

# 12. Abstraction

29 Oct 2015

# Objectives

- Describe the significance of abstraction in object-oriented programs.
- Define abstract classes, using “abstract” keyword.
- Define abstract methods, using “abstract” keyword.
- Define and implements interface.
- Write programs by using interface and applying polymorphism in program design.

# Abstraction

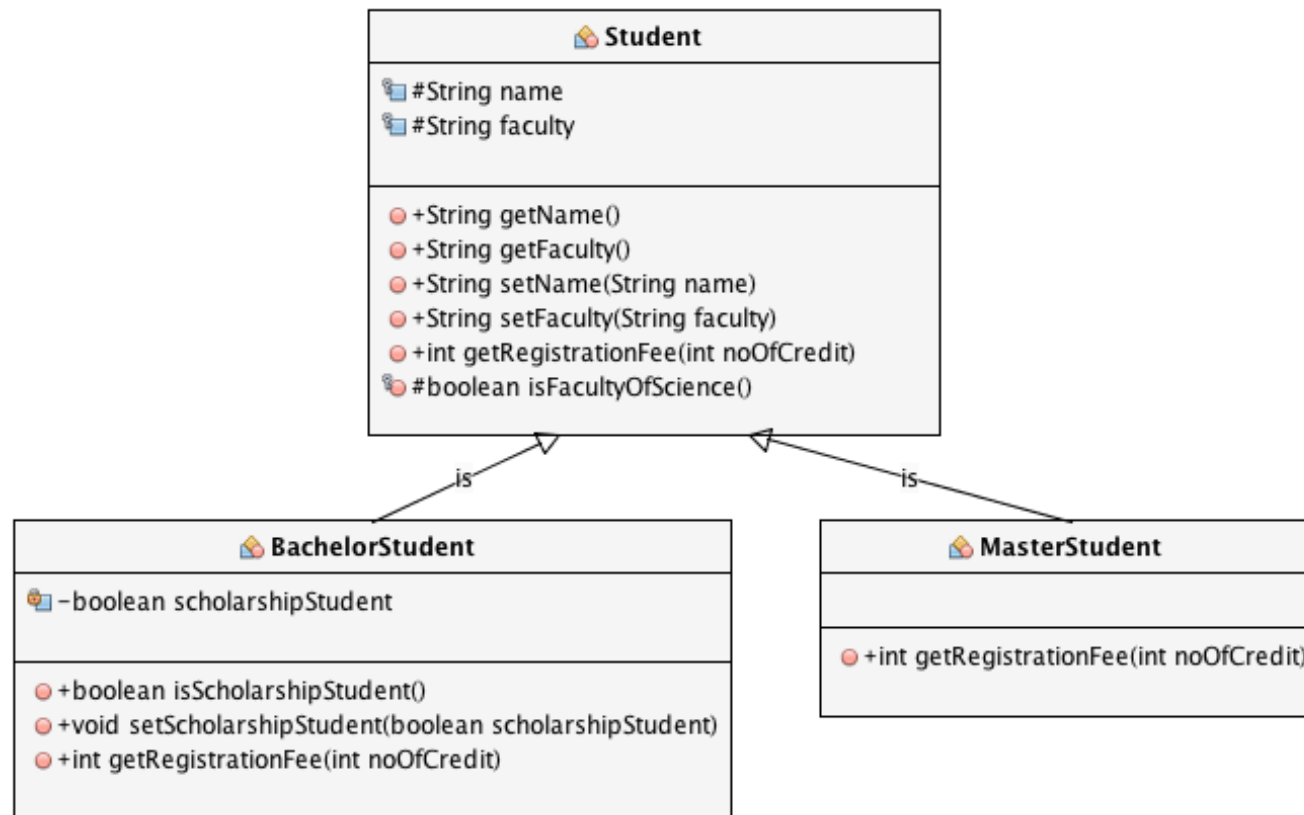
- Abstraction is the concept of describing something in generic terms, i.e. abstracting away the details, in order to focus on what is important. This idea translates to OOP by using an inheritance hierarchy, where more abstract classes are at the top and more concrete classes, at the bottom.

