Object Oriented Programming

Java

Part 5: Associations

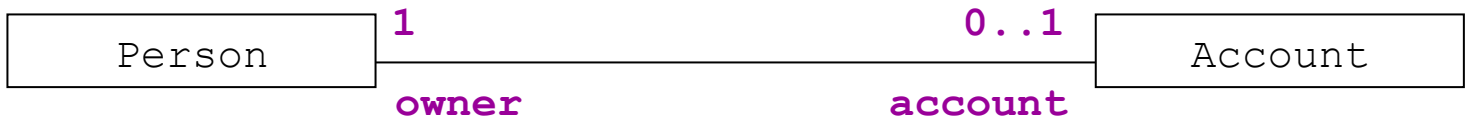
Association – revision

- An **association** represents a reference between objects.
- In an association are defined:
  - **Identifier** – term that describes the association.
  - **Role** – roles represented by the association in each of the related classes.
  - **Multiplicity** – number of objects associated in each of the related association.

Association (1)

- **Association**: represented by fields of the associated type.
  1. The association is established when both fields are initialized.
  2. Deleting an association requires that both fields cease to have a reference to the associated object.

Association (2)



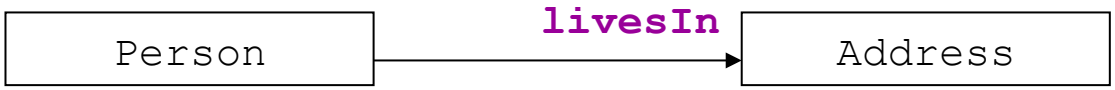

```
public class Person{
   Account account;
   ...
   Person(){
        account = null;
         ...
   }
   void associateAccount(Account a) {
        account = a;
   }
}
```

```
public class Account{
   Person owner;
   ...
   Account(Person o){
        owner = o;
         ...
   }
}
```

Association (3)

- **Directed associations:** in Java, only the from-class contains a field to the the to-class.
- **Association multiplicities**: in Java, multiplicity distinct from `0..1` and `1` is implemented with fields of type array, `Vector`, … in the associated classes.
- **Associations with extra information**: in Java, the associated classes have an extra field to the association class.

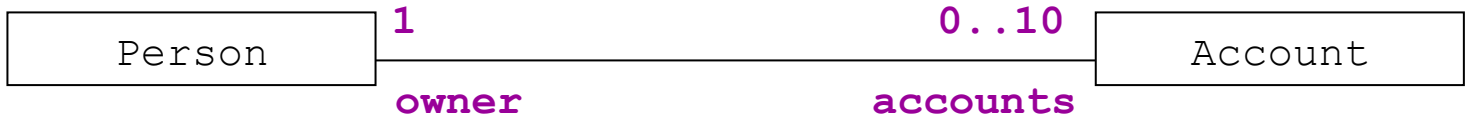
Association (4)




```
public class Person{
   Account account;
   Address livesIn;
   ...
   Person(Address a) {
        account = null;
        linesIn = a;
         ...
   }
}
```

```
public class Address{
   String street;
   ...
}
```

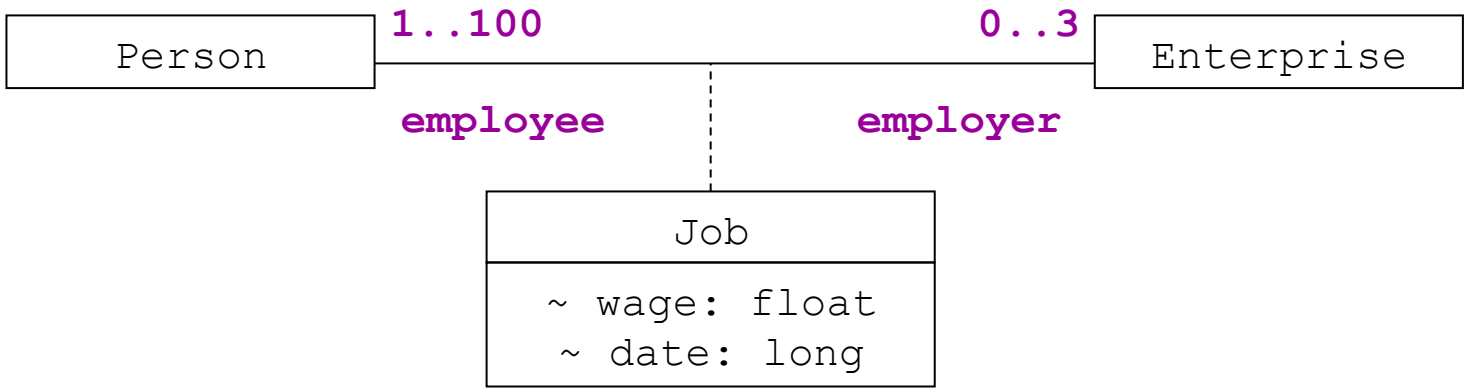
Association (5)




```
public class Person{
   int idxNextAccount=0;
   static final int nbMaxAccount=10;
   Account[] accounts;
   ...
   Person() {
        accounts = new Account[nbMaxAccount];
   }
   void associateAccount(Account c){
        if (idxNextAccount<nbMaxAccount)
                accounts[idxNextAccount++] = c;
        else
                System.out.println("Maximum number attained!");
   }
}
```

```
public class Account{
   Person owner;
   ...
   Account(Person o){
        owner = o;
   }
}
```

