WEEK 13

Polymorphism

Outline

- Using classes with Polymorphism
  - Reference, behaviour
- Using the `'instanceof'` operator
- Using the type casting for object reference

Learning Objectives

- To use polymorphism with polymorphic reference & polymorphic behaviour.
- To test object type with `'instanceof'` operator & perform object reference type casting.

Polymorphism

- Object-oriented concept that allows us to create software that deals with multiple objects.
- One of the elegant uses of inheritance.

$$\left.\begin{array}{l}\text{Poly} \longrightarrow \text{Many}\\ \text{morphism} \longrightarrow \text{forms}\end{array}\right\}\text{Having multiple forms}$$

Polymorphism

Polymorphic Reference

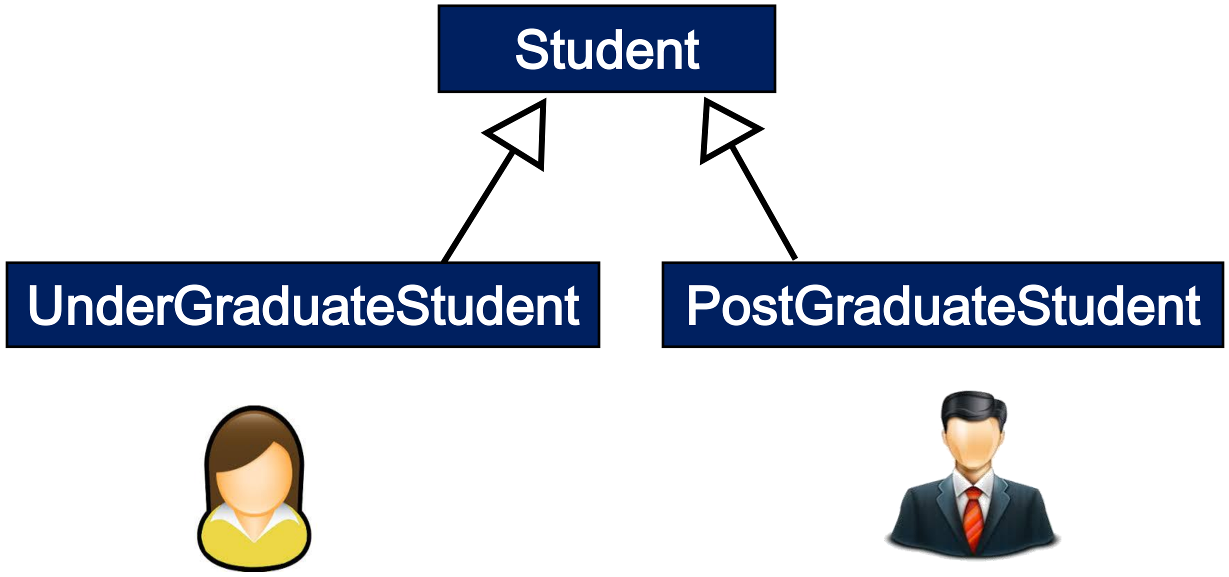
- A variable that can refer to different types of objects at different points in time.
- A reference variable of a parent class type can point to an object of its child class.

Polymorphic Behaviour

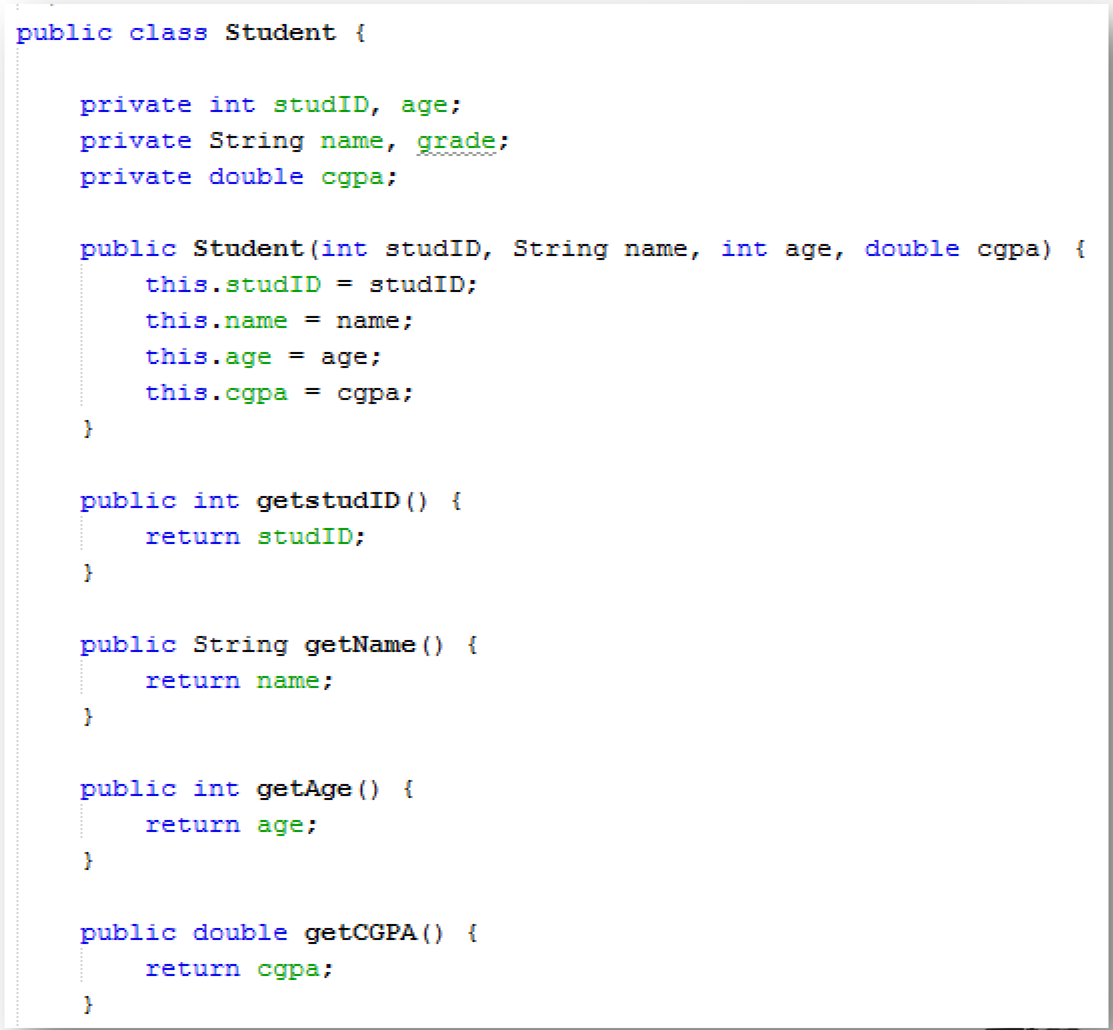
- When a method is invoked using the parent class variable, the class of the object determines which method should be run currently.

