10. Inheritance and Polymorphism

15 Oct 2015

Objectives

- Describe the significance of inheritance in OOP.
- Write programs that are easily extensible and modifiable by applying polymorphism in program design.
- Define data members and methods, using the protected modifier.

Inheritance

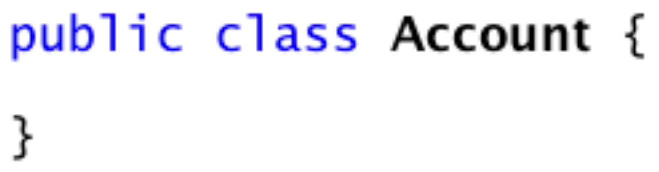
- Inheritance is a mechanism where in a new class is derived from an existing class. In Java, classes may inherit or acquire the non private data members and non private methods of other classes. A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass.

Inheritance

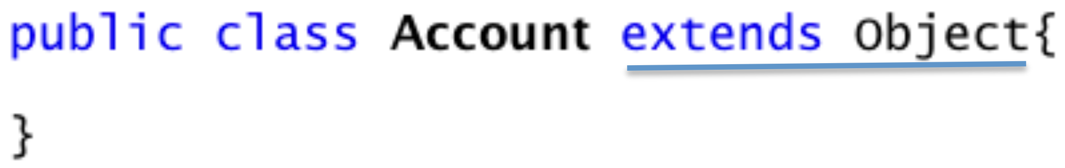
- Every classes in Java must inherit from other class.
- We can declare that one class inherit another class by using the ‘extends’ keyword in the class definition.

Inheritance

- In programmer-defined class, if there is no explicit ‘extends’ declared in that class, it extends Java Object class by default.
  - These 2 classes are equivalent



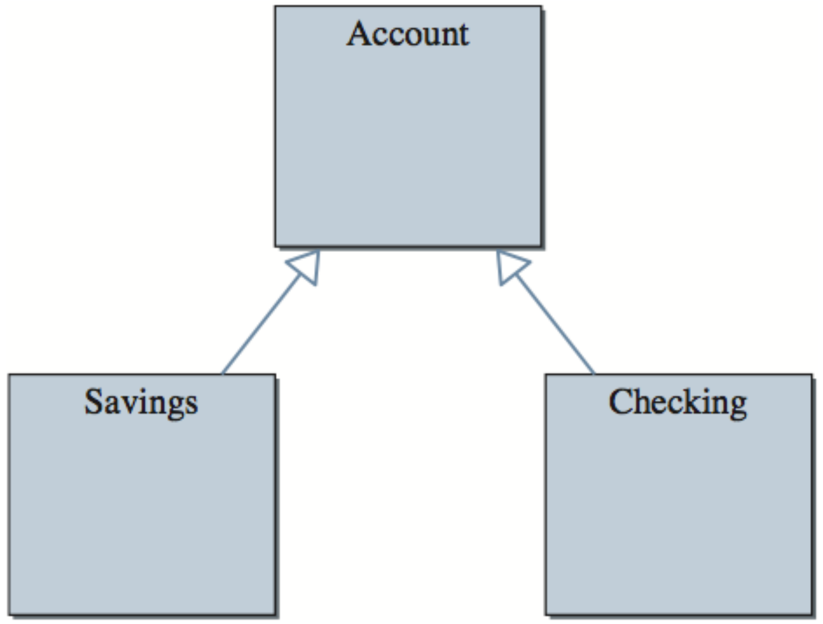
```
public class Account {
}
```



```
public class Account extends Object{
}
```

Inheritance

- Ex. From this class hierarchy, Savings and Checking class acquire all non private data members and non private methods from Account class



Inheritance

– All 3 classes from class hierarchy

```
public class Account {
     ...
}
```

```
public class Saving extends Account{
     ...
}
```

```
public class Checking extends Account{
     ...
}
```

Inheritance

- Superclass:
  - A class that has been extended by another class. It allows the extending class to inherit its state and behaviors.
  - Called superclass, base class or parent class.
  - Eg. Account class

Inheritance

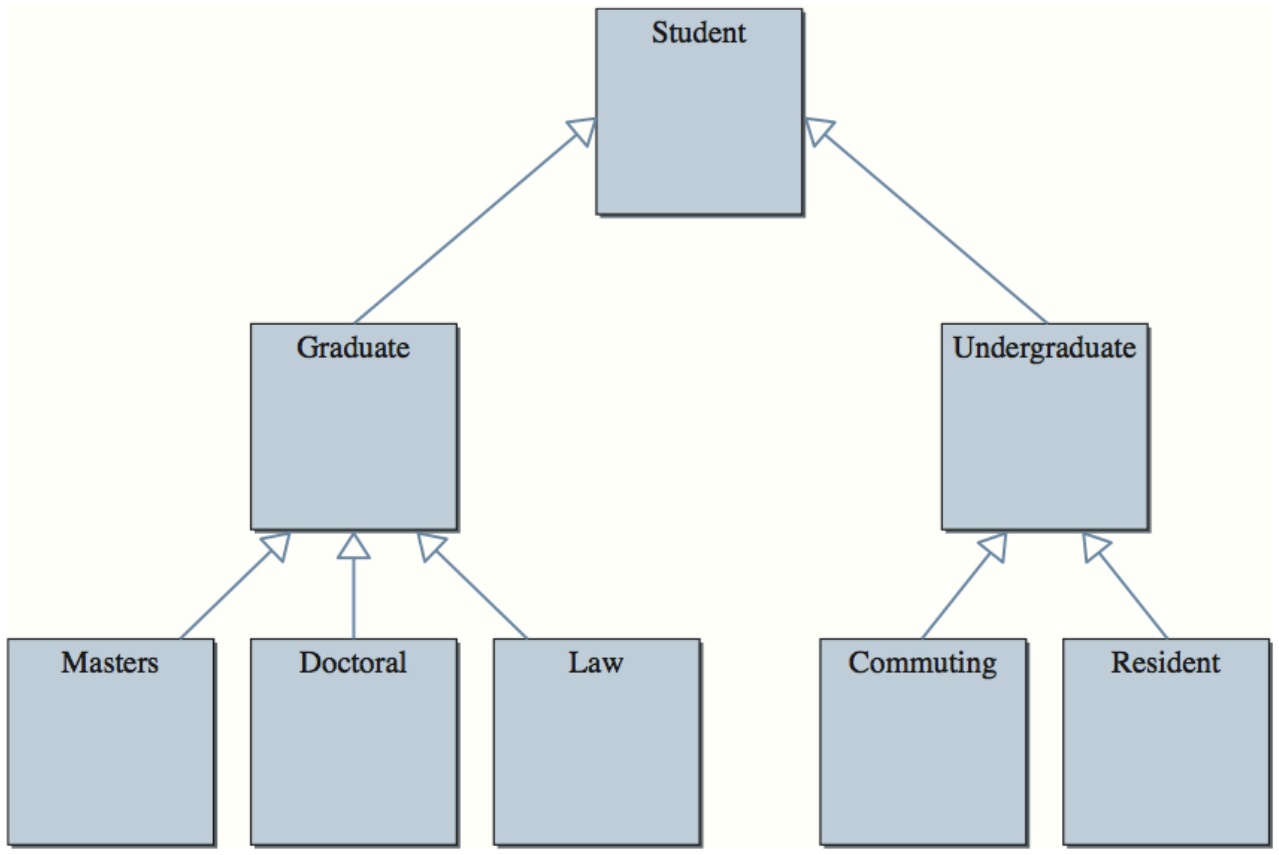
- Subclass:
  - A class that extends another class. The subclass inherits the state and behaviors of the class it extends.
  - Called subclass, derived class or child class.
  - Eg. Savings, Checking class

Inheritance

- Inheritance is very powerful, and if it is used properly, we can develop complex programs very efficiently and elegantly.
- Inheritance is not limited to one level. A subclass can be a superclass of other classes, forming an inheritance hierarchy.

