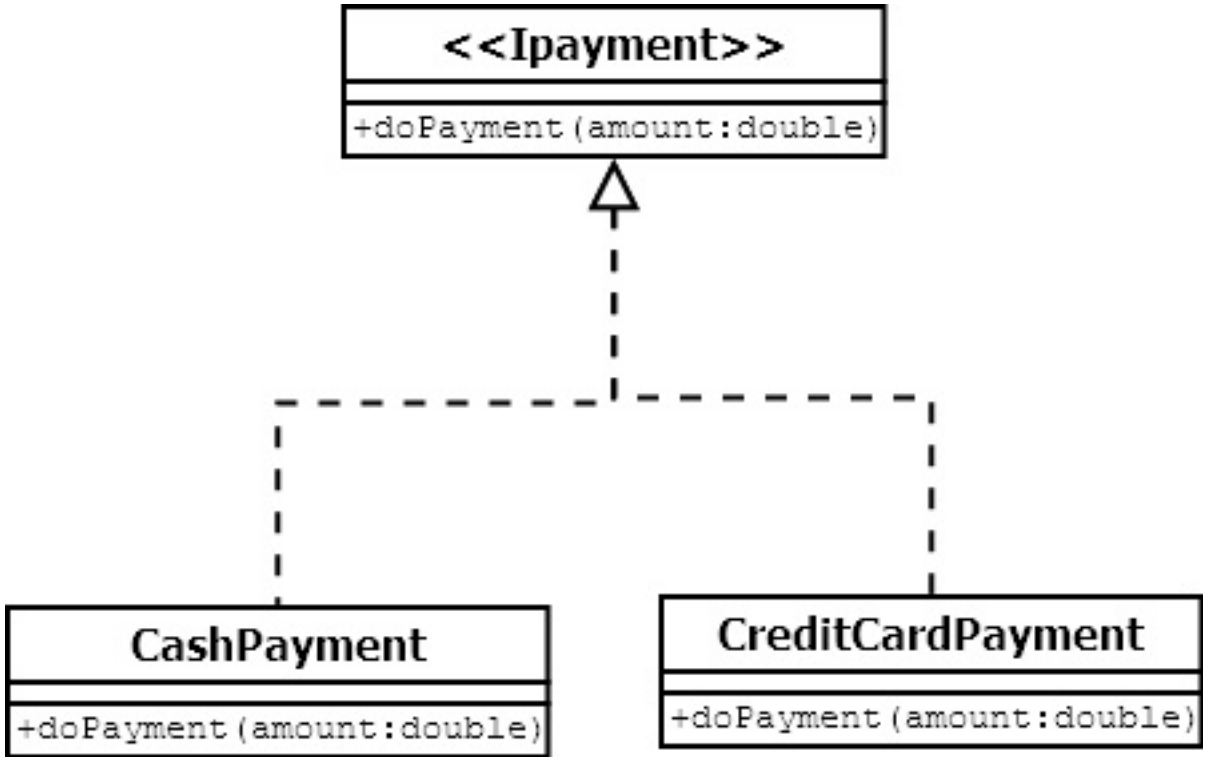
```
}
//Employee.java
public class Employee {
    private long empId;
    private String empName;
    private Address address;// Aggregation
    private double salary;
    //Composition
    public Employee() {
        this.address = new Address();
    }
//getter,setter, toString
}
//CompositionTest.java
public class CompositionTest {
@SuppressWarnings("null")
public static void main(String[] args) {
    //when you initiated Employee, Address initiated as we
ll.
    Employee e = new Employee();
    e.setEmpId(100);
    e.setEmpName("Richard");
    e.setSalary(10000);
    //Get Address then set field
    e.getAddress().setId(1);
    e.getAddress().setLine("hi");
    e.getAddress().setCity("dhaka");
    e.getAddress().setCountry("BD");
```

```
    System.out.println(e);
    e = null;
    if(e!=null){
        System.out.println(e.getAddress());
    }
    else{
        System.out.println("No address");
    }
    }
}
```

5.3. Generalization

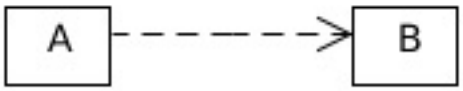


5.4. Realization



6. Dependency:

A depends on B. This is a very loose relationship.




```
public class Customer {
    private String customerId;
    private String customerName;
    //getter and setter
    }
```