Polymorphism Example

Consider these classes:

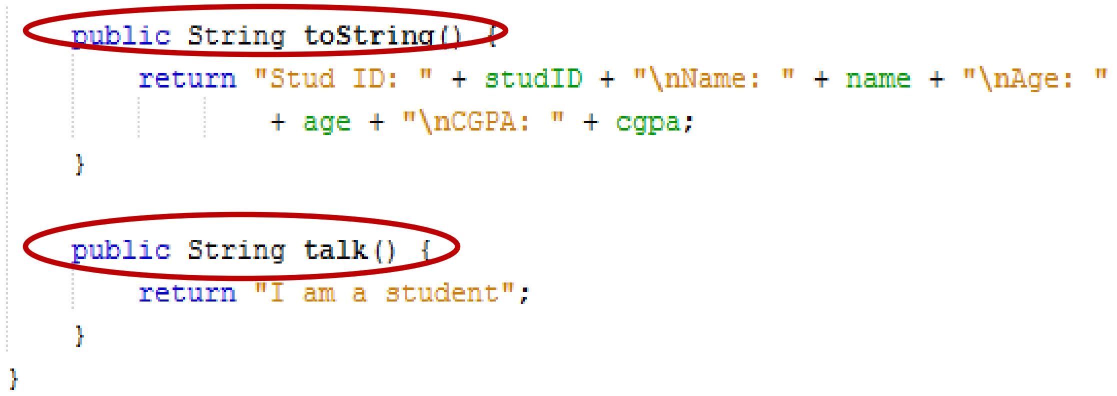


First we create the parent class : **`Student`**



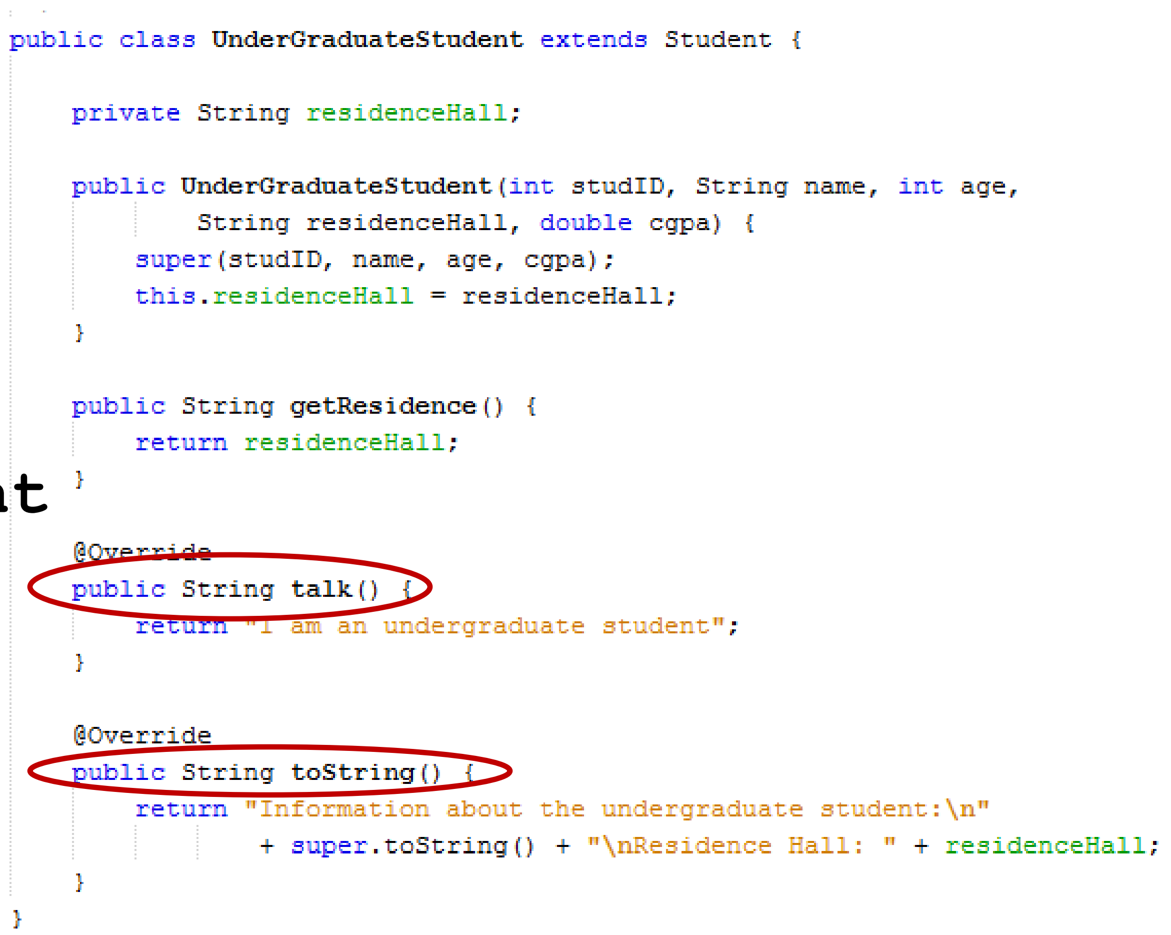
```java
public class Student {

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

    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;
    }

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

`Student` Class (cont)



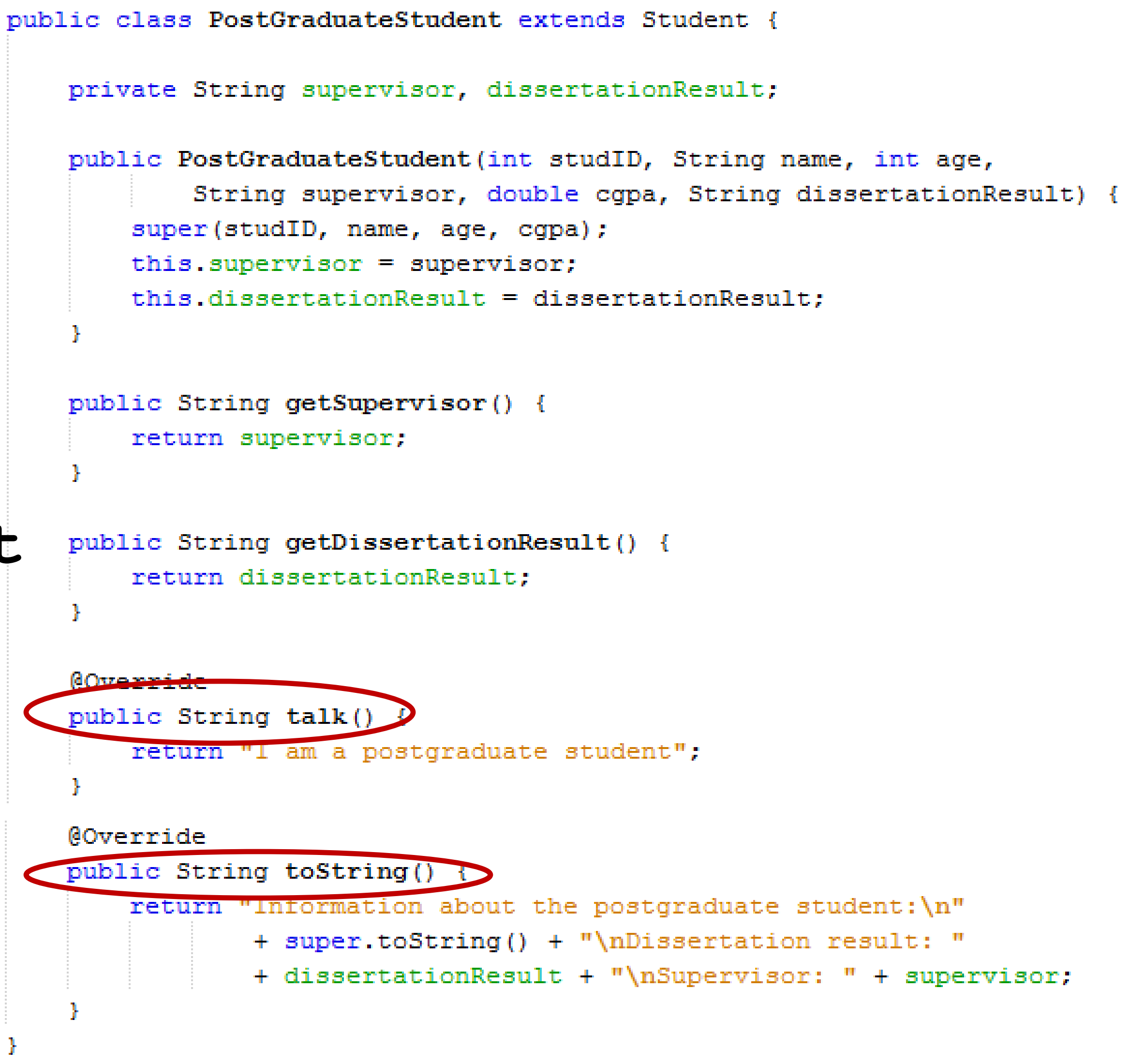
```
    public String toString() {
        return "Stud ID: " + studID + "\nName: " + name + "\nAge: "
                + age + "\nCGPA: " + cgpa;
    }