Inheritance

– Inheritance hierarchy of Student

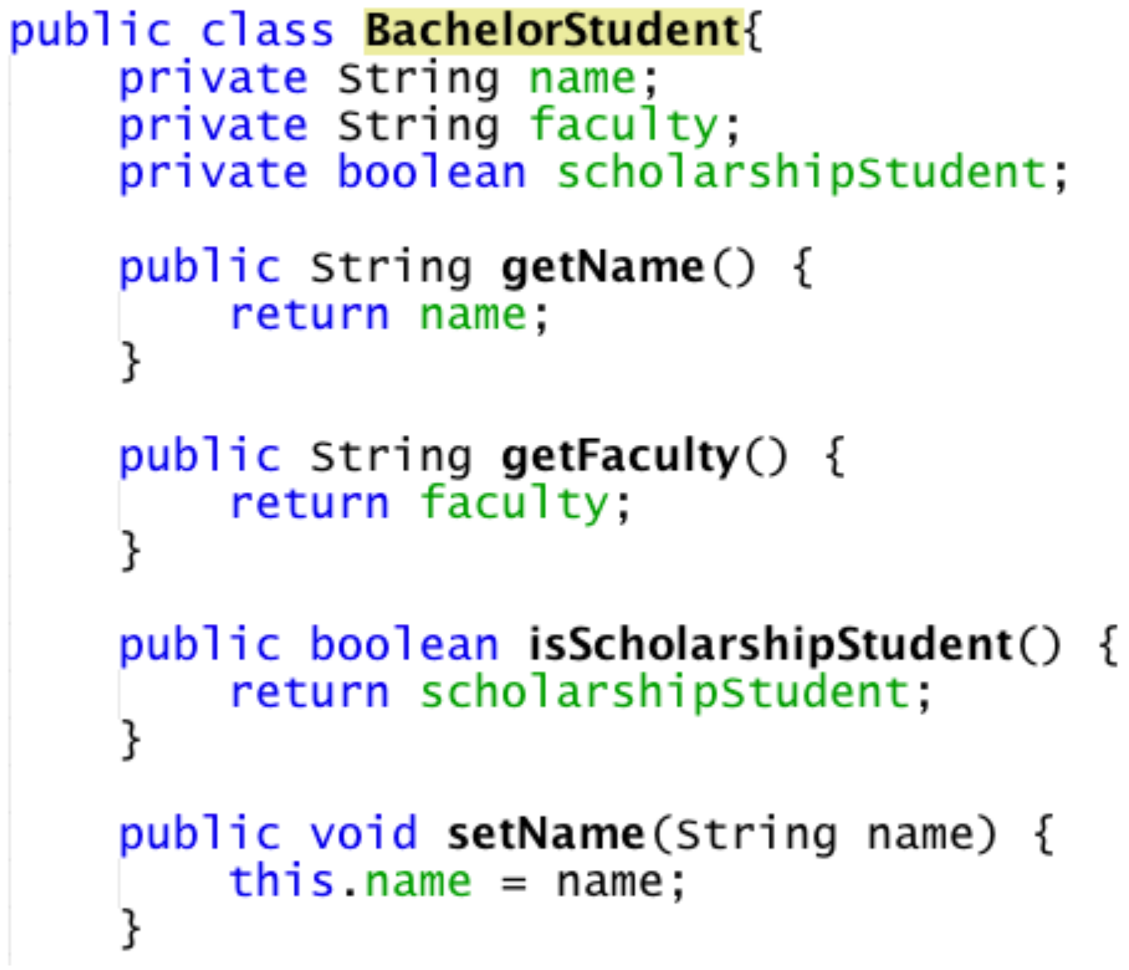


A Simple Example

- Ex. In Student Registration Program, we have 2 types of students
  1. Bachelor degree student
  2. Master degree student

A Simple Example

– BachelorStudent class



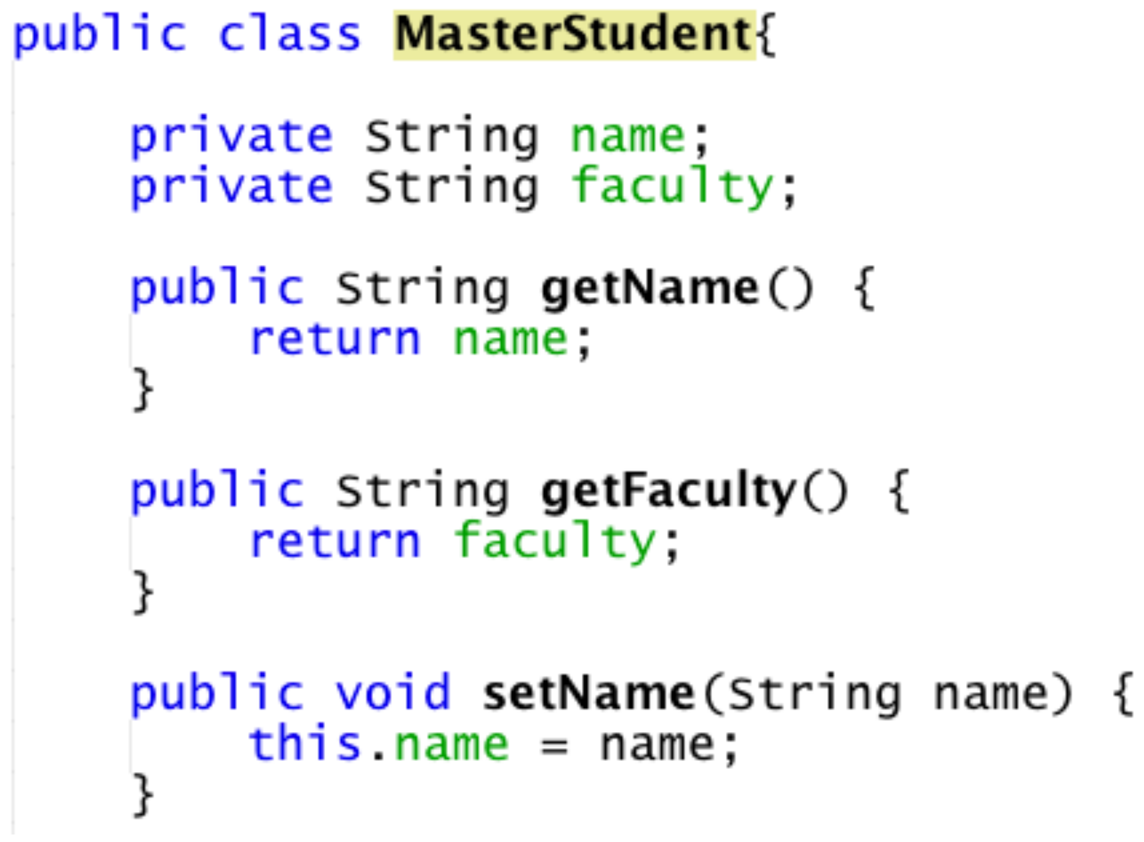
```
public class BachelorStudent{
    private String name;
    private String faculty;
    private boolean scholarshipStudent;

    public String getName() {
        return name;
    }

    public String getFaculty() {
        return faculty;
    }

    public boolean isScholarshipStudent() {
        return scholarshipStudent;
    }

    public void setName(String name) {
        this.name = name;
    }
```

A Simple Example

```java
    public void setFaculty(String faculty) {
        this.faculty = faculty;
    }

    public void setScholarshipStudent(boolean scholarshipStudent) {
        this.scholarshipStudent = scholarshipStudent;
    }

    private boolean isFacultyOfScience(){
        return faculty.equals("Science");
    }

    public double getRegistrationFee(int numberOfCredit){
        if(scholarshipStudent){
            return 0;
        }
        if(isFacultyOfScience()){
            return 5000 + 100 * numberOfCredit;
        }
        return 6000 + 100 * numberOfCredit;
    }
}
```

