Ch 7 Designing Java Classes Part II

Associations

Methods

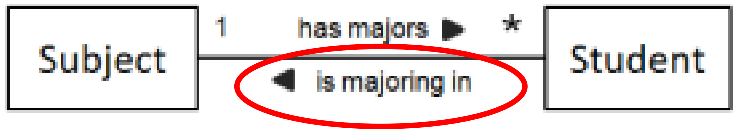
- Readability of code
- Reusing code
- Parameters vs arguments

Associations

Classes will have relationships with other classes. When designing you must decide whether to implement an association, and how to implement it.

Figure 7.1

Figure 7.1: A simple Class Diagram.



A student *is majoring in* a subject

Our assumption

A student will have **at most one major**

The student class has a private field with getter/setter

Implementation **in Student**:

```
private Subject major;
```

Private field

```
public Subject getMajor(){
    return major;
}
```

getter

```
public void setMajor(Subject newMajor){
    major = newMajor;
}
```

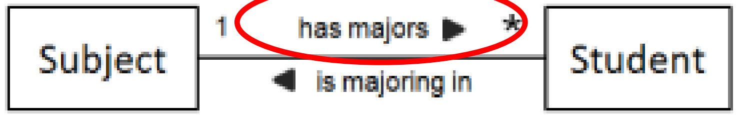
setter

Associations

Classes will have relationships with other classes. When designing you must decide whether to implement an association, and how to implement it.

Figure 7.1

Figure 7.1: A simple Class Diagram.

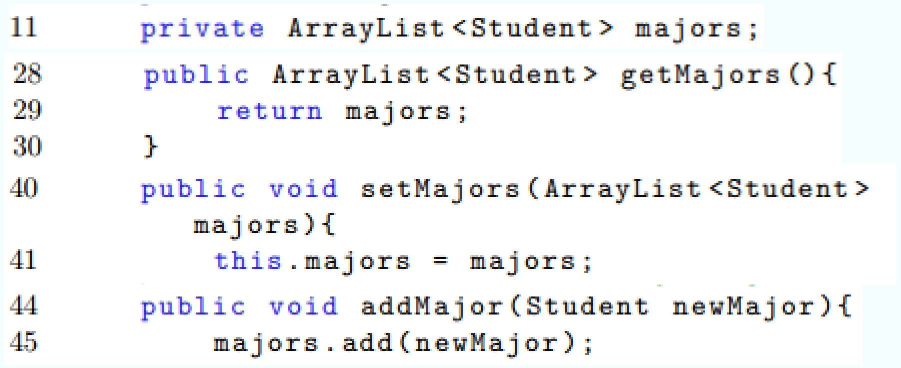


A subject *__has majors__* (students)

A subject may have many majors

The Subject class has an ArrayList with getter/setter & addMajor(…)

Implementation **in Subject**:



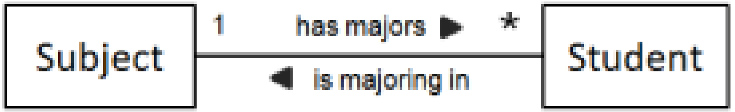
```
    private ArrayList<Student> majors;
    public ArrayList<Student> getMajors(){
        return majors;
    }
    public void setMajors(ArrayList<Student>
      majors){
        this.majors = majors;
    public void addMajor(Student newMajor){
        majors.add(newMajor);
```

Java Classes – making a connection between objects

Associations:

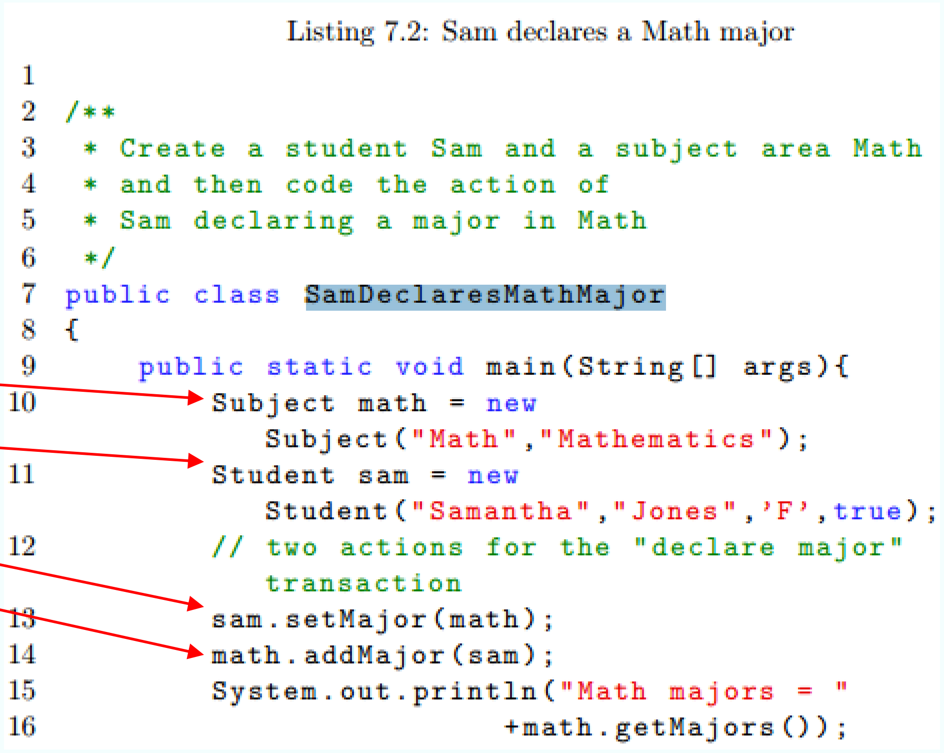
Figure 7.1

Figure 7.1: A simple Class Diagram.



Consider SamDeclaresMathMajor.java

1. Instantiate a subject … math
2. Instantiate a student … sam
3. Set Sam's major to be math
4. Add Sam to the list of math majors



Listing 7.2: Sam declares a Math major

```java

/**
 * Create a student Sam and a subject area Math
 * and then code the action of
 * Sam declaring a major in Math
 */
public class SamDeclaresMathMajor
{
    public static void main(String[] args){
        Subject math = new
           Subject("Math","Mathematics");
        Student sam = new
           Student("Samantha","Jones",'F',true);
        // two actions for the "declare major"
           transaction
        sam.setMajor(math);
        math.addMajor(sam);
        System.out.println("Math majors = "
                        +math.getMajors());
```

Java Classes- **reusing code**

Methods are used for two purposes

1. To make a program more readable through decomposition
2. To reuse code instead of duplicating code

Consider that the code in SamDeclaresMathMajor.java could be replicated for every student declaring a major

```
jill.setMajor (math) ;
math.addMajor (jill) ;
sam.setMajor (math) ;
acs.addMajor (sam) ;
bob.setMajor (math) ;
acs.addMajor (bob) ;
```

For jill

For sam

For bob

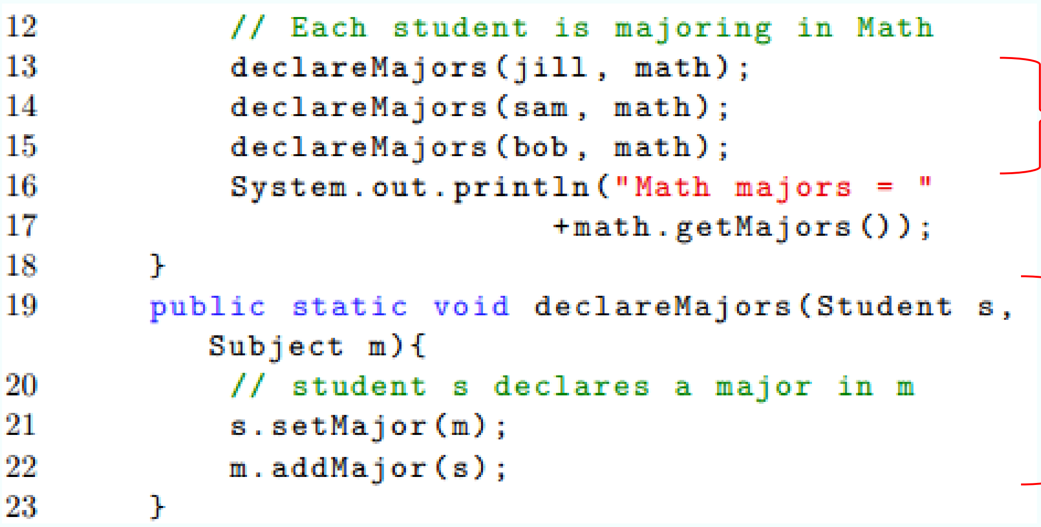
We can replace this kind of code by writing a method that sets a student's major and adds the student to a subject →