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

Then we create the child class: **`UnderGraduateStudent`**



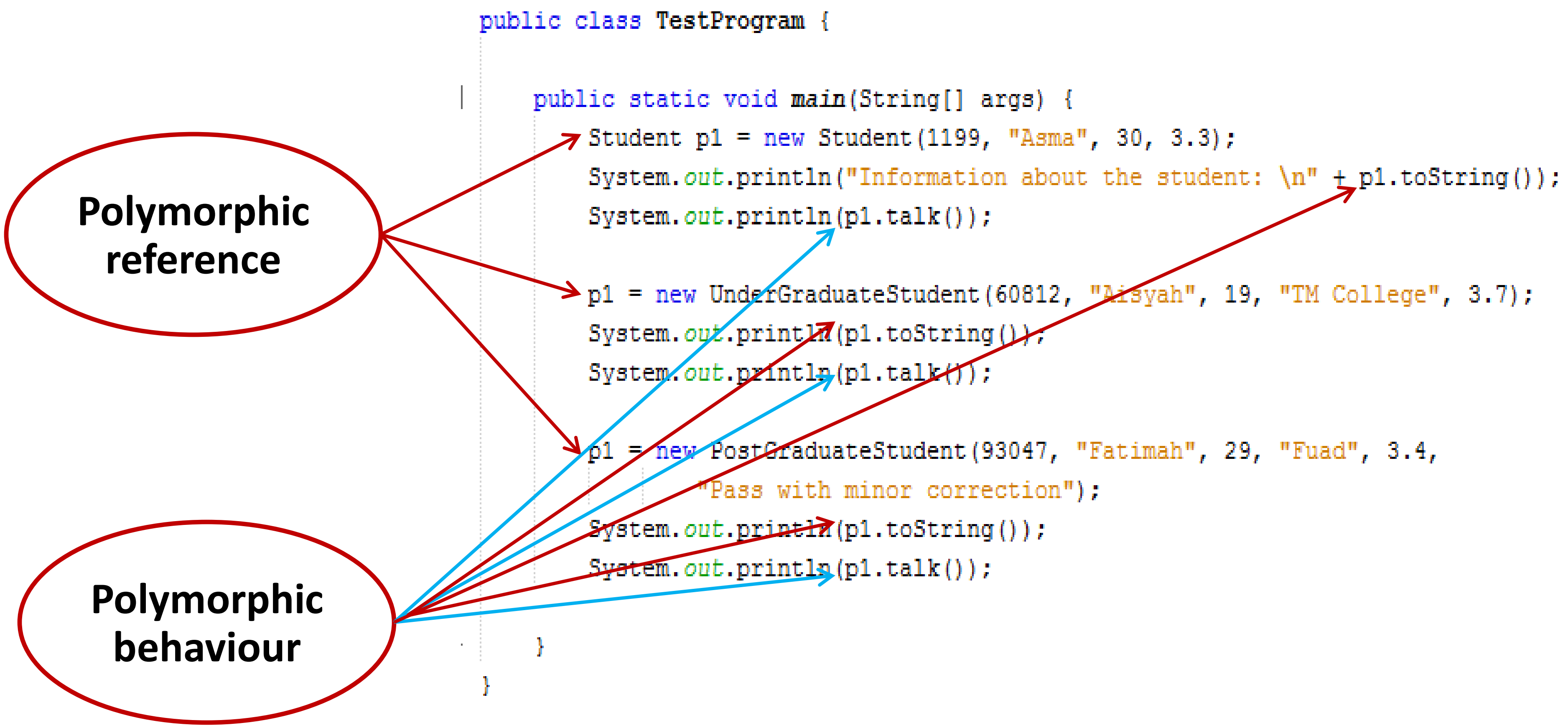
```java
public class UnderGraduateStudent extends Student {

    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;
    }
}
```