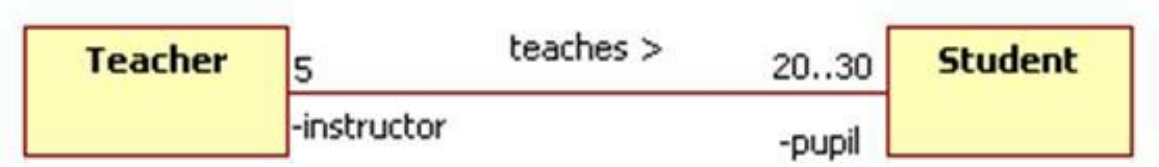
```
public class CustomerView {
    public void displayCustomer(Customer c){
        System.out.println("Customer Id:"+c.getCustomerId(
)+
```

```
                    " Customer Name:"+c.getCustomerName()+" "
                    );
    }
}
```

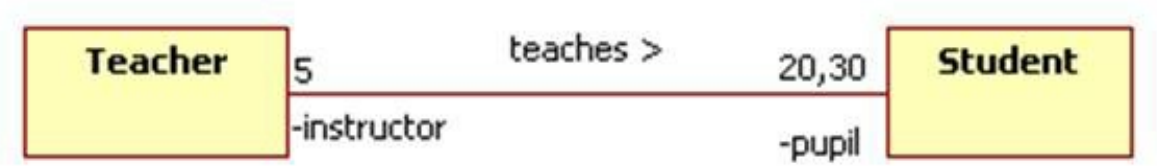
```
public class CustomerTest {
    public static void main(String[] args) {
        Customer richard = new Customer();
        richard.setCustomerId("C001");
        richard.setCustomerName("Richard");
        CustomerView cv = new CustomerView();
        cv.displayCustomer(richard);
    }
}
```

7. Multiplicity

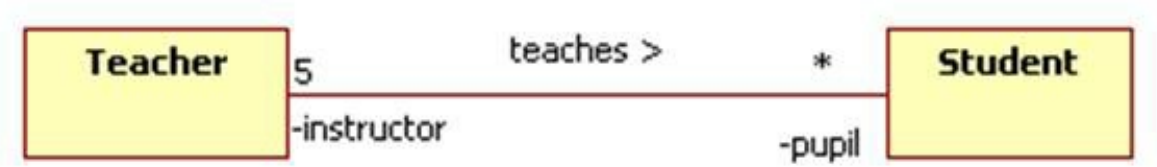
- A Teacher has between 20 to 30 students in a term, and that a student has exactly five teachers.



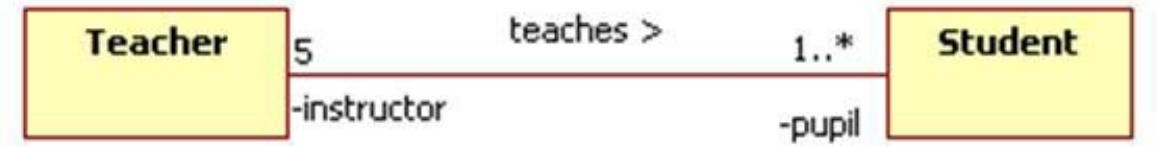
- If a teacher had 20 or 30 students, then the class diagram.



- If a teacher had zero or more students, then the class diagram.



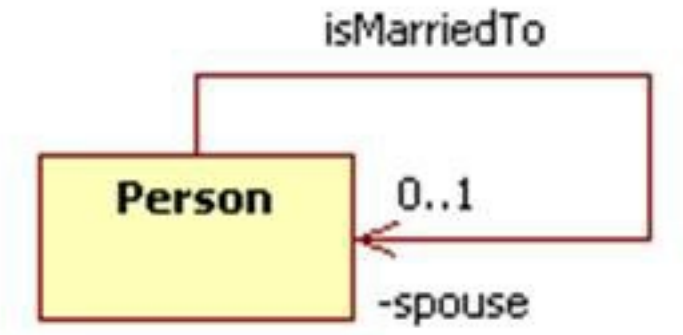
- If a teacher had one or more students,then the class diagram.




```
Class Teacher{
private String teacherId;
private String teacherName;
private List<Student> pupil;// 0 or more// 1 or more
//setter and getter
    }
}

Class Student{
private String studentId;
private String studentName;
private List<Teacher> instructors;
//setter and getter
```