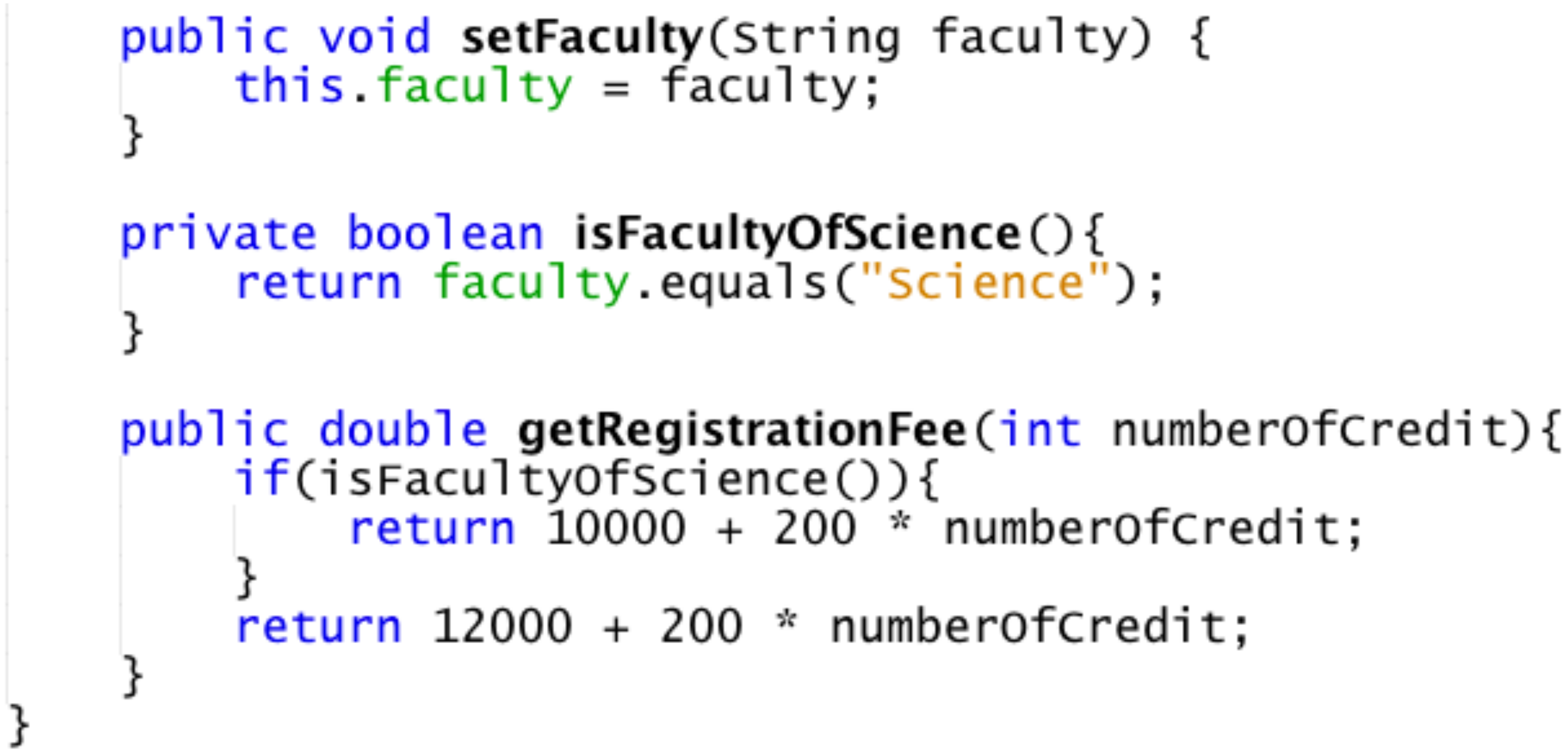
A Simple Example

– MasterStudent class



```
public class MasterStudent{

    private String name;
    private String faculty;

    public String getName() {
        return name;
    }

    public String getFaculty() {
        return faculty;
    }

    public void setName(String name) {
        this.name = name;
    }
```

A Simple Example



```
    public void setFaculty(String faculty) {
        this.faculty = faculty;
    }

    private boolean isFacultyOfScience(){
        return faculty.equals("Science");
    }

    public double getRegistrationFee(int numberOfCredit){
        if(isFacultyOfScience()){
            return 10000 + 200 * numberOfCredit;
        }
        return 12000 + 200 * numberOfCredit;
    }
}
```

A Simple Example

- These 2 classes have some common data members (name and faculty) and methods (getter and setter of name and faculty).
- These 2 classes also have getRegistrationFee() method but the different is how to calculate registration fee.

A Simple Example

- In OOP, we often organize classes in hierarchy to avoid duplication and reduce redundancy by using inheritance.
- By pulling out all the common data members and methods into the superclass, and leave the specialized data members and methods in the subclasses, so do not need to be repeated common codes in all the subclasses.

A Simple Example

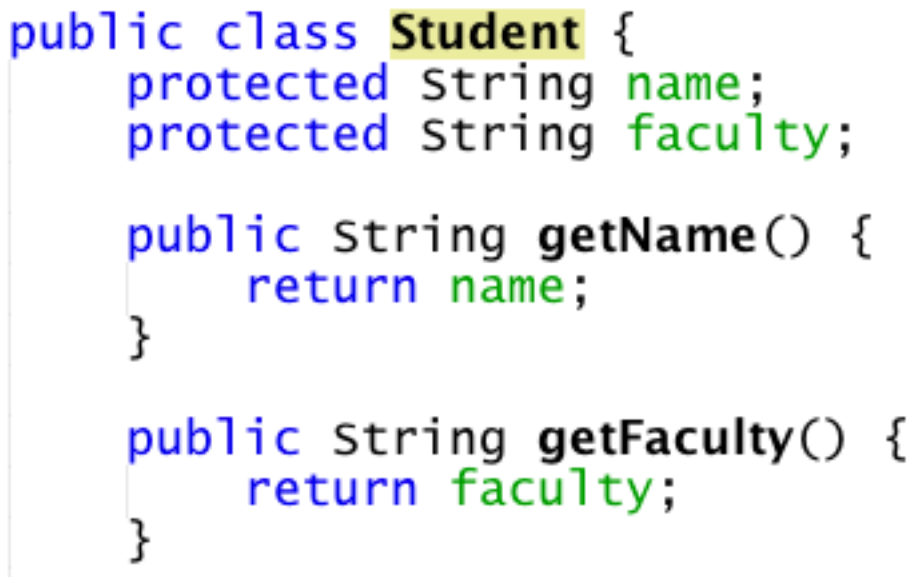
- When we pulling out all the common data members and methods, if it is ‘private’ we should change access modifier ‘private’ to ‘protected’

A Simple Example

- From this example, we create superclass - Student class. In Student class there are common data members and methods
  - Common data members and methods change from ‘private’ to ‘protected’
  - Now getRegistrationFee() method we can edit to return ‘-1’ because this method will never used.