Then we create another child class: **`PostGraduateStudent`**



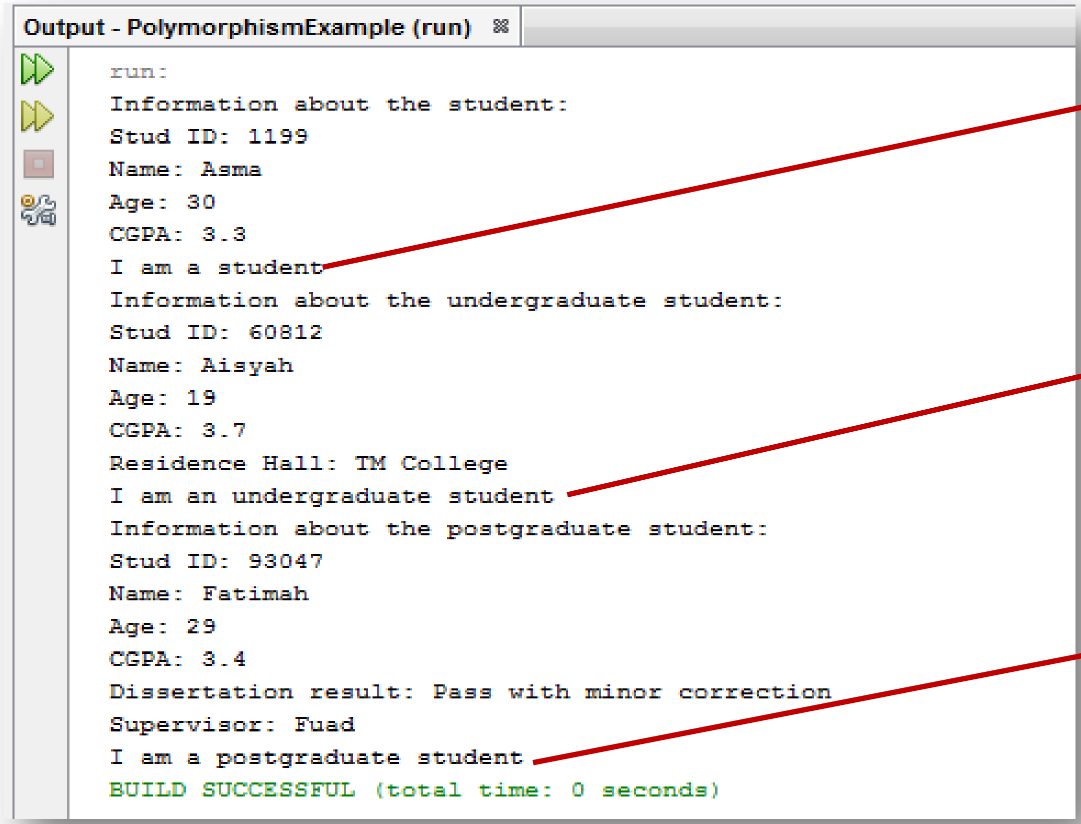
```
public class PostGraduateStudent extends Student {

    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";
    }

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

Next we create a test program: **`TestProgram`**



```java
public class TestProgram {

    public static void main(String[] args) {
        Student p1 = new Student(1199, "Asma", 30, 3.3);
        System.out.println("Information about the student: \n" + p1.toString());
        System.out.println(p1.talk());

        p1 = new UnderGraduateStudent(60812, "Aisyah", 19, "TM College", 3.7);
        System.out.println(p1.toString());
        System.out.println(p1.talk());

        p1 = new PostGraduateStudent(93047, "Fatimah", 29, "Fuad", 3.4,
                "Pass with minor correction");
        System.out.println(p1.toString());
        System.out.println(p1.talk());