Association (6)




```
public class Person{
   Enterprise[] employers;
   Job[] jobs;
   ...
   void associateJob(
        Enterprise employer,
        Job job){...}
}
```

```
public class Job{
   float wage;
   long date;
   ...
}
```

Association (7)

```
public class Enterprise{
    int idxNextJob=0;
    static final int nbMaxJobs=100;
    Person[] employees;
    Job[] jobs;
    ...
    Enterprise() {
          employees = new Person[nbMaxJobs];
          jobs = new Emprego[nbMaxJobs];
          ...
    }
    void newJob(Person person, Job job) {
          if (idxNextJob<nbMaxJobs) {
                employees[idxNextJob]=person;
                jobs[idxNextJob++]=job;
          } else System.out.println("Maximum number attained!");
    }
}
```

Association (8)

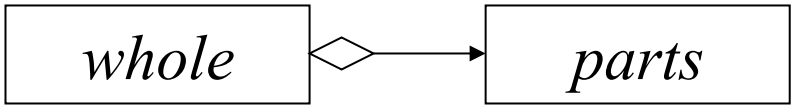
- **It is the responsibility of the programmer to ensure the proper establishment of associations and maintenance of their consistency.**

Aggregation/Composition – revision (1)

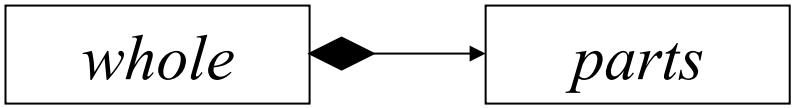
- An **aggregation/composition** is an association which denotes that the *whole* is formed by *parts*.
- The aggregation/composition is said to be a relationship of **“has-a”**.

Aggregation/Composition – revision (2)

- The **aggregation** is an association which denotes that the *whole* is formed by *parts*.



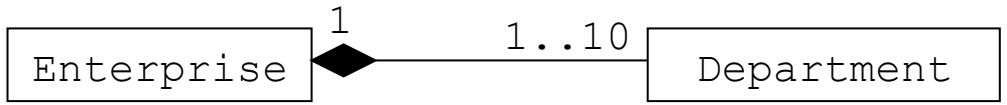
- In **composition** there is no-sharing, that is, when the owning object is destroyed, so are the contained objects.



Aggregation/Composition (1)

- An **aggregation/composition** is implemented in Java as an association, with fields which are references to the associated objects.
- **Regarding composition, it is still necessary to ensure consistency over the disappearance of the *whole*, which in turn implies the disappearance of the *parts*.**

Aggregation/Composition (2)




```
public class Enterprise{
   int idxNextDepartment = 0;
   static final int nbMaxDepartments = 10;
   Department[] departments;
   ...
   Enterprise() {
        departments = new Department[nbMaxDepartments];
        ...
   }
```

Aggregation/Composition (3)

```
    // ... Continues previous slide

    void newDepartment () {
        if (idxNextDepartment<nbMaxDepartments)
            departments[nbNextDepartment++]
              = new Department(this);
        else System.out.println("Maximum number attained!");
    }
}
```

```
public class Department {
    Enterprise enterprise;
    ...
    Department(Enterprise e) {
        enterprise = e;
    }
}
```