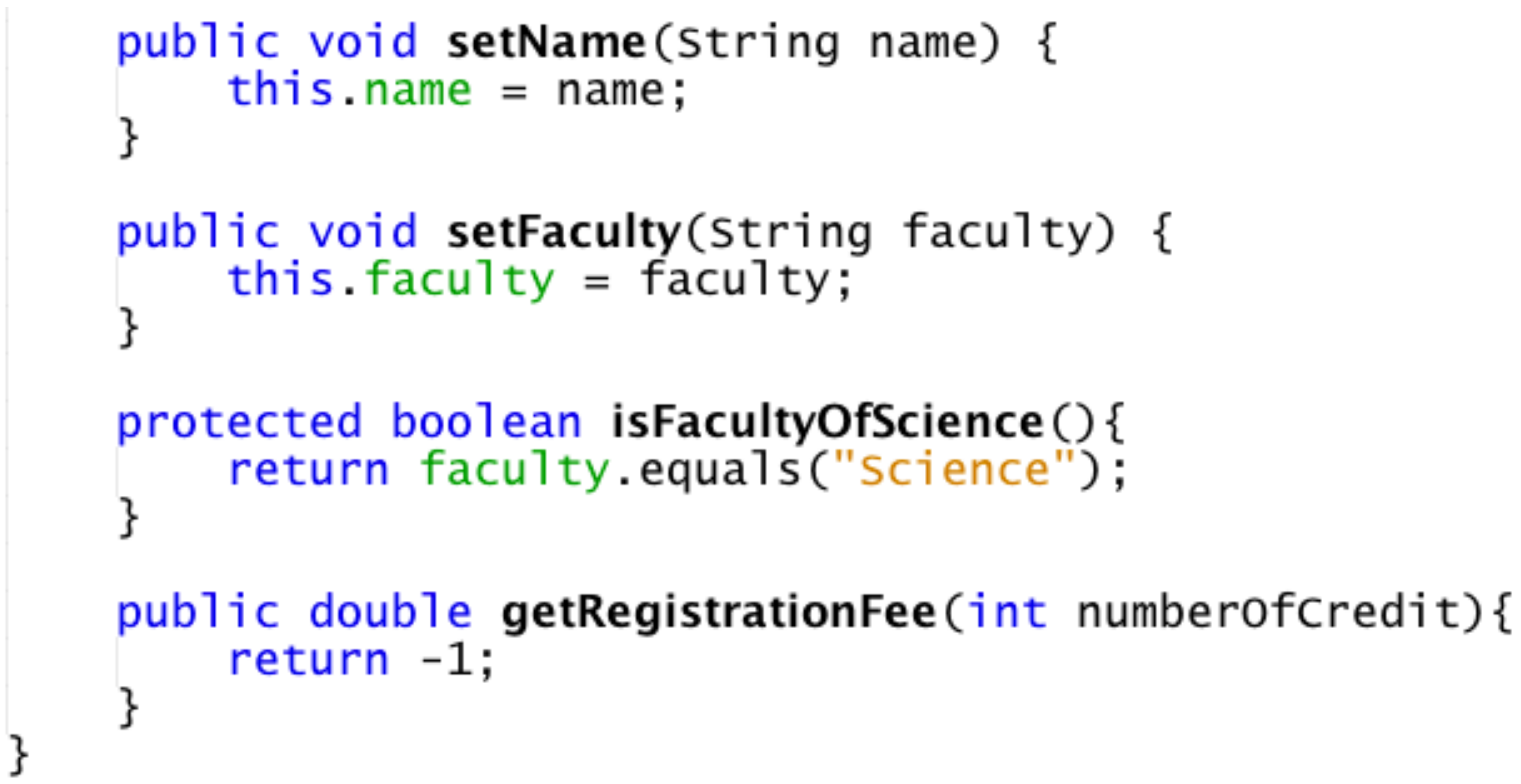
A Simple Example

– Student class



```
public class Student {
    protected String name;
    protected String faculty;

    public String getName() {
        return name;
    }

    public String getFaculty() {
        return faculty;
    }
```

A Simple Example



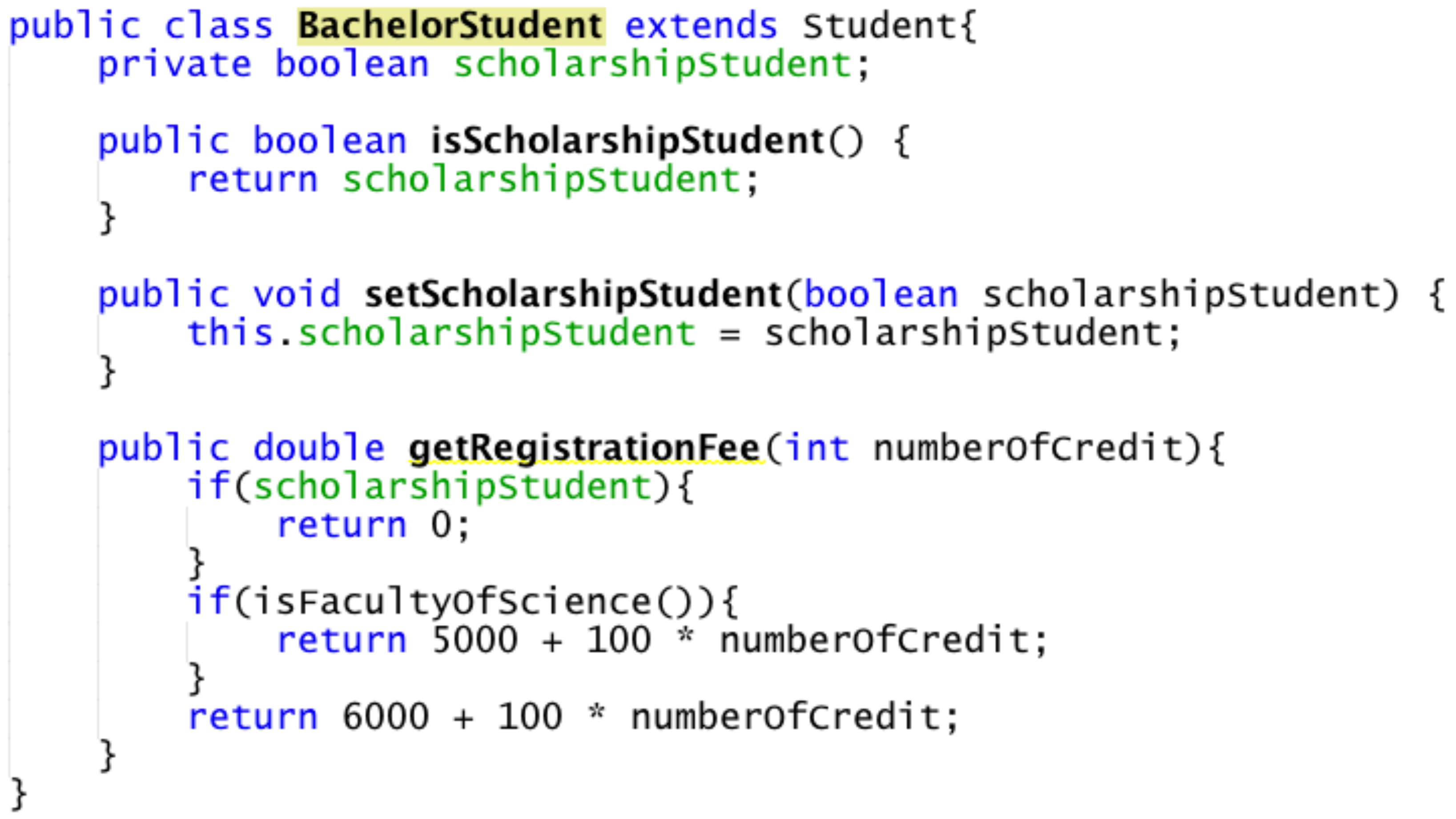
```java
    public void setName(String name) {
        this.name = name;
    }

    public void setFaculty(String faculty) {
        this.faculty = faculty;
    }

    protected boolean isFacultyOfScience(){
        return faculty.equals("Science");
    }

    public double getRegistrationFee(int numberOfCredit){
        return -1;
    }
}
```

A Simple Example

- After pulling out all the common data members and methods, now we must declare BachelorStudent and MasterStudent ‘extends’ Student class and delete all common codes from these 2 classes except getRegistrationFee() methods which are different in implementation in each class.
- Now all 3 classes have it’s own getRegistrationFee().