    }
}
```

Output:



Output - PolymorphismExample (run)

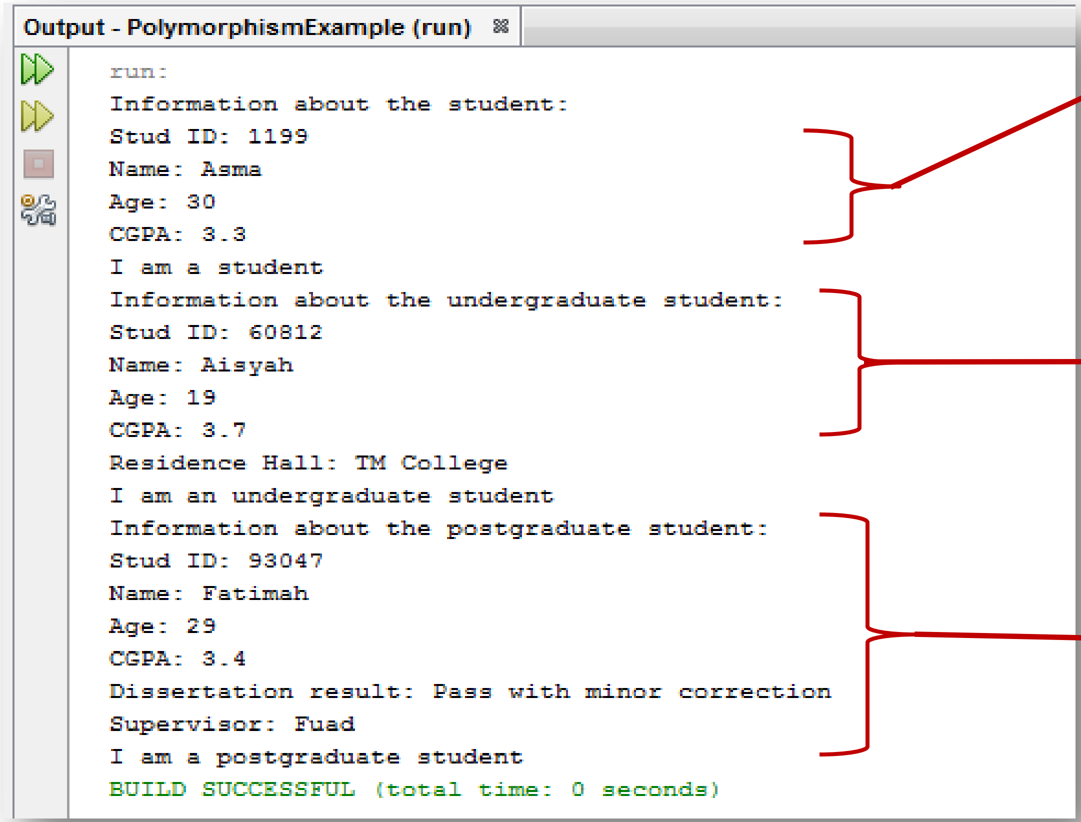
```
run:
Information about the student:
Stud ID: 1199
Name: Asma
Age: 30
CGPA: 3.3
I am a student
Information about the undergraduate student:
Stud ID: 60812
Name: Aisyah
Age: 19
CGPA: 3.7
Residence Hall: TM College
I am an undergraduate student
Information about the postgraduate student:
Stud ID: 93047
Name: Fatimah
Age: 29
CGPA: 3.4
Dissertation result: Pass with minor correction
Supervisor: Fuad
I am a postgraduate student
BUILD SUCCESSFUL (total time: 0 seconds)
```

The `talk()` method in `Student` class is executed

