City University

SE 416: Software Engineering Laboratory

Lecture 1 & 2

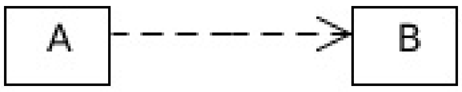
Supta Richard Philip

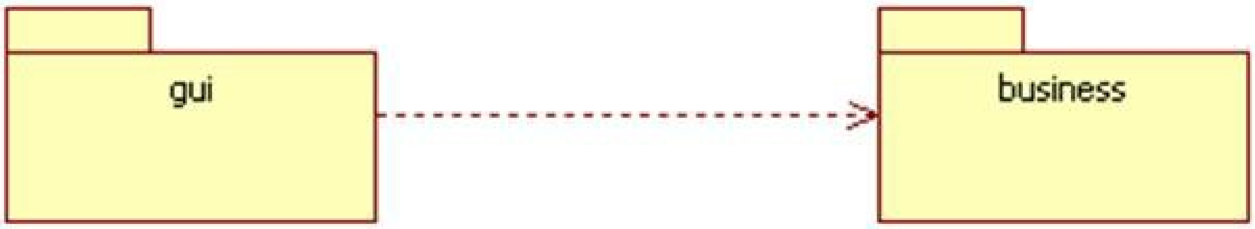
supta.philip@gmail.com

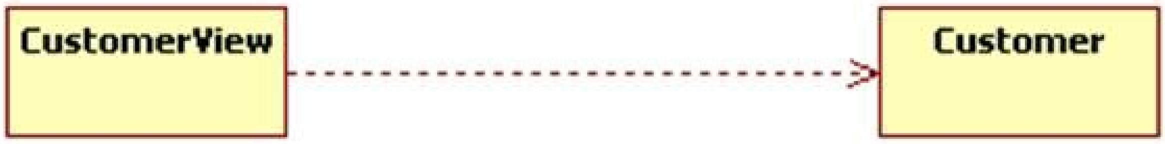
Relationships in UML

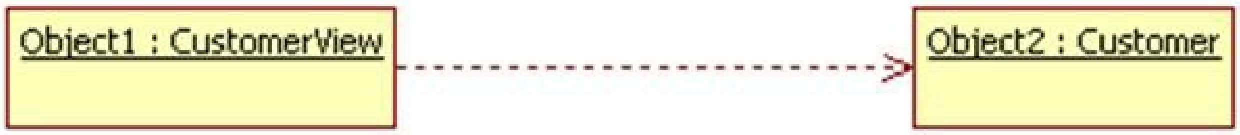
> *It is the messages sent among objects that give a system dynamic behavior, and these are represented in UML through the relationships among classes. There are four kinds of relationships.*

1. Dependency: A depends on B. This is a very loose relationship.







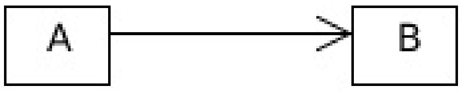


```
public class Customer {
    private String customerId;
    private String customerName;
    //getter and setter
    }
```

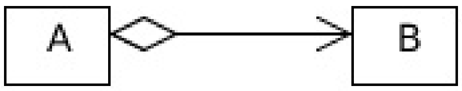
```
public class CustomerView {

    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);
    }
}
```

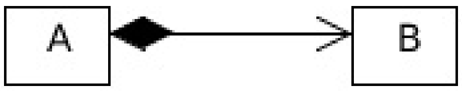
2. Association: A sends messages to a B. In programming terms, it means instances of A can call methods of instances of B, for example, if a B is passed to a method of an A.



3. Aggregation: An A is made up of B. This is a part-to-whole relationship, where A is the whole and B is the part. In code, this essentially implies A has fields of type B.

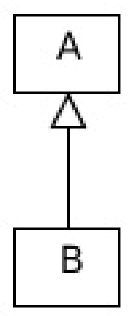


4. Composition: An A is made up of B with lifetime dependency. That is, A aggregates B, and if the A is destroyed, its B are destroyed as well.