A Simple Example

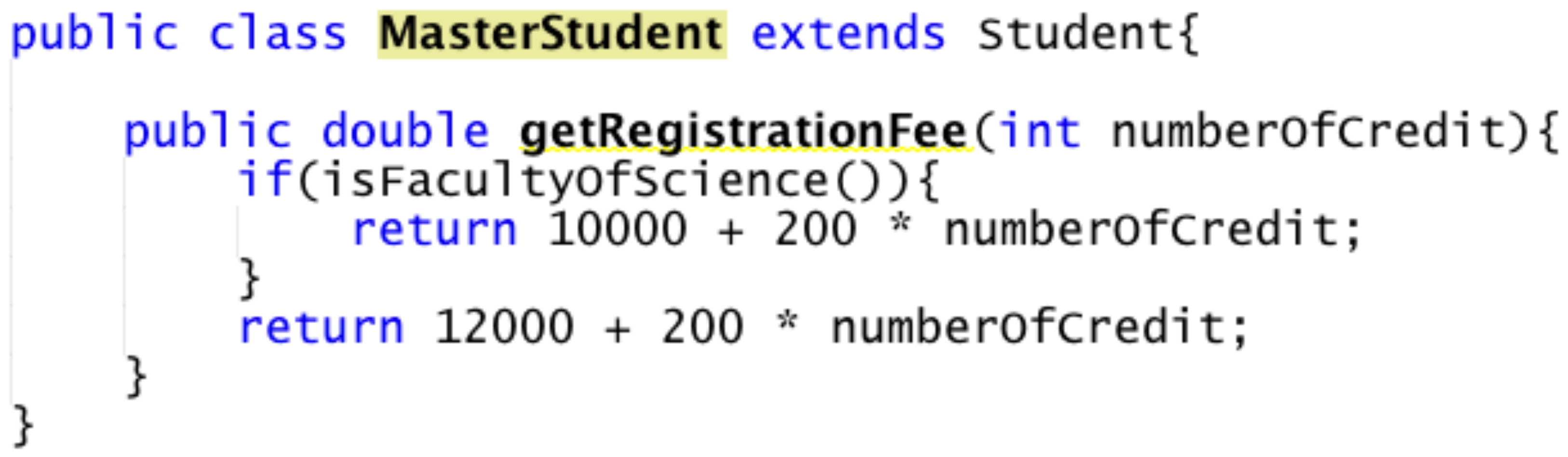
– BachelorStudent class (new)



```
public class BachelorStudent extends Student{
    private boolean scholarshipStudent;

    public boolean isScholarshipStudent() {
        return scholarshipStudent;
    }

    public void setScholarshipStudent(boolean scholarshipStudent) {
        this.scholarshipStudent = scholarshipStudent;
    }

    public double getRegistrationFee(int numberOfCredit){
        if(scholarshipStudent){
            return 0;
        }
        if(isFacultyOfScience()){
            return 5000 + 100 * numberOfCredit;
        }
        return 6000 + 100 * numberOfCredit;
    }
}
```

A Simple Example

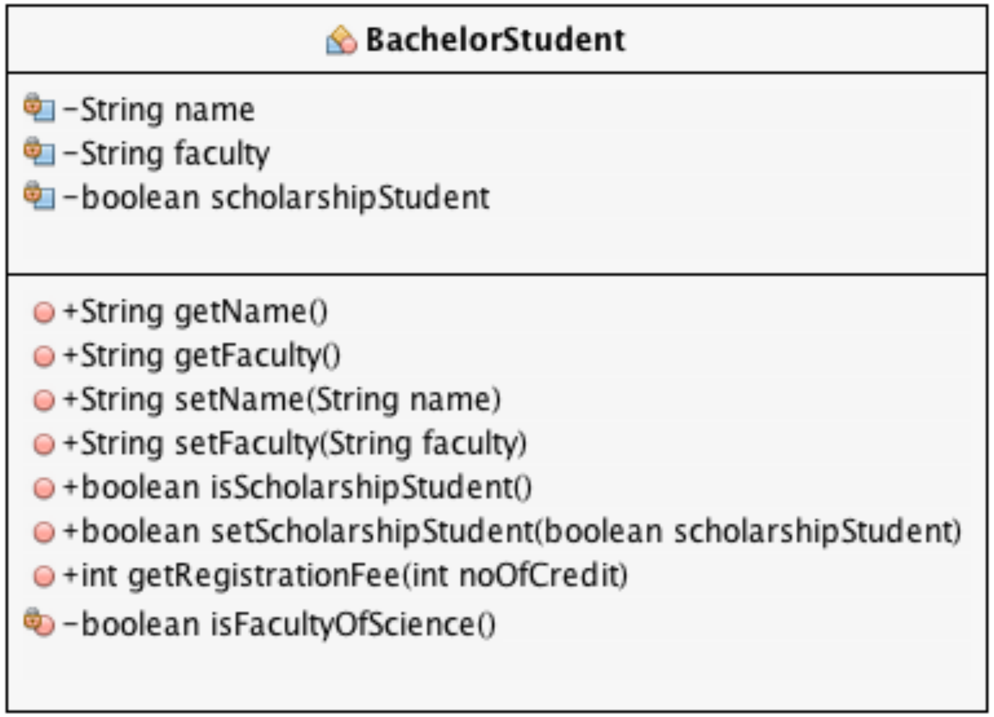
– MasterStudent class (new)

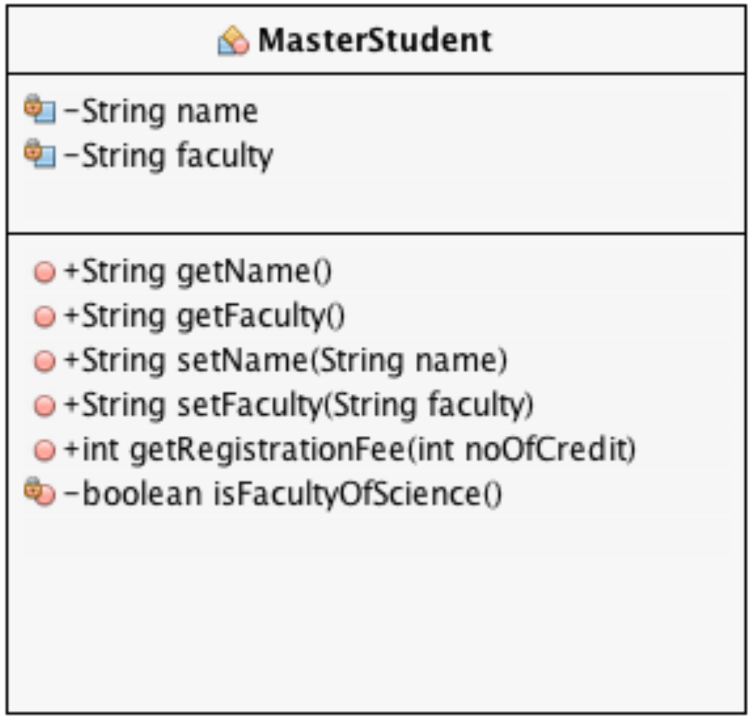


```
public class MasterStudent extends Student{

    public double getRegistrationFee(int numberOfCredit){
        if(isFacultyOfScience()){
            return 10000 + 200 * numberOfCredit;
        }
        return 12000 + 200 * numberOfCredit;
    }
}
```