The `talk()` method in `UnderGraduateStudent` class is executed (override)

The `talk()` method in `PostGraduateStudent` class is executed (override)

Output:



Output - PolymorphismExample (run)

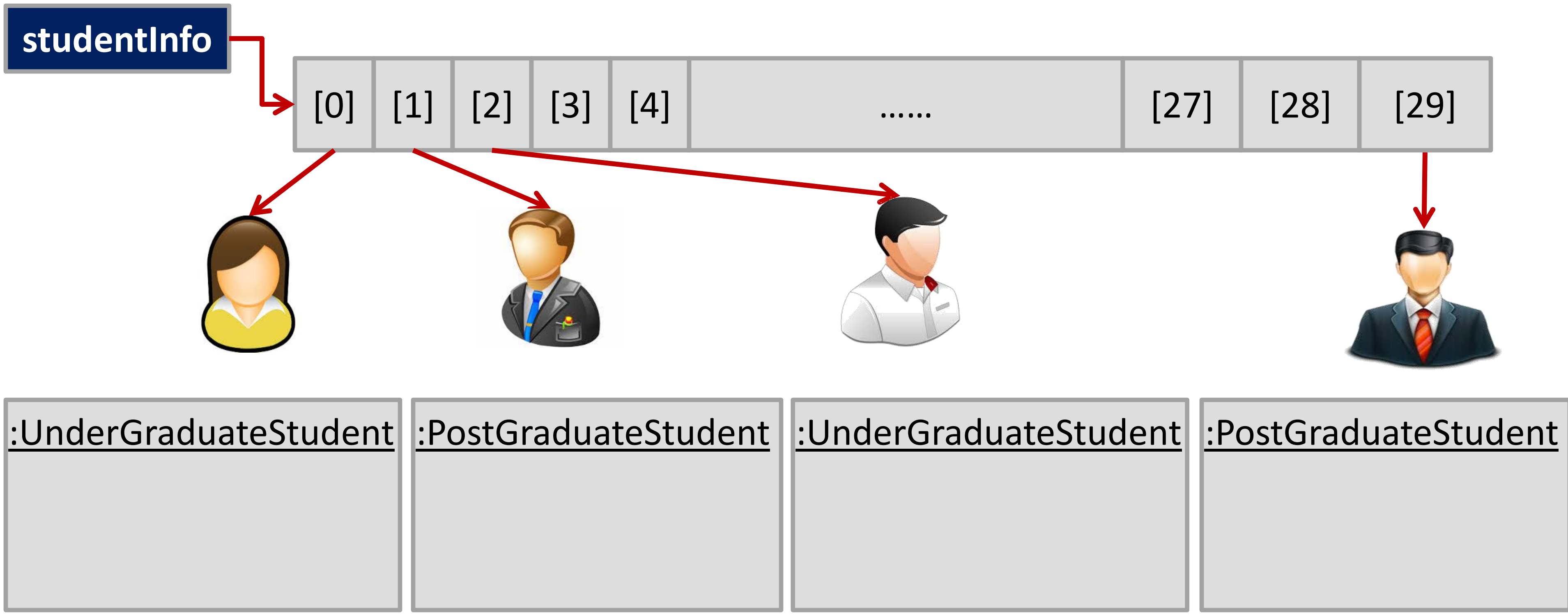
```
run:
Information about the student:
Stud ID: 1199
Name: Asma
Age: 30
CGPA: 3.3
I am a student
Information about the undergraduate student:
Stud ID: 60812
Name: Aisyah
Age: 19
CGPA: 3.7
Residence Hall: TM College
I am an undergraduate student
Information about the postgraduate student:
Stud ID: 93047
Name: Fatimah
Age: 29
CGPA: 3.4