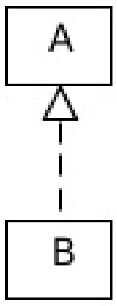
Two other important relationships deal with the relationship among classes.

1. Generalization: A generalizes B. Equivalently, B is a subclass of A. In Java, this is extends.



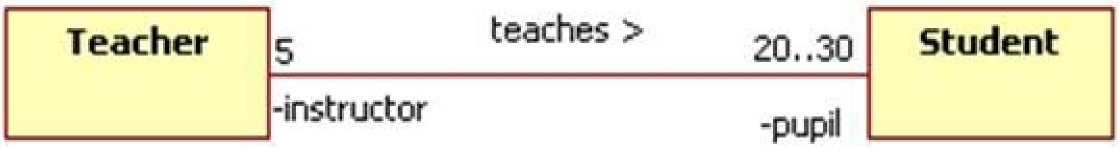
2. Realization: B realizes (the interface defined in) A. As the parenthetical name implies, this is used to show that a class realizes an interface. In Java, this is implements, and so it would

be common for A to have the «interface» stereotype.

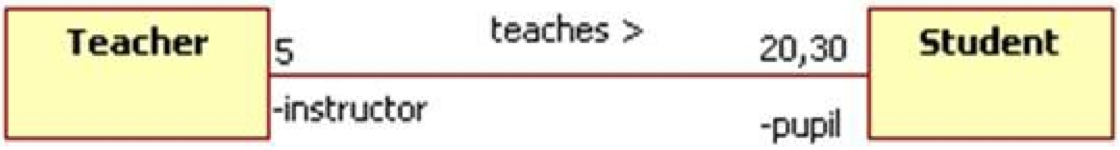


Multiplicity

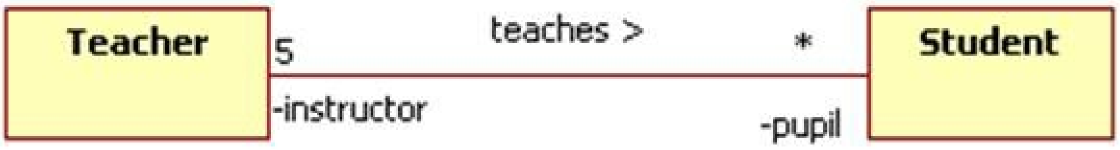
- A Teacher has between 20 to 30 students in a term, and that a student has exactly five teachers.



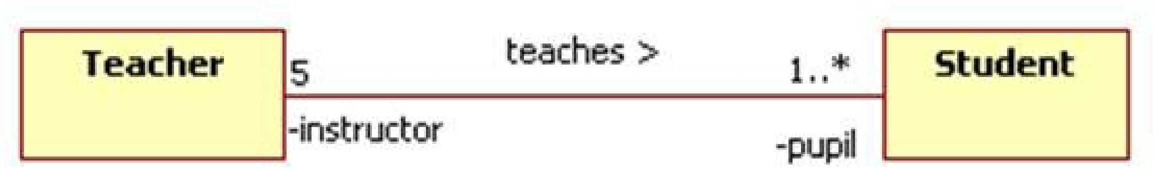
- If a teacher had 20 or 30 students, then the class diagram.



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

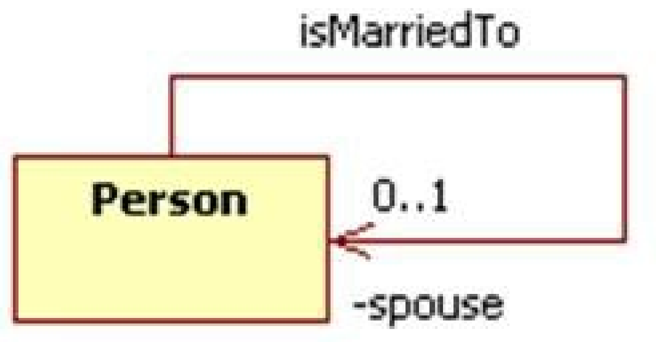


- If a teacher had one or more students,then the class diagram.