A Simple Example

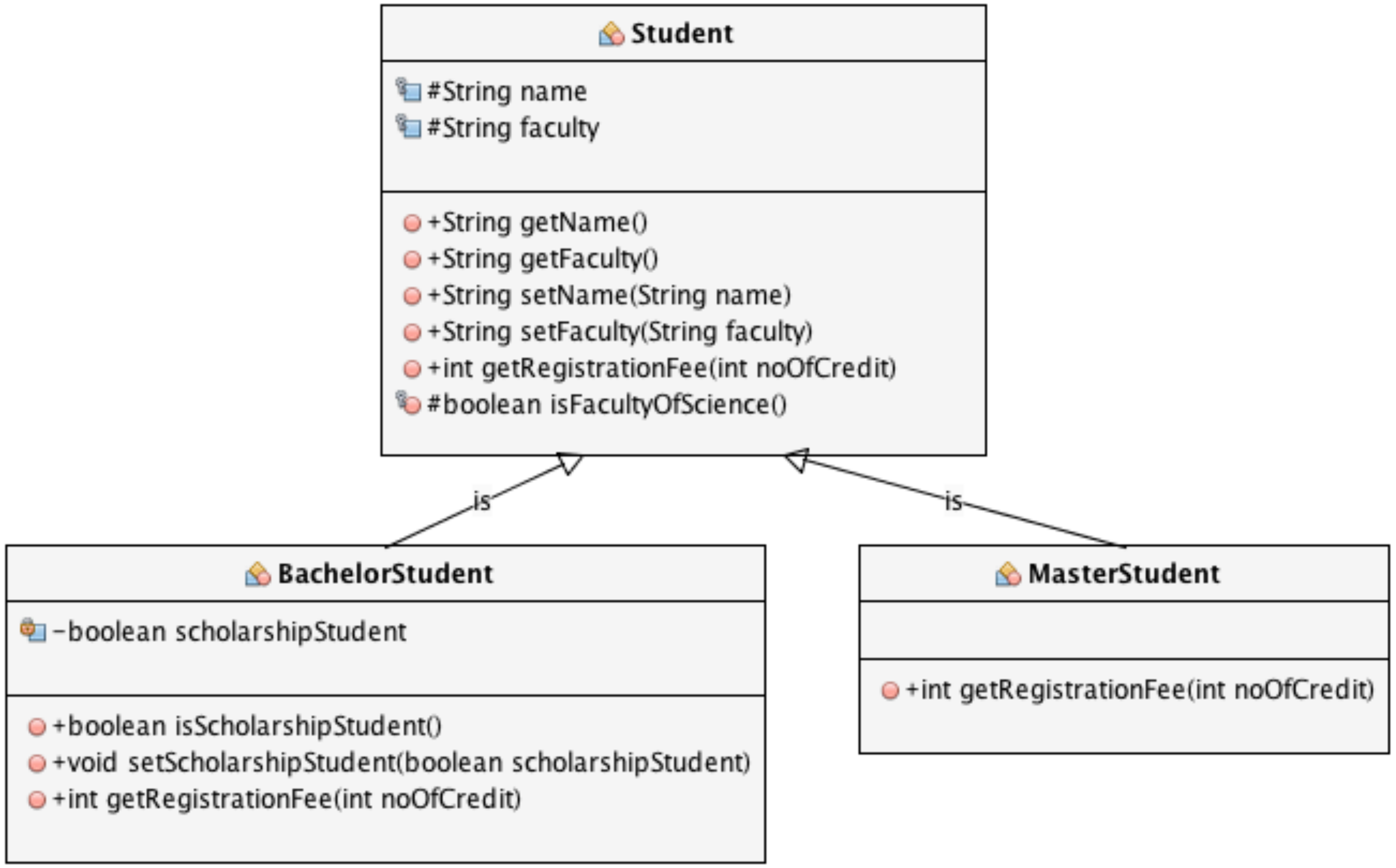
– Class diagram before using inheritance


BachelorStudent
-String name
-String faculty
-boolean scholarshipStudent
+String getName()
+String getFaculty()
+String setName(String name)
+String setFaculty(String faculty)
+boolean isScholarshipStudent()
+boolean setScholarshipStudent(boolean scholarshipStudent)
+int getRegistrationFee(int noOfCredit)
-boolean isFacultyOfScience()


MasterStudent
-String name
-String faculty
+String getName()
+String getFaculty()
+String setName(String name)
+String setFaculty(String faculty)
+int getRegistrationFee(int noOfCredit)
-boolean isFacultyOfScience()

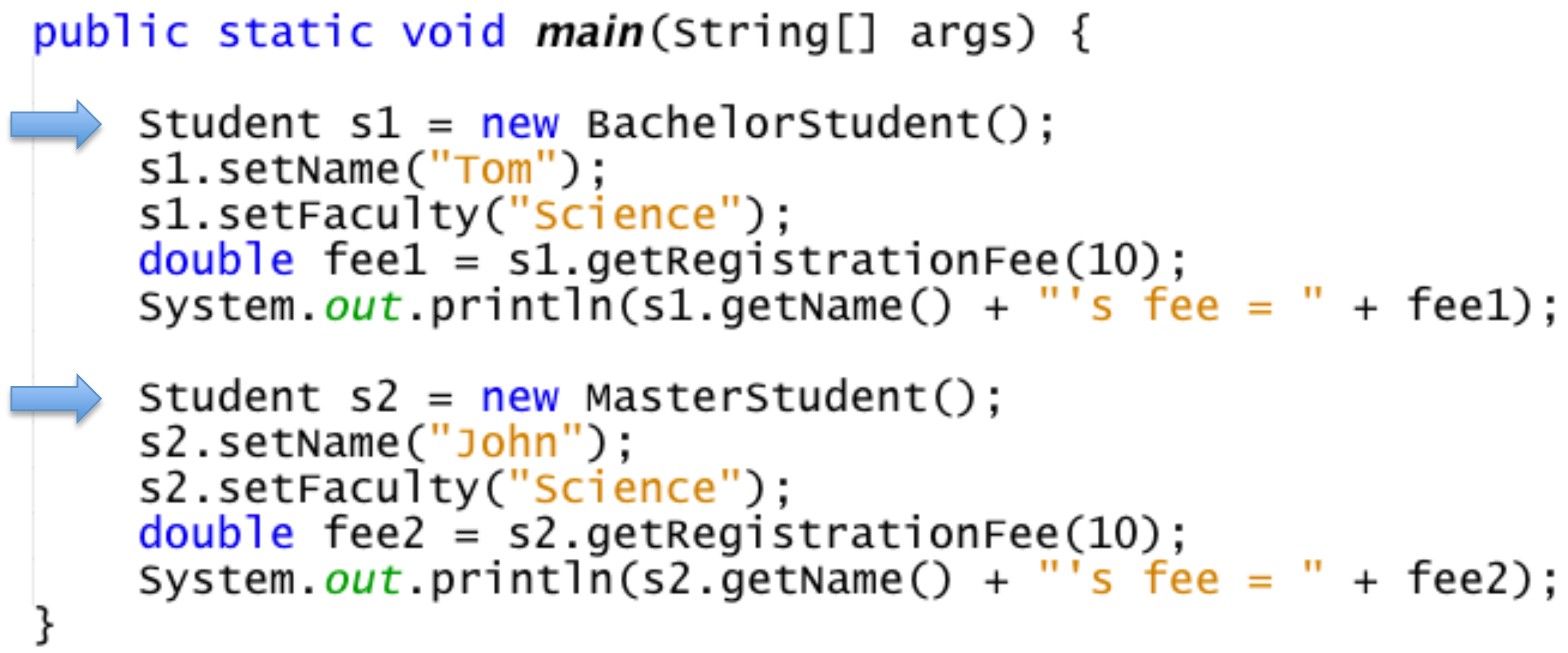
A Simple Example

– Class diagram after using inheritance


Student
#String name
#String faculty
+String getName()
+String getFaculty()
+String setName(String name)
+String setFaculty(String faculty)
+int getRegistrationFee(int noOfCredit)
#boolean isFacultyOfScience()
is
is
BachelorStudent
-boolean scholarshipStudent
+boolean isScholarshipStudent()
+void setScholarshipStudent(boolean scholarshipStudent)
+int getRegistrationFee(int noOfCredit)
MasterStudent
+int getRegistrationFee(int noOfCredit)

A Simple Example

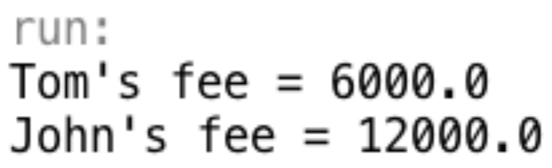
- In main() method, we can use Student as a type of variables reference to BachelorStudent and MasterStudent object



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

    Student s1 = new BachelorStudent();
    s1.setName("Tom");
    s1.setFaculty("Science");
    double fee1 = s1.getRegistrationFee(10);
    System.out.println(s1.getName() + "'s fee = " + fee1);

    Student s2 = new MasterStudent();
    s2.setName("John");
    s2.setFaculty("Science");
    double fee2 = s2.getRegistrationFee(10);
    System.out.println(s2.getName() + "'s fee = " + fee2);
}
```

A Simple Example

– Output



```
run:
Tom's fee = 6000.0
John's fee = 12000.0
```

Access Modifiers

- public (+): A data member or method declared public can be accessed from any other class.
- private (-): A data member or method declared private can only be accessed within the declared class itself.
- protected (#): A data member or method declared protected in a superclass can be accessed within the declared class and by the subclasses of it.

Method Overriding

- Class Student is the superclass and implements a method getRegistrationFee() then it’s subclass called BachelorStudent and MasterStudent inherits every methods from Student class. However, these 2 classes override the method getRegistrationFee(), replacing its functionality from Student class. This is called method overriding.

Polymorphism

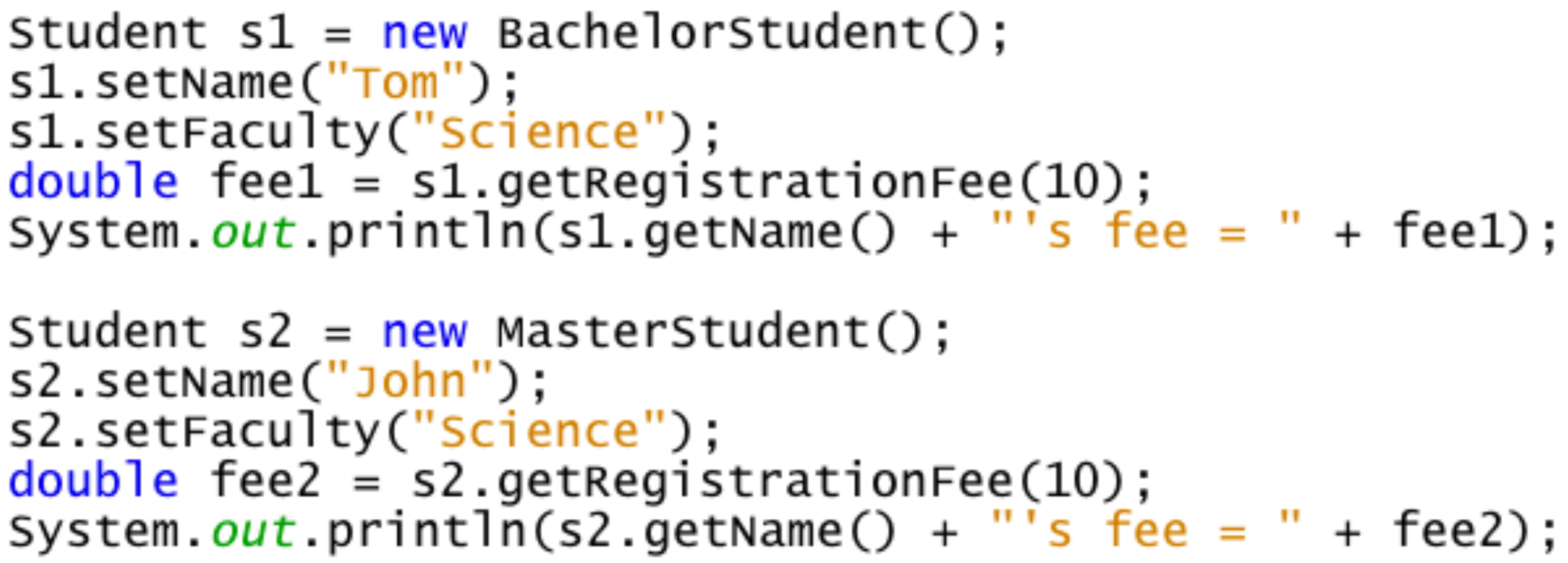
- Polymorphism means “many forms”, it is an OOP’s ability to process objects differently depending on their actual type.
- We must override method inherited from superclass in subclasses in order to use polymorphism ability.

Polymorphism

- Ex. From previous example, BachelorStudent and MasterStudent class override the method getRegistrationFee() by replacing codes to calculate registration fee for each type of student.

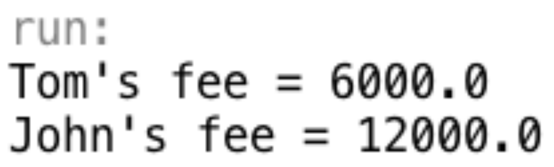
Polymorphism

– main() method



```
Student s1 = new BachelorStudent();
s1.setName("Tom");
s1.setFaculty("Science");
double fee1 = s1.getRegistrationFee(10);
System.out.println(s1.getName() + "'s fee = " + fee1);

Student s2 = new MasterStudent();
s2.setName("John");
s2.setFaculty("Science");
double fee2 = s2.getRegistrationFee(10);
System.out.println(s2.getName() + "'s fee = " + fee2);
```

– Output



```
run:
Tom's fee = 6000.0
John's fee = 12000.0
```

Polymorphism

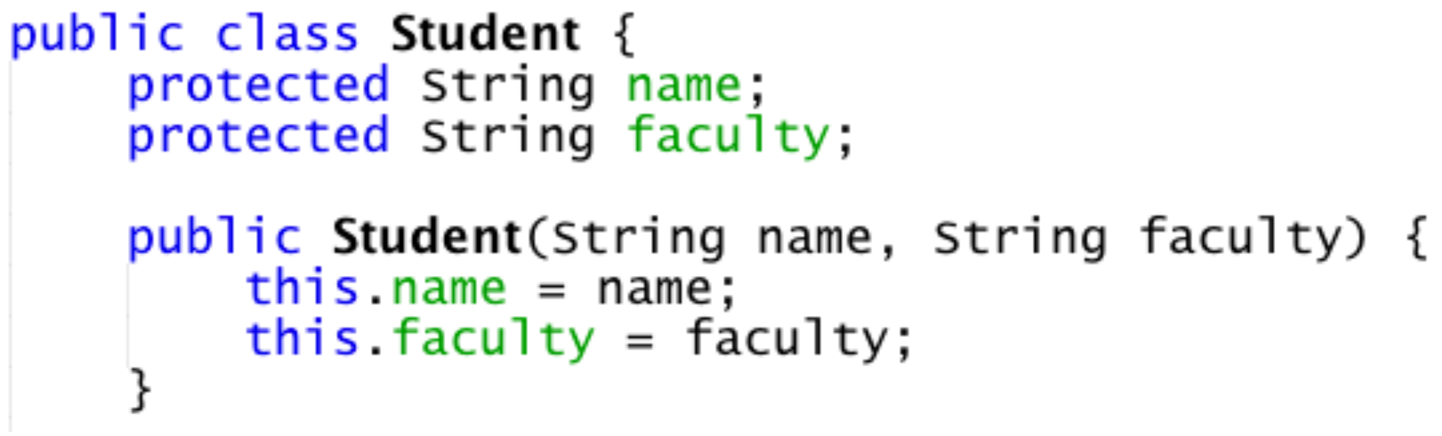
- From codes in main() method, we have variables s1, s2 of type Student, but the real object types which s1 and s2 refer to are objects of BachelorStudent and MasterStudent respectively.
- When we called s1.getRegistrationFee(10); then getRegistrationFee() method in BachelorStudent class is called.
- When we called s2.getRegistrationFee(10); then getRegistrationFee() method in MasterStudent class is called.

Inheritance and Constructors

- If in superclass has non-default constructor, in subclass’ constructor must call superclass’ constructor using super(args..) in the first line.
- If in superclass only has default constructor, we have no need to call super() in subclass’ constructor. It is inserted automatically by the Java compiler.

Inheritance and Constructors

- Ex. From previous example, we can create Student’s constructor like this

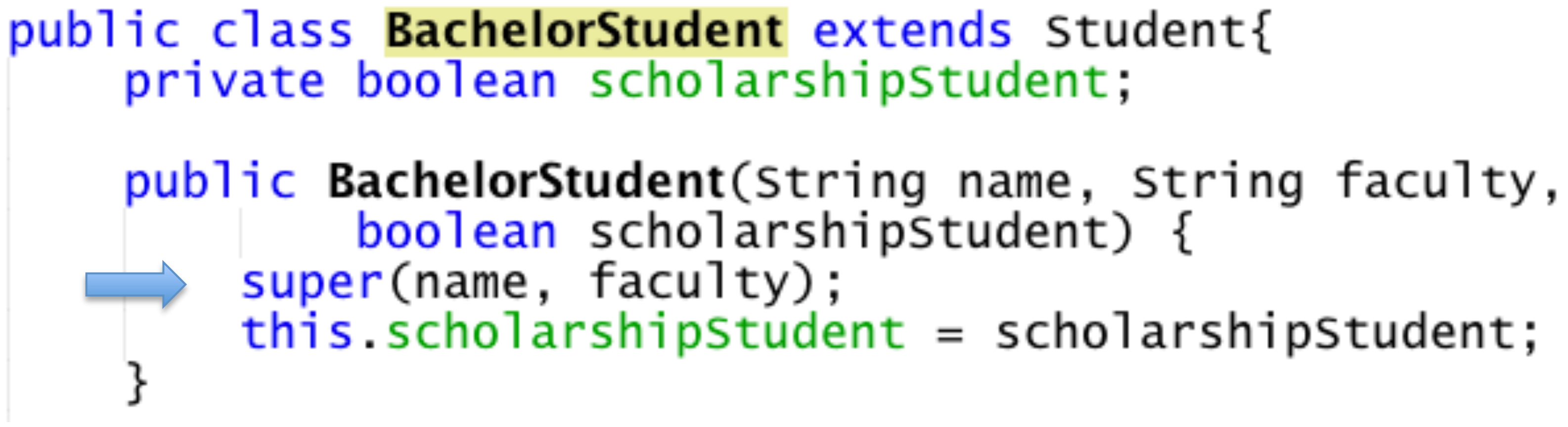


```
public class Student {
    protected String name;
    protected String faculty;

    public Student(String name, String faculty) {
        this.name = name;
        this.faculty = faculty;
    }
```

Inheritance and Constructors

– Then in constructor of BachelorStudent and MasterStudent must call super(name, faculty);

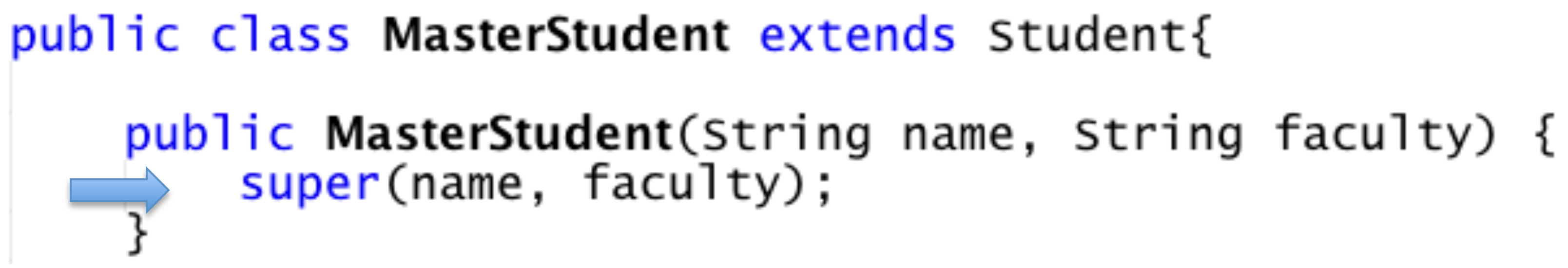
– BachelorStudent ’s constructor



```
public class BachelorStudent extends Student{
    private boolean scholarshipStudent;

    public BachelorStudent(String name, String faculty,
            boolean scholarshipStudent) {
        super(name, faculty);
        this.scholarshipStudent = scholarshipStudent;
    }
```

Inheritance and Constructors

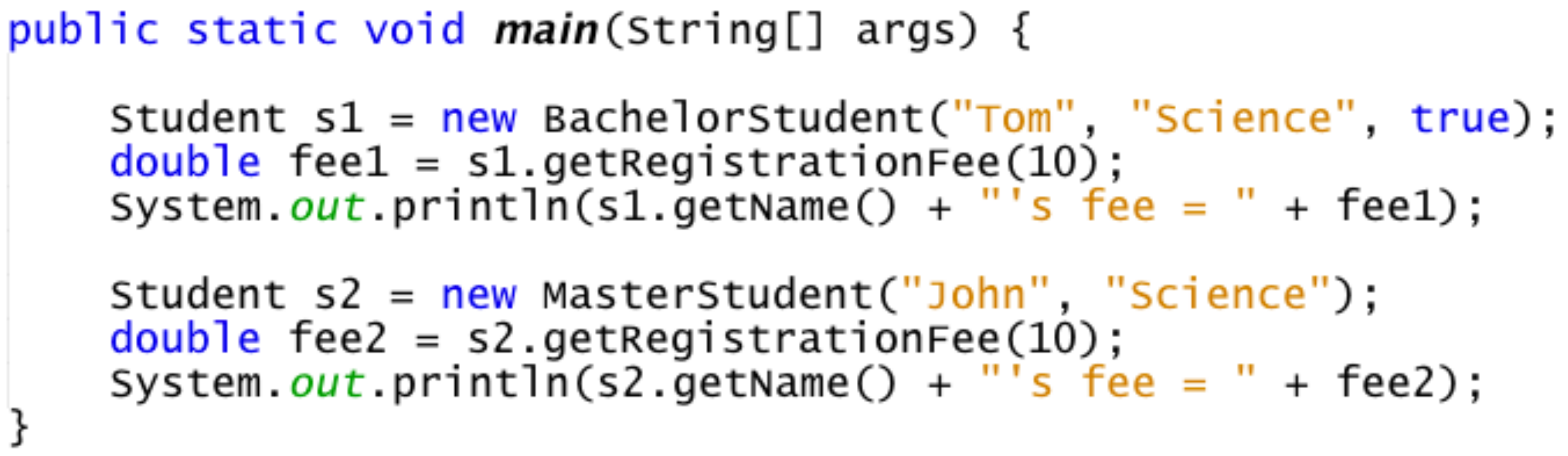
– MasterStudent ’s constructor



```
public class MasterStudent extends Student{

    public MasterStudent(String name, String faculty) {
        super(name, faculty);
    }
```

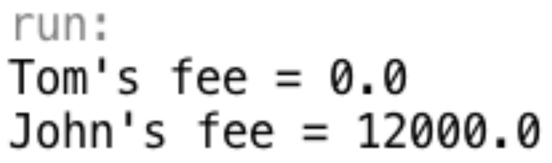
Inheritance and Constructors

– main() method



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

    Student s1 = new BachelorStudent("Tom", "Science", true);
    double fee1 = s1.getRegistrationFee(10);
    System.out.println(s1.getName() + "'s fee = " + fee1);

    Student s2 = new MasterStudent("John", "Science");
    double fee2 = s2.getRegistrationFee(10);
    System.out.println(s2.getName() + "'s fee = " + fee2);
}
```

– Output:



```
run:
Tom's fee = 0.0
John's fee = 12000.0
```

Summary

- Inheritance and polymorphism are powerful OOP features to develop extensible and modifiable code.
- Inheritance mechanism is used to share common code among the related classes.
- The third visibility modifier is the protected modifier.

Summary

- Polymorphism tell us that the method executed will vary according to the class which the actual object belongs.
- The first statement in a constructor of a subclass must be a call to a constructor of the superclass using super(arg..). If we don’t call super(arg..) explicitly, then the statement super() is inserted automatically by the Java compiler.

Reference

- C. Thomas Wu, An Introduction to Object-Oriented Programming with Java, 5th Edition
  - Chapter 13: Inheritance and Polymorphism

Question?