Dissertation result: Pass with minor correction
Supervisor: Fuad
I am a postgraduate student
BUILD SUCCESSFUL (total time: 0 seconds)
```

The `toString()` method in `Student` class is executed

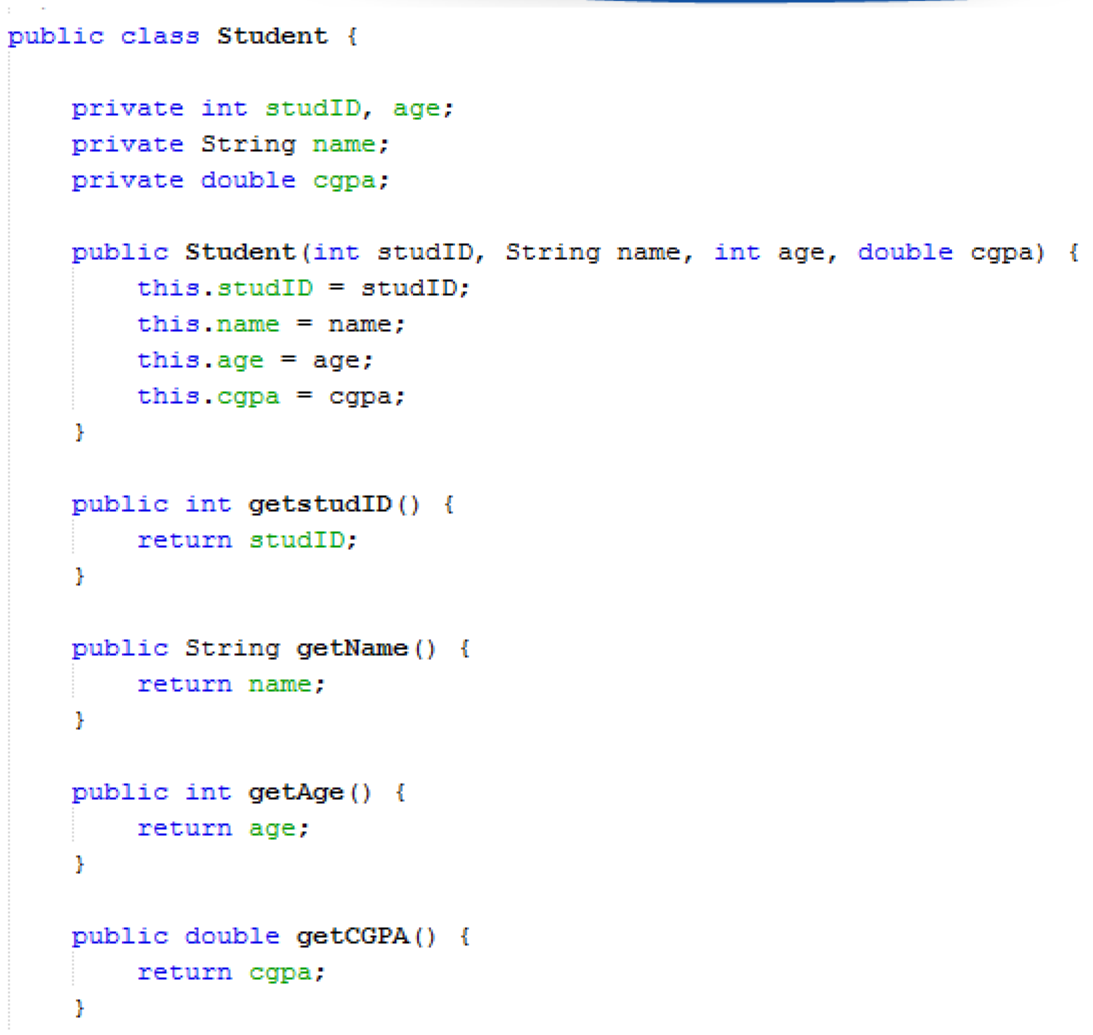
The `toString()` method in `UnderGraduateStudent` class is executed (override)

The `toString()` method in `PostGraduateStudent` class is executed (override)

Creating the studentInfo Array with multiple objects

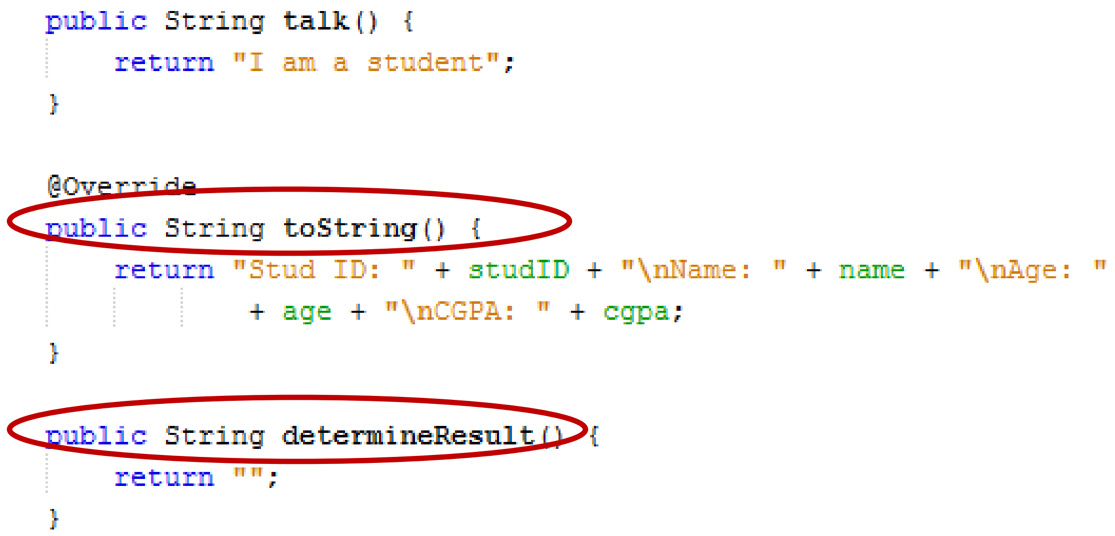


First we create the parent class: **`Student`**



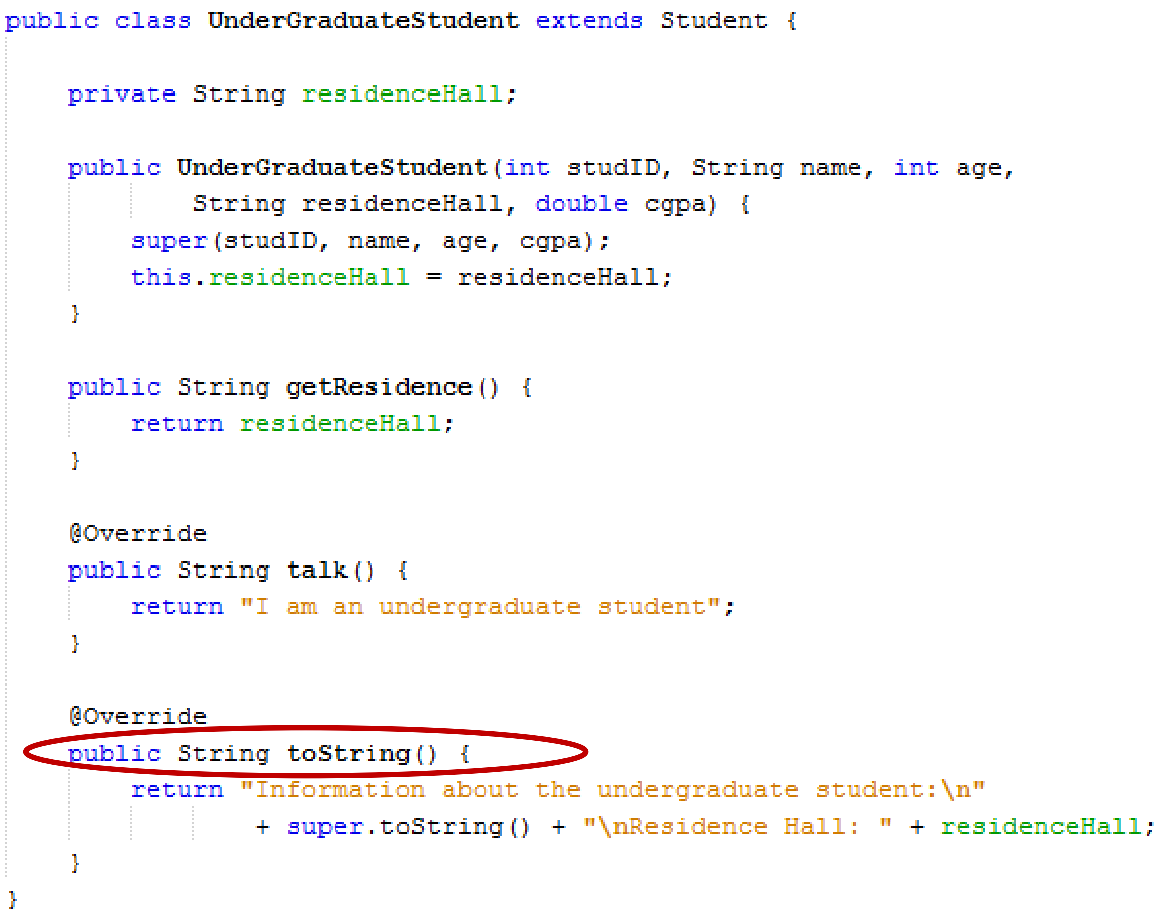
```java
public class Student {