Java Classes- **reusing code**

Methods are used for two purposes

1. To make a program more readable through decomposition
2. To reuse code instead of duplicating code

Consider that the code in SamDeclaresMathMajor.java could be replicated for every student declaring a major



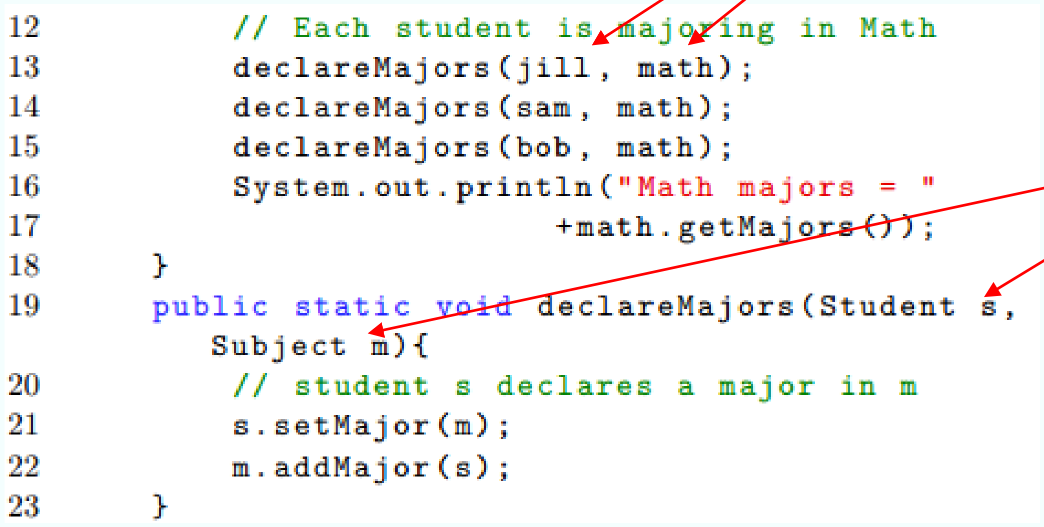
```
            // Each student is majoring in Math
            declareMajors(jill, math);
            declareMajors(sam, math);
            declareMajors(bob, math);
            System.out.println("Math majors = "
                                +math.getMajors());
        }
        public static void declareMajors(Student s,
           Subject m){
            // student s declares a major in m
            s.setMajor(m);
            m.addMajor(s);
        }
```

Three calls to the method below

A method to handle declaring a major

Java Classes - **Parameters and arguments**

Arguments passed to a method



```
        // Each student is majoring in Math
        declareMajors(jill, math);
        declareMajors(sam, math);
        declareMajors(bob, math);
        System.out.println("Math majors = "
                        +math.getMajors());
    }
    public static void declareMajors(Student s,
       Subject m){
        // student s declares a major in m
        s.setMajor(m);
        m.addMajor(s);
    }
```

Parameters defined for a method

Arguments are copied into the parameters on entry, but there is no copying on return.

Arguments must match parameters by type.

Java Classes - **Parameters and arguments**

Parameter Lists / Arguments

A parameter list defines the type of data that will be passed in to a method

Arguments appear in the call statement.

Arguments are copied into the parameters on entry, but there is no copying on return.

But for objects its possible to modify them in the called method

See ObjectModifiedByCalledMethod.java