Lecture 21

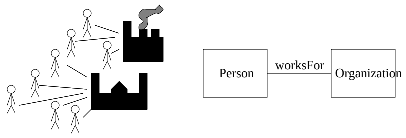
In the ‘real world’, there are often relationships between objects: one object is associated with another.

If this is so in the real world, then we want to model it in our software systems.

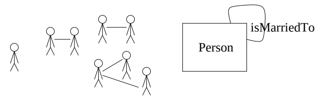
In this lecture, we see how this is done. It exploits the notion of reference types.

Associations

- Associations can exist between objects of different classes

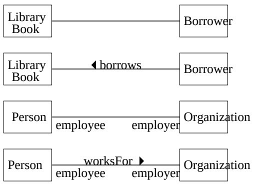


- They can also exist between objects of the same class



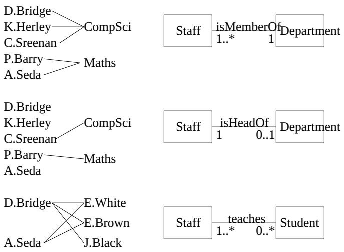
Diagrammatic conventions

- Diagrams help us to visualise programs that comprise inter-related objects from many classes
- Optionally, we can name the association and/or name the roles that objects play in the association



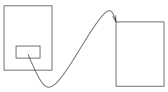
Multiplicities

- Optionally, we can specify the minimum/maximum number of objects that can participate in an association, e.g. 1, 0..1, 0..*, 1..*, etc.

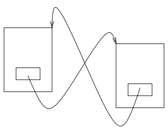


Implementing associations in Java

- In object A, use an instance variable to hold a reference to object B



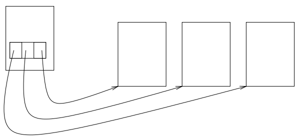
- Optionally, *also* use an instance variable in object B to hold a reference to object A



- This makes it possible for the objects to send each other messages

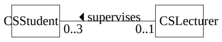
Implementing multiplicities

- Sometimes an object needs a data structure (e.g. an array or better still something that implements `Set` or `List`) to hold several references



- Constructors & setters may also have to check that min/max multiplicities are being properly observed

Example




```
public class CSStudent
{
   public CSStudent(PersonName aName)
   {  name = aName;
   }

   public PersonName getName()
   {  return name;
   }
```



```
   public CSLecturer getSupervisor()
   {
   }

   public void setSupervisor(
      CSLecturer theSupervisor)
   {
   }

   private PersonName name;


}
```



```
public class CSLecturer
{
   public CSLecturer(PersonName aName)
   {  name = aName;
      supervisees =
         new CSStudent[MAX_SUPERVISEES];
   }

   public PersonName getName()
   {  return name;
   }
```



```
   public CSStudent[] getSupervisees()
   {  return supervisees;
   }

   public void addSupervisee(
      CSStudent aSupervisee)
   {  /* Find an empty cell in the array
         & insert aSupervisee into that
         cell.
       */
   }

   private PersonName name;
   private CSStudent[] supervisees;
   public static final int
                  MAX_SUPERVISEES = 3;
}
```


Some client code

```
CSLecturer lect =
  new CSLecturer(
   new PersonName("Dr", "Guy", "Dance"));
CSStudent stud1 =
  new CSStudent(
   new PersonName("Mr", "Reggie", "Mental"))
CSStudent stud2 =
  new CSStudent(
   new PersonName("Ms", "Anna", "Key"));
stud1.setSupervisor(lect);
lect.addSupervisee(stud1);
stud2.setSupervisor(lect);
lect.addSupervisee(stud2);
System.out.println(
  stud2.getSupervisor().getName().
                          getSurname());
```


A word of warning

- Consider the following client code:

  ```
  CSStudents[] someStudents =
                    lect.getSupervisees();
  ```

- The variable `someStudents` now references the same array as `lect`'s instance variable

- Consider the consequences of doing things to the contents of `someStudents`, e.g.:

  ```
  someStudents[0] = null;
  ```