WEEK 14

Abstract Class and Interface

Outline

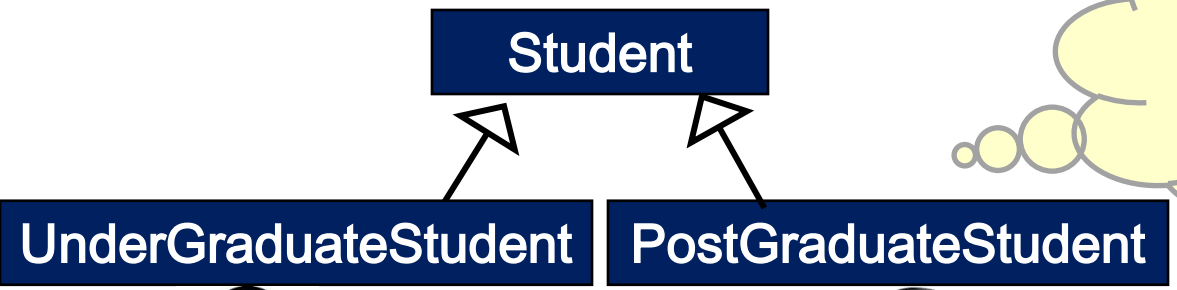
- Abstract Class and Abstract Methods
- Interface Definition and Inheritance versus Interface

Learning Objectives

- To define abstract class and methods
- To define interface
- To differentiate between abstract class and interface
- To write programs that are easily extensible and modifiable by applying an integration of polymorphism and abstract class concepts

Introduction

- Sometimes it is desirable to force programmers to redefine methods of the parent class, when:
  - there is no good default for the parent class
  - only child class programmer can know the implementation of the method.
- Usually we do not need to create any instances of the parent class.
- Consider these classes:



We create instances for UnderGraduateStudent and PostGraduateStudent, but not for Student

What is Abstract Class?

- We define a class to be abstract if we do not allow any object to be created from it.
- An abstract class is a class that is declared with the reserved word `abstract` in its heading.
- We cannot instantiate an object of an abstract class as what we commonly do for a concrete class.
- An abstract class can contain everything that a concrete class can have. Additionally, abstract class can also contain abstract methods.

What is Abstract Method?

- A method that has only the header with no body (implementation).
- Its header contains the reserved word `abstract` and ends with semicolon(;).