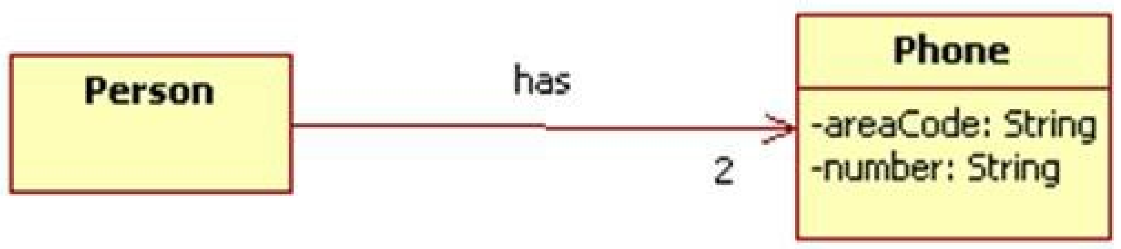
```
Class Teacher{
private String teacherId;
private String teacherName;
private List<Student> pupil;
//setter and getter
    }
}

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

- Self Association



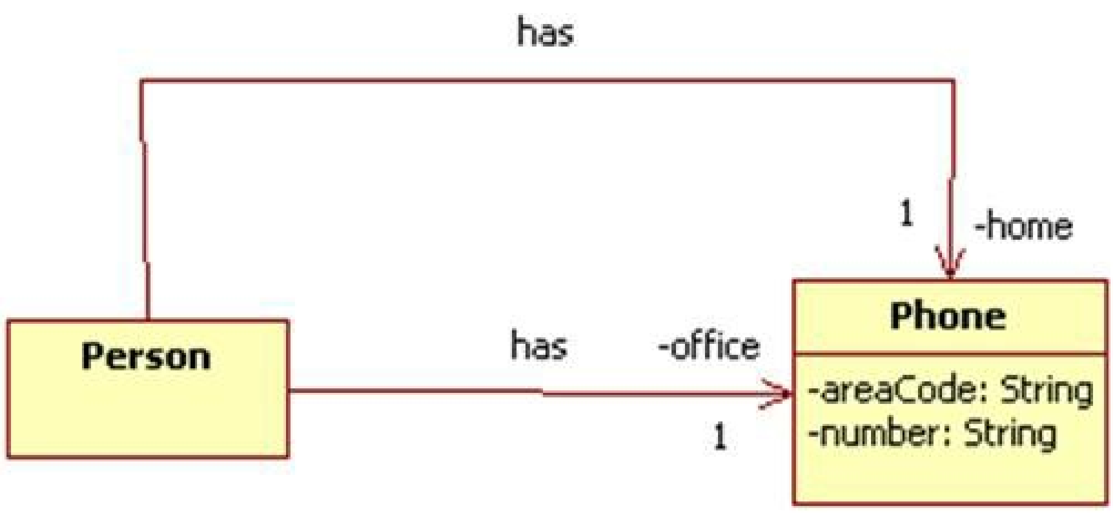
```
class Person {
    private Person spouse;
    // etc.
}
```

- Association Example 1



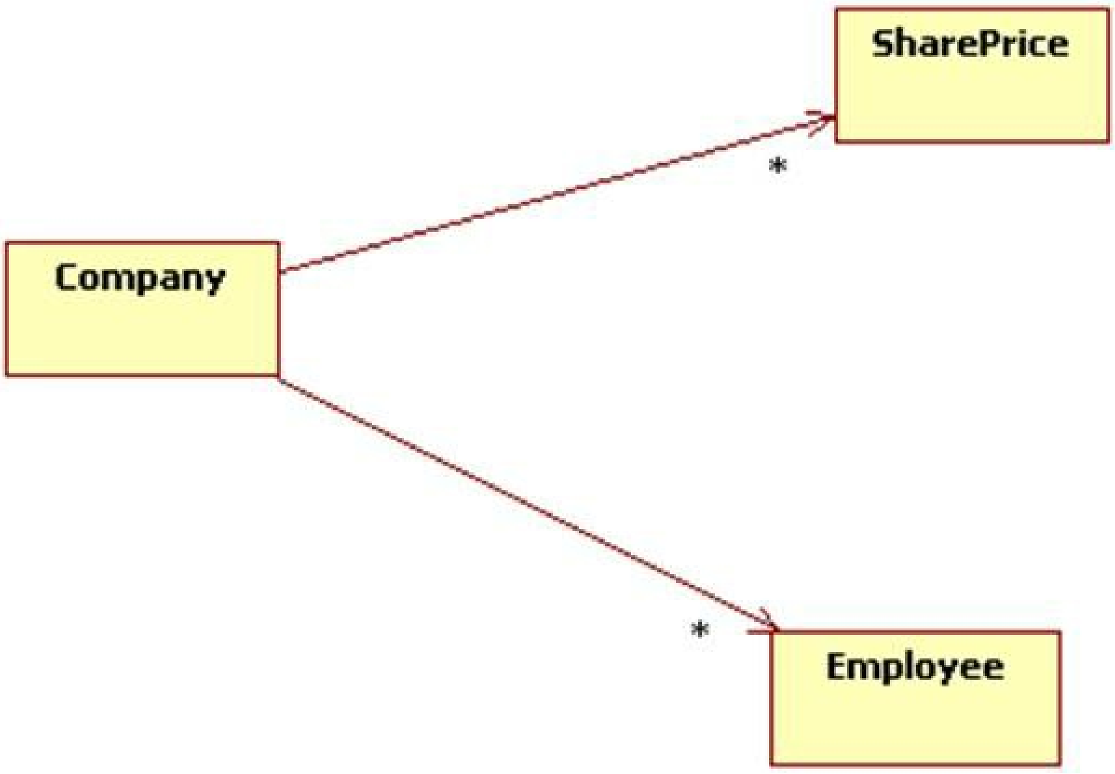
```
class Person {
    private Phone[] phones = new Phone[2];
    // etc.
}
```

- Association Example 2



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