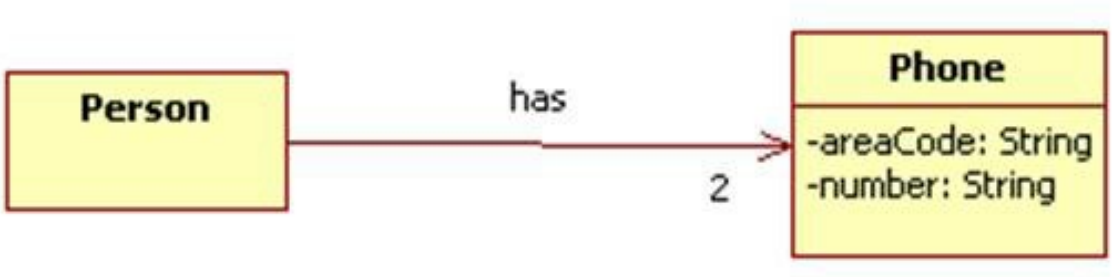
```
    }
}
```

- Self Association



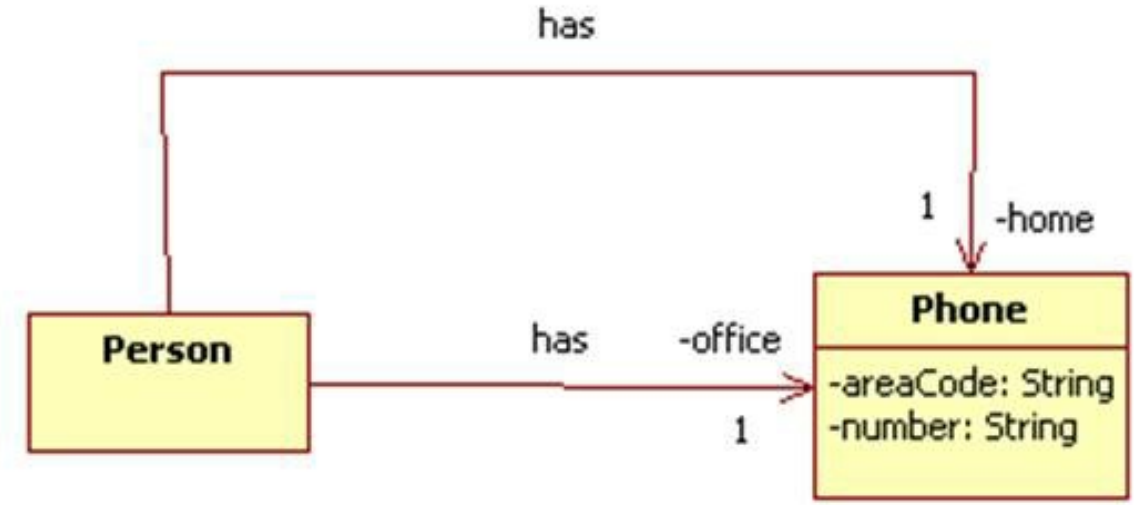

```
class Person {
    private Person spouse;
    // etc.
}
```