- Syntax:

```
abstract <return_type> method_name();
```

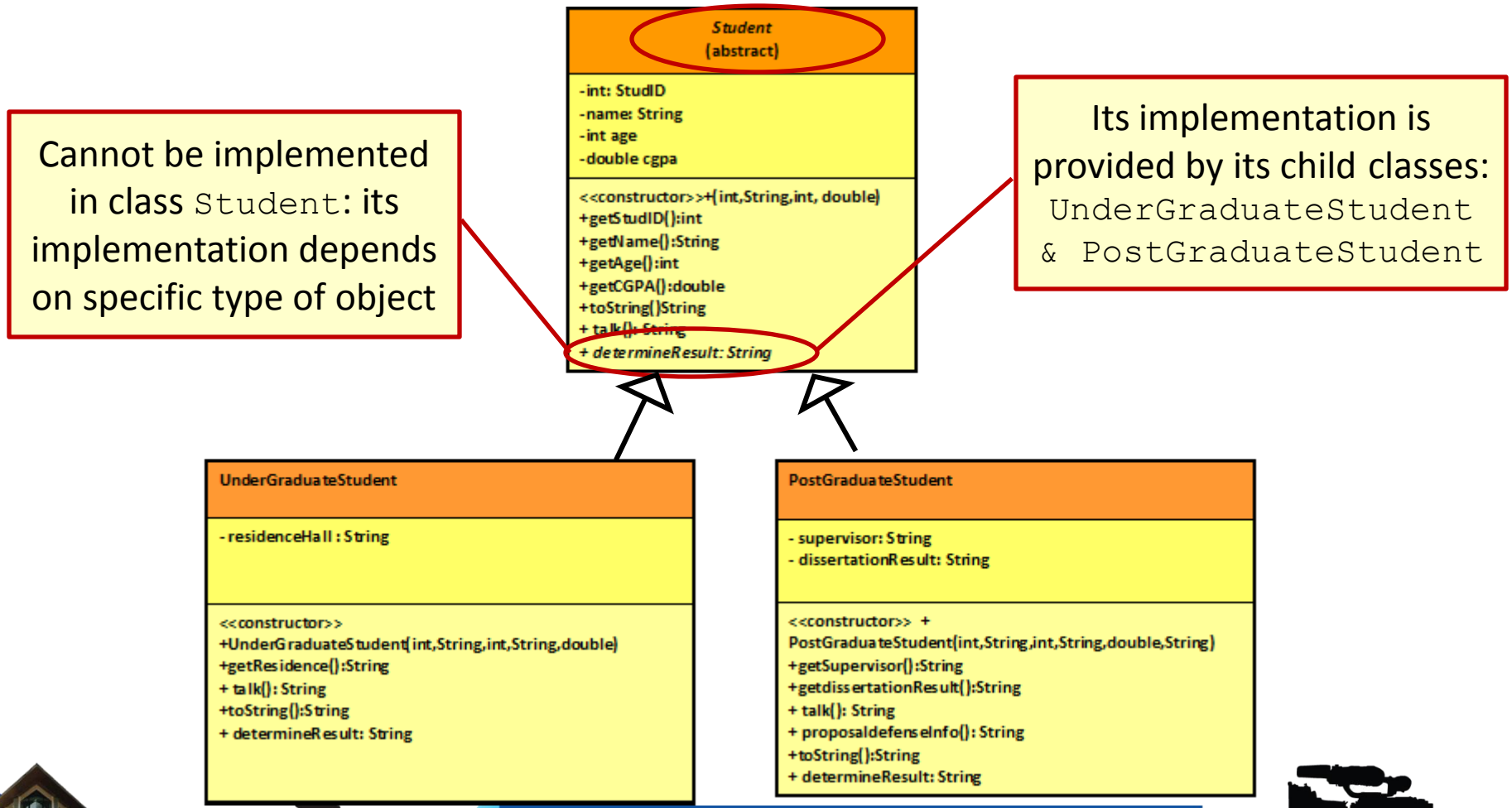
- Example:

```
abstract double computeScore();
```

What is Abstract Method?

- Abstract method cannot be private type because private members cannot be accessed.
- Abstract method also cannot be final because final method cannot be overridden.
- Constructor and static method cannot be used as abstract method.

Example of abstract class & methods



First we create the parent class: **`Student`**

```
public abstract class Student {

    private int studID, age;
    private String name;
    private double cgpa;

    abstract String determineResult();

    public Student(int studID, String name, int age, double cgpa) {
        this.studID = studID;
        this.name = name;
        this.age = age;
        this.cgpa = cgpa;
    }

    public int getstudID() {
        return studID;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
```

`Student` Class (cont)

```
public double getCGPA() {
    return cgpa;
}

public String toString() {
    return "Stud ID: " + studID + "\nName: " + name + "\nAge: " + age
            + "\nCGPA: " + cgpa;
}

public String talk() {
    return "I am a student";
}
```

Then we create the child class: `UnderGraduateStudent`

```
public class UnderGraduateStudent extends Student {

    private String residenceHall;

    public UnderGraduateStudent(int studID, String name, int age,
            String residenceHall, double cgpa) {
        super(studID, name, age, cgpa);
        this.residenceHall = residenceHall;
    }

    public String getResidence() {
        return residenceHall;
    }

    @Override
    public String talk() {
        return "I am an undergraduate student";
    }

    @Override
    public String toString() {
        return "Information about the undergraduate student:\n"
                + super.toString() + "\nResidence Hall: " + residenceHall;
    }
```

`UnderGraduateStudent` Class (cont)

```
@Override
public String determineResult() {
    if (getCGPA() >= 3.5) {
        return getName() + " will get the Dean List Award";
    } else {
        return getName() + " will not get the Dean List Award";
    }
}
```

Then we create the child class: `PostGraduateStudent`

```
public class PostGraduateStudent extends Student {

    private String supervisor, dissertationResult;

    public PostGraduateStudent(int studID, String name, int age,
            String supervisor, double cgpa, String dissertationResult) {
        super(studID, name, age, cgpa);
        this.supervisor = supervisor;
        this.dissertationResult = dissertationResult;
    }

    public String getSupervisor() {
        return supervisor;
    }

    public String getdissertationResult() {
        return dissertationResult;
    }

    @Override
    public String talk() {
        return "I am a postgraduate student";
    }
```