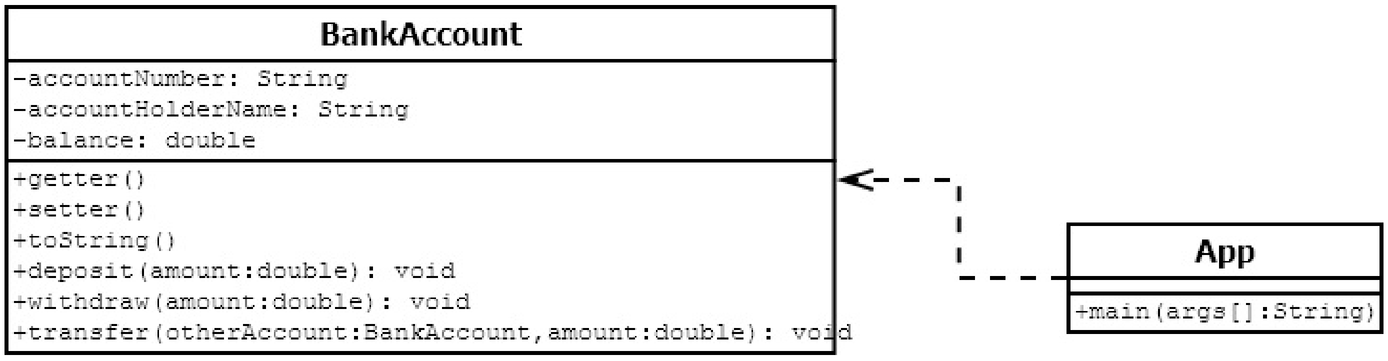
- Association Example 3



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

Practice Problem

- Write the java code from the following class diagram.


BankAccount
-accountNumber: String
-accountHolderName: String
-balance: double
+getter()
+setter()
+toString()
+deposit(amount:double): void
+withdraw(amount:double): void
+transfer(otherAccount:BankAccount,amount:double): void
App
+main(args[]:String)

References

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

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