# JAVA PROGRAMMING II

Abstract Class

CPCS 203

## Abstract Superclasses and Abstract Methods

- Abstract classes are like regular classes with data and methods, **but you cannot create instances (objects) of abstract classes using the new operator**.
- An abstract method cannot be placed in a non-abstract class.
- If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be declared abstract.

## Abstract Superclasses and Abstract Methods

- A classes that contains abstract methods must be abstract. However, it is possible to declare an abstract class that contains no abstract methods.
- A subclass can be abstract even if its superclass is concrete.

## Abstract Superclasses and Abstract Methods

- When we define a superclass, we often do not need to create any instances of the superclass.
- Depending on whether we need to create instances (objects) of the superclass, we must define the class differently.
- We will study examples based on the `Student` superclass defined earlier.

## Abstract Superclasses and Abstract Methods

- Example: A Student Must Be Undergraduate or Graduate
  - If a student must be either an undergraduate or a graduate student, we only need instances (objects) of UndergraduateStudent or GraduateStudent.
  - Therefore, we must define the Student class so that no instances (objects) may be created of it.

## Abstract Superclasses and Abstract Methods

- **An abstract class is a class defined with the modifier *abstract*. No instances can be created from an abstract class.**

## Abstract Superclasses and Abstract Methods

```
abstract class Student {
    protected final static int NUM_OF_TESTS = 3;
    protected String name;
    protected int[] test;
    protected String courseGrade;

    public Student() {
        this(“No name”);
    }

    public Student(String studentName) {
        name = studentName;
        test = new int[NUM_OF_TESTS];
        courseGrade = “******”;
    }

    abstract public void computeCourseGrade();
```

**Note: If a subclass of this abstract superclass does not implement this abstract methods, the subclass must be declared abstract.**

# Abstract Superclasses and Abstract Methods

```
    public String getCourseGrade() {
            return courseGrade;
    }
    public String getName() {
            return name;
    }
    public int getTestScore(int testNumber) {
            return test[testNumber-1];
    }
    public void setName(String newName) {
            name = newName;
    }
    public void setTestScore(int testNumber, int testScore) {
            test[testNumber-1] = testScore;
    }
}
```

## Abstract Superclasses and Abstract Methods

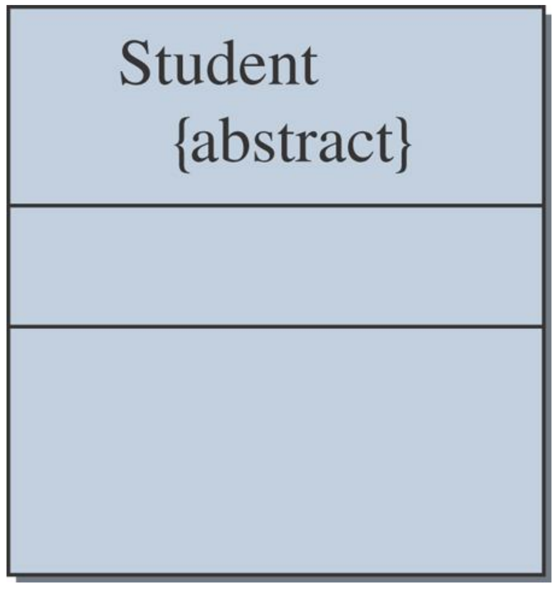
- An abstract method is a method with the keyword **abstract**, and it ends with a semicolon instead of a method body.
- A class is abstract if the class contains an abstract method or does not provide an implementation of an inherited abstract method.

## Abstract Superclasses and Abstract Methods

- We say a method is implemented (concrete) if it has a method body.
- If a subclass has no abstract methods and no unimplemented inherited abstract methods, then the subclass is no longer abstract, and instances (objects) may be created of it.
- **An abstract class must contain the keyword *abstract* in its definition**.

# Abstract Superclasses and Abstract Methods

- In a program diagram, we represent an abstract class by using the keyword **abstract**.