`PostGraduateStudent` Class (cont)

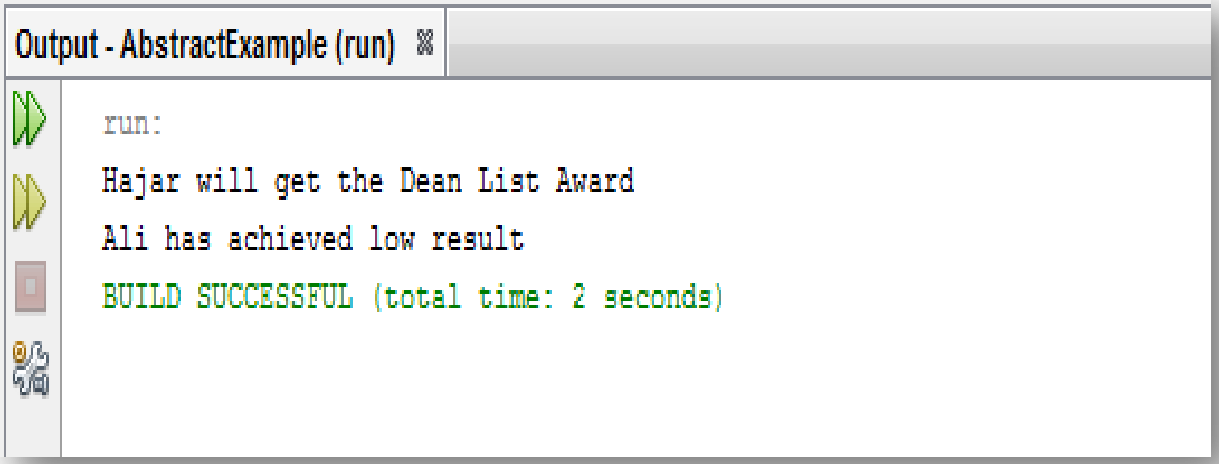
```
public String proposalDefenseInfo(String ans) {
    if (ans.equals("Yes")) {
        return "I have defended my proposal;";
    } else {
        return "I have not defend my proposal";
    }
}

@Override
public String toString() {
    return "Information about the postgraduate student:\n"
            + super.toString() + "\nDissertation result: "
            + dissertationResult + "\nSupervisor: " + supervisor;
}

@Override
public String determineResult() {
    if ((getCGPA() >= 3.70) && (dissertationResult.
            equals("Pass with minor correction"))) {
        return "This student student has achieved excellent result";
    } else if (((getCGPA() > 3.0) && (getCGPA() < 3.7))
            && (dissertationResult.equals("Pass with major correction"))) {
        return getName() + " has achieved average result ";
    } else {
        return getName() + " has achieved low result";
    }
}
```

Next we create a test program: `TestPolymorphicAbstract`

```
public class TestPolymorphicAbstract {

    public static void main(String args[]) {
        Student s1;
        s1 = new UnderGraduateStudent(1988, "Hajar", 18, "TNB COllege", 3.89);
        System.out.println(s1.determineResult());

        s1 = new PostGraduateStudent(98887, "Ali", 38, "Ida", 3.0,
                "Pass with major correction");
        System.out.println(s1.determineResult());
    }
}
```

Polymorphic reference is used to create objects from different classes

Output:


Output - AbstractExample (run)
run:
Hajar will get the Dean List Award
Ali has achieved low result
BUILD SUCCESSFUL (total time: 2 seconds)

What is Interface?

- An interface type is similar to a class, but only contains constants (if any) and abstract methods
- All methods in an interface type are automatically public.
- An interface cannot be instantiated, it can only be implemented by classes.
- Interface is used to support multiple inheritance in Java.