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

    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;
    }

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

`Student` Class (cont)



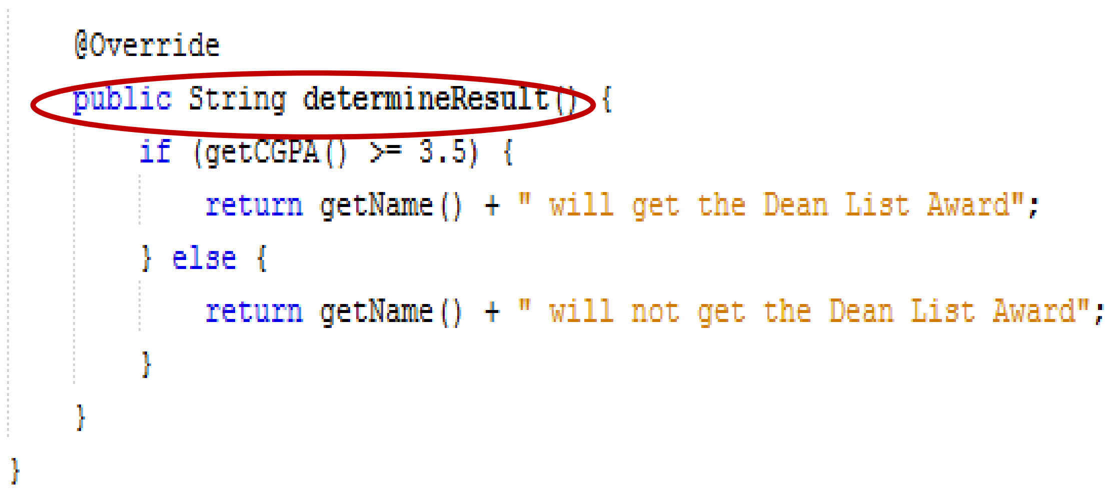
```
public String talk() {
    return "I