- Association Example 1



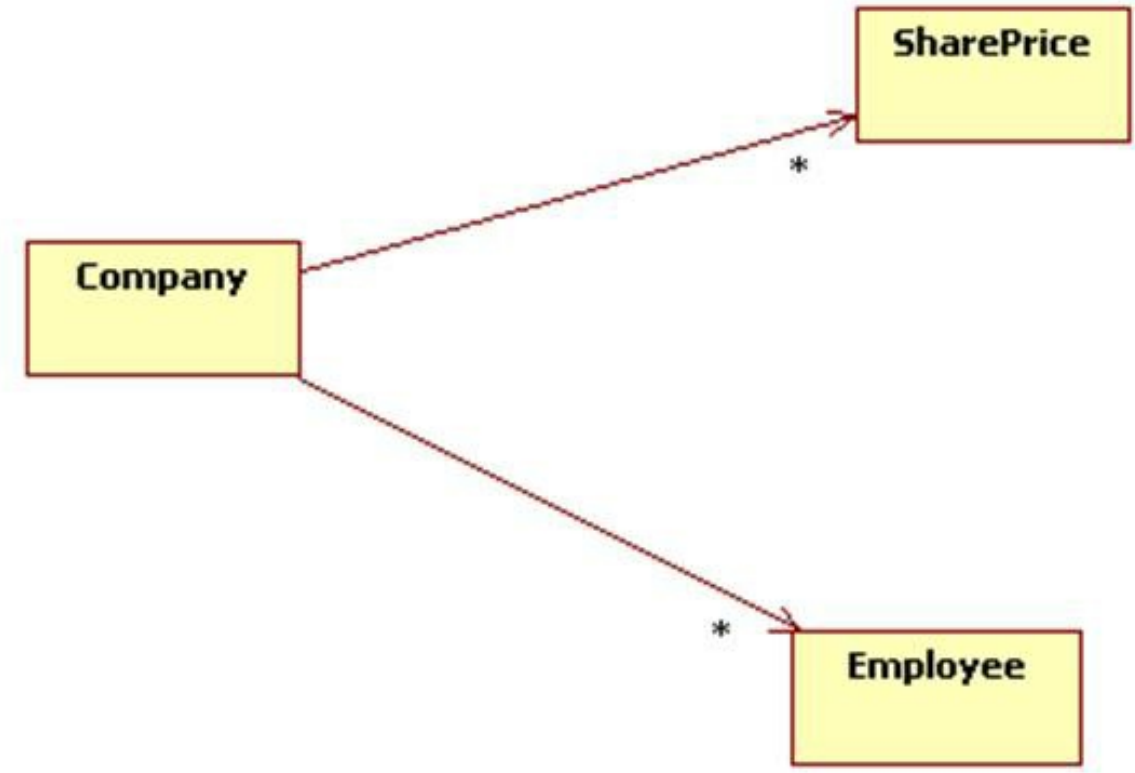

```
class Person {
    private Phone[] phones = new Phone[2];
    // etc.
}
```

- Association Example 2




```
class Person {
   private Phone home;
   private Phone office;
   // etc.
}
```

- Association Example 3



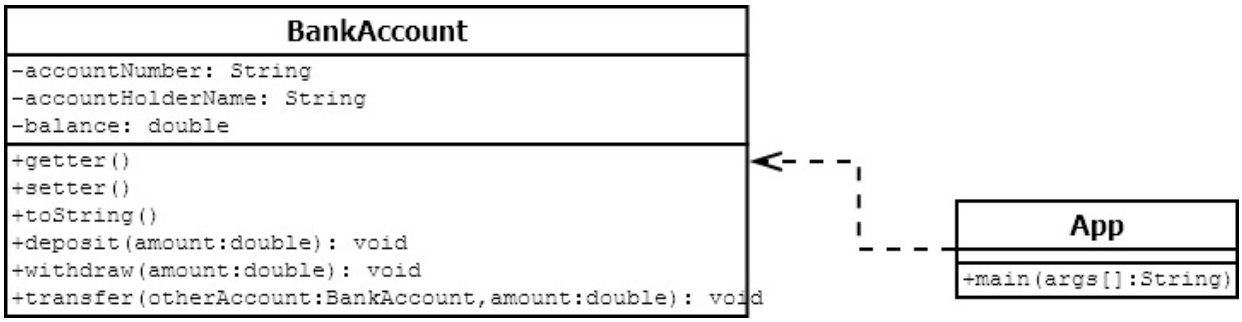

```
class Company {
    private Collection<Employee> employees;
    private Collection<SharePrice> sharePrices;
    // etc.
}
```

7. Coupling

8. Cohesion

Excercises

- Write the java code from the following class diagram.



- Watch the video tutorials and Write the java code.
  https://www.youtube.com/watch?v=WEL0qRsFi9I

References

https://www.dariawan.com/tutorials/java/association-aggregation-and-composition-in-java/

http://www.cs.sjsu.edu/~pearce/modules/lectures/uml/class/association