The reserved word **`interface`** is used for defining an interface in Java

Syntax :

```
public interface <InterfaceName>
{
  <constant declarations>
  <method signatures>
}
```

OR

```
public interface <InterfaceName>
{
  <constant declarations>
  abstract <method signatures>
}
```

Example:

```
public interface Responsible {
    String getResponsibility();
}
```

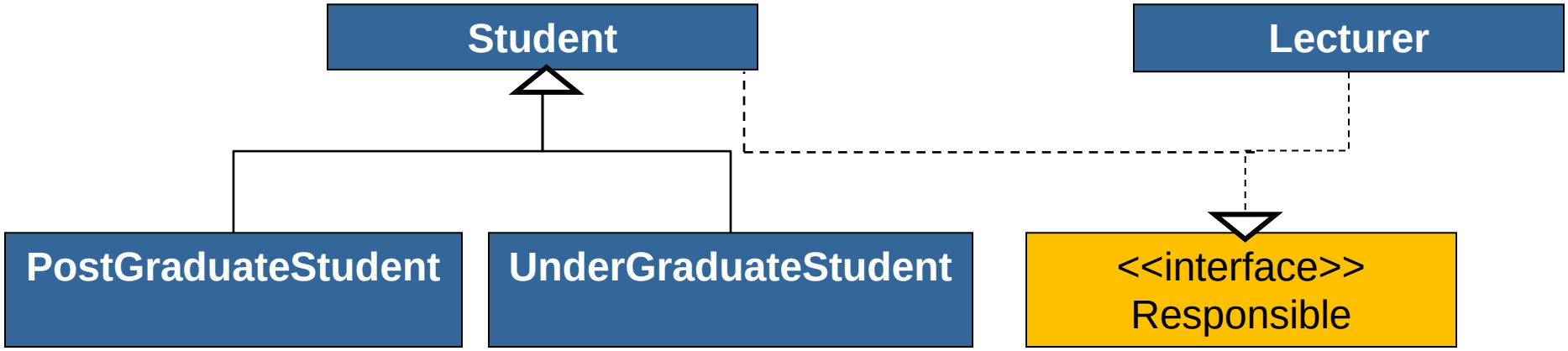
The reserved word `implements` is used for implementing an interface in Java

Syntax:

```
public <className> implements <InterfaceName1,…,<InterfaceNameN> {
        <override all methods in interface>
   }
```

If a class implements an interface, it must override all of the abstracts methods that exist in the interface

Example of Interface



- Both `Student` class and `Lecturer` class implement Responsible interface.
- A *dotted* arrow with triangular tip denotes the "is-a" relationship between a class and an interface.
- Both `PostGraduateStudent` and `UnderGraduateClass` are child for `Student`.
- A *solid* arrow with triangular tip denotes the "is-a" relationship between a parent and a child.

First we create the **`Responsible`** Interface

```
public interface Responsible {
    String getResponsibility();
}
```

Next we create class **`Student`** that implements the **`Responsible`** interface

```
public abstract class Student implements Responsible {

    private int studID, age;
    private String name;
    private double cgpa;

    abstract public String determineResult();

    public Student(int studID, String name, int age, double cgpa) {
        this.studID = studID;
        this.name = name;
        this.age = age;
        this.cgpa = cgpa;
    }

    public int getstudID() {
        return studID;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
```

Student Class (cont)

```
    public double getCGPA() {
        return cgpa;
    }

    public String toString() {
        return "Stud ID: " + studID + "\nName: " + name + "\nAge: "
                + age + "\nCGPA: " + cgpa;
    }

    public String talk() {
        return "I am a student";
    }

    public String getResponsibility() {
        return "My responsibility is to study and pass the exam";
    }
}
```

Then we create another class **`Lecturer`** that also implements the **`Responsible`** interface

```
public class Lecturer implements Responsible {

    private int staffID;
    private String name, department, areaofExpertise, subjectsTaught;

    Lecturer(int staffID, String name, String department,
            String areaofExpertise, String subjectsTaught) {
        this.staffID = staffID;
        this.name = name;
        this.department = department;
        this.areaofExpertise = areaofExpertise;
        this.subjectsTaught = subjectsTaught;
    }

    public int getStaffID() {
        return staffID;
    }

    public String getName() {
        return name;
    }

    public String getDepartment() {
        return department;
    }
```

Lecturer Class (cont)

```
public String getareaofExpertise() {
    return areaofExpertise;
}

public String getsubjectsTaught() {
    return subjectsTaught;
}

public String talk() {
    return "I am a lecturer";
}

public String getResponsibility() {
    return "My responsibility is to teach students and ensure they "
            + "understand what is being taught";
}
```

Next we create a test program: `TestInterface`

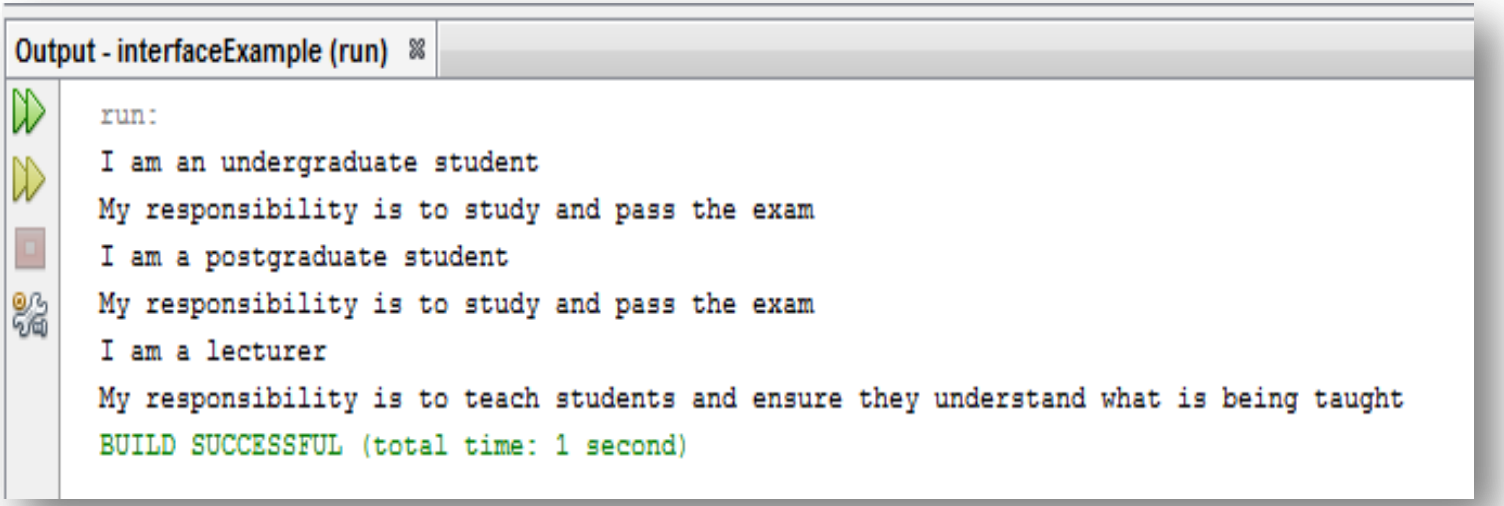
```
public class TestInterface {

    public static void main(String args[]) {
        Student us1 = new UnderGraduateStudent(60812, "Aisyah", 19,
                "TM College", 3.4);
        System.out.println(us1.talk());
        System.out.println(us1.getResponsibility());

        us1 = new PostGraduateStudent(93047, "Siti", 30, "Ayu", 3.2,
                "Pass with major correction");
        System.out.println(us1.talk());
        System.out.println(us1.getResponsibility());

        Lecturer lec1 = new Lecturer(3432, "Suhana", "SOC",
                "Software Engineering", "Programming 1, Data Structure");
        System.out.println(lec1.talk());
        System.out.println(lec1.getResponsibility());
    }
}
```

Method `getResponsibility()` in the `Student` class is executed

Method `getResponsibility()` in the `Student` class is executed

Method `getResponsibility()` in the `Lecturer` class is executed

Output:


Output - interfaceExample (run)
run:
I am an undergraduate student
My responsibility is to study and pass the exam
I am a postgraduate student
My responsibility is to study and pass the exam
I am a lecturer
My responsibility is to teach students and ensure they understand what is being taught
BUILD SUCCESSFUL (total time: 1 second)

Summary

- Abstract method is a method that has only the heading with no implementation
- A class must be declared as abstract when it contains at least one abstract method.
- An interface is similar to a class, but only contains constants (if any), abstract methods or method signature.
- Interface is used to support multiple inheritance.