## Abstract Superclasses and Abstract Methods

- Example: Student Does Not Have to Be Undergraduate or Graduate.
- In this case, we may design the Student class in one of two ways.
  - We can make the Student class instantiable (able to create an object) OR
  - We can leave the Student class abstract and add a third subclass, OtherStudent, to handle a student who does not fall into the UndergraduateStudent or GraduateStudent categories.

## Abstract Superclasses and Abstract Methods

- With the first approach, we delete the keyword abstract from the class and method definition. We provide a method body for computeCourseGrade.

```
class Student {
  ...

  public void computeCourseGrade(){
      int total = 0;
      for (int i = 0; i < NUM_OF_TESTS;  i++){
           total += test[i];
      }
```

## Abstract Superclasses and Abstract Methods

```
      if (total/NUM_OF_TESTS >= 50){
           courseGrade = “Pass”;
      }else{
           courseGrade = “No Pass”;
      }
   }
...
}
```

- This design allows us to create an instance of Student to represent a non-regular student.

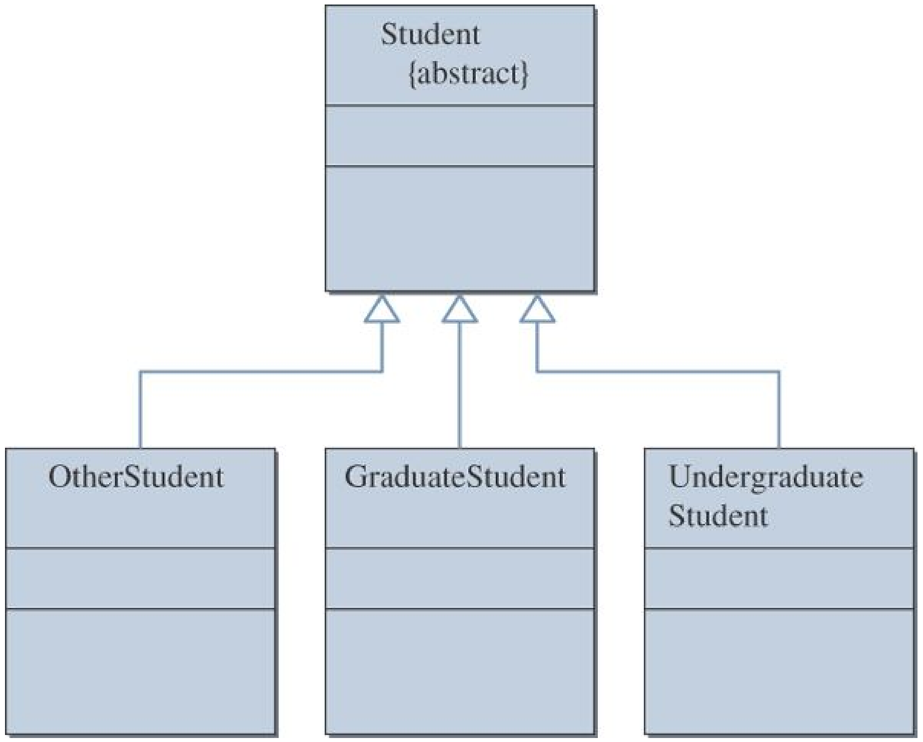
# Abstract Superclasses and Abstract Methods

- In the second approach, we leave the Student class abstract. We define a third subclass, OtherStudent (better approach):

```
class OtherStudent extends Student {
   public void computeCourseGrade() {
        int total = 0;
        for (int i=0;  i < NUM_OF_TESTS;  i++){
               total += test[i];
        }
        if (total/NUM_OF_TESTS >= 50){
               courseGrade = “Pass”;
        }else{
               courseGrade = “No Pass”;
        }
   }
}
```

## A Superclass and Three Subclasses

- A program diagram of the abstract superclass Student and its three subclasses.



## Abstract Superclasses and Abstract Methods

- The best approach depends on the particular situation.
- When considering design options, we can ask ourselves which approach allows easier modification and extension.
- Not all methods can be declared abstract. Private methods and static methods can not be declared abstract.

## Thank You