# Abstraction

- From abstraction concept, there are 2 types of classes
  1. Abstract Class (Class at the non-lowest of hierarchy)
  2. Concrete Class (Class at the lowest of hierarchy)

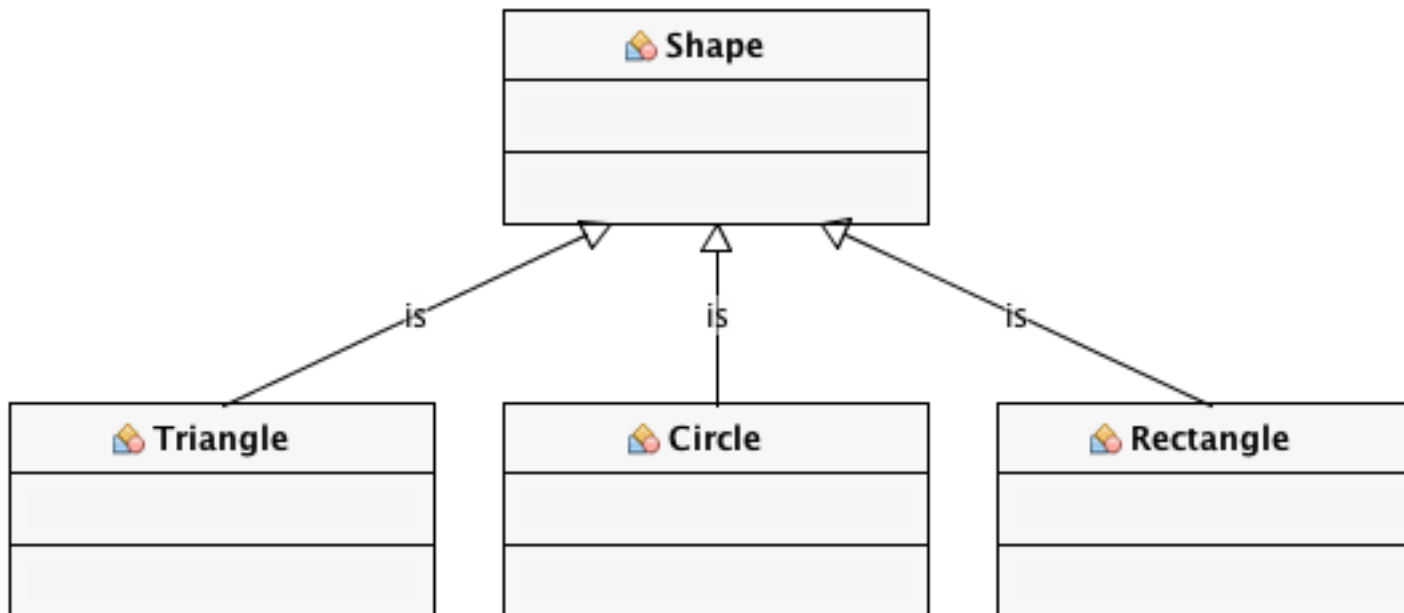
# Abstraction

- From these 2 class hierarchy
  - Student class hierarchy



# Abstraction

- Shape class hierarchy



# Abstraction

1. Abstract Class: An abstract class is one that is designed only as a parent class and from which child classes may be derived, and which is not itself suitable for instantiation (create new object).
  - e.g.1 Student class
  - e.g.2 Shape class

# Abstraction

2. Concrete Class: A concrete class is a class at the lowest of class hierarchy for which instances may be created. This contrasts with abstract classes.
  - e.g.1 BachelorStudent and MasterStudent class
  - e.g.2 Rectangle, Triangle and Circle class

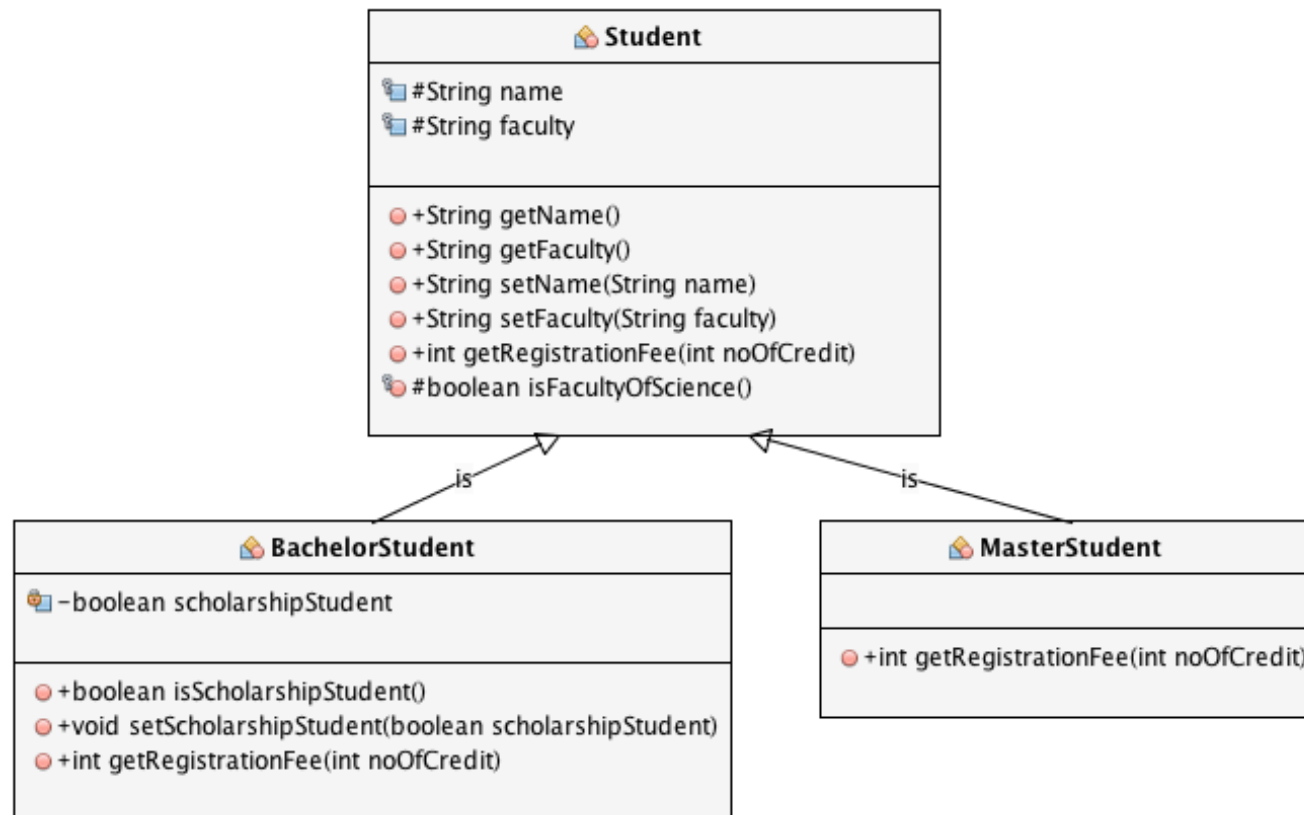


# Abstract Class

- In Java, an abstract class is a class that is declared abstract. Abstract classes cannot be instantiated, but they can be subclassed.
- In Java, we use “abstract” keyword to declare that class is abstract then we can't create new object from that class.

# Abstract Class

- Ex. From Student class hierarchy



# Abstract Class

– Student class

```
public class Student {  
    protected String name;  
    protected String faculty;  
  
    public String getName() {  
        return name;  
    }  
  
    public String getFaculty() {  
        return faculty;  
    }  
}
```

# Abstract Class

```
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;  
}  
}
```

# Abstract Class

## – BachelorStudent class

```
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;
    }
}
```

# Abstract Class

– MasterStudent class

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

# Abstract Class

- We can use Student as a generic variable data type to refer 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);  
}
```

# Abstract Class

– Output (Correct):

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



# Abstract Class

- But we still can use Student as a variable data type to refer to Student object, which generate incorrect output.

```
public static void main(String[] args) {  
    → Student s1 = new Student("Tom", "Science");  
    double fee1 = s1.getRegistrationFee(10);  
    System.out.println(s1.getName() + "'s fee = " + fee1);  
    → Student s2 = new Student("John", "Science");  
    double fee2 = s2.getRegistrationFee(10);  
    System.out.println(s2.getName() + "'s fee = " + fee2);  
}
```

# Abstract Class

– Output (Incorrect):

```
run:  
Tom's fee = -1.0  
John's fee = -1.0
```

# Abstract Class

- Now we declare the Student class to be an abstract.



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

# Abstract Class

- Then if we try to create object from Student class, compiler will generate an error.

```
public static void main(String[] args) {  
    Student s1 = new Student("Tom", "Science");    //compile error  
    Student s2 = new Student("John", "Science");    //compile error  
    ...  
}
```


# Abstract Method

- In Java, an abstract method is a method that is declared abstract without an implementation (without braces, and followed by a semicolon)
- In Java, we use “abstract” keyword to declare the method in superclass abstract, then we have no need to implement that method in superclass and all its subclass must implement the method’s detail.

# Abstract Method

- After we declare the Student class to be an abstract, there is getRegistrationFee() method that return -1 and it useless.

```
public abstract class Student {  
    protected String name;  
    protected String faculty;  
    ...  
    public double getRegistrationFee(int numberOfCredit){  
        return -1;  
    }  
    ...  
}
```



# Abstract Method

- If we forget to override `getRegistrationFee()` method in `BachelorStudent` and `MasterStudent` class, the `getRegistrationFee()` method in `Student` class will be used.

# Abstract Method

- From code below, if in BachelorStudent and MasterStudent class have no overridden getRegistrationFee() method, it will make incorrect output

```
public static void main(String[] args) {  
    Student s1 = new BachelorStudent("Tom", "Science", false);  
    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);  
}
```



# Abstract Method


– Output (Incorrect):

```
run:  
Tom's fee = -1.0  
John's fee = -1.0
```

# Abstract Method

- Now we declare the getRegistrationFee() method in Student class to be an abstract method and remove its body.

```
public abstract class Student {  
    protected String name;  
    protected String faculty;  
  
    public abstract double getRegistrationFee(int numberOfCredit);  
    ...  
}
```



# Abstract Method

- Then in BachelorStudent and MasterStudent class will be forced to implement getRegistrationFee() method. If they are not implement getRegistrationFee() method, compiler will generate an error.

# Abstract Method

- BachelorStudent class: If there is no getRegistrationFee() method, compiler will generate an error.

```
public class BachelorStudent extends Student{  
    private boolean scholarshipStudent;  
  
    public double getRegistrationFee(int numberOfCredit){  
        if(scholarshipStudent){  
            return 0;  
        }  
        if(isFacultyOfScience()){  
            return 5000 + 100 * numberOfCredit;  
        }  
        return 6000 + 100 * numberOfCredit;  
    }  
    ...  
}
```

# Abstract Method

- MasterStudent class: If there is no getRegistrationFee() method, compiler will generate an error.

```
public class MasterStudent extends Student{  
    → public double getRegistrationFee(int numberOfCredit){  
        if(isFacultyOfScience()){  
            return 10000 + 200 * numberOfCredit;  
        }  
        return 12000 + 200 * numberOfCredit;  
    }  
    ...  
}
```

# Abstract Class & Abstract Method

- When we declared abstract class, it may or not may have abstract method inside abstract class.
- But when we declared abstract method, the class that contain abstract method must be abstract class.

# Interface

- An interface in Java is similar to a class, but the body of an interface can include only abstract-public methods and public-static-final data members. A class implements an interface by providing code for each method declared by the interface.
- We can use interface name as a variable data type like superclass name.

# Interface

- We use interface to force any classes which not in the same class hierarchy to have common method in order to use polymorphism ability.
- To create interface, we use “interface” keyword instead of “class” keyword.
- To implements interface, we use “implements” keyword.



# Interface

- Ex. Displayable interface

```
public interface Displayable {  
    public static final String CONSTANT_STRING1 = "Message is: ";  
    public abstract void display();  
}
```

# Interface


- When we declared BachelorStudent class implements Displayable interface, then BachelorStudent class must have display() method and it can use constant variable declared in Displayable interface

```
public class BachelorStudent extends Student implements Displayable{  
    ...  
    public void display() {  
        System.out.println(CONSTANT_STRING1 + name + ", " + faculty);  
    }  
    ...  
}
```

# Interface

- When we declared Subject class implements Displayable interface, then Subject class must have display() method and it can use constant variable declared in Displayable interface

```
public class Subject implements Displayable{  
    private String id;  
    private String name;  
  
    public Subject(String id, String name) {  
        this.name = name;  
        this.id = id;  
    }  
  
    public void display() {  
        System.out.println(CONSTANT_STRING1 + id + " " + name);  
    }  
}
```



# Interface

- In main() method, we can use Displayable interface as data type like Student class

```
public static void main(String[] args) {  
    Student s1 = new BachelorStudent("Tom", "Science", false);  
    Subject subj1 = new Subject("sc101", "Basic Java");  
  
    → Displayable d1 = (Displayable) s1;  
       d1.display();  
  
    → Displayable d2 = subj1;  
       d2.display();  
}
```

# Interface

– Output:

```
run:  
Message is: Tom, Science  
Message is: sc101 Basic Java
```

# Summary

- From abstraction concept, there are 2 types of classes – abstract class and concrete class.
- In Java, we define abstract class and abstract method by using “abstract” keyword.
- Abstract class is the class that is designed only as a parent class and cannot be instantiated.

# Summary

- Abstract method is a method that is declared abstract without an implementation.
- Interface in Java is similar to a class, but an interface can include only abstract-public methods and public-static-final data members.

# Reference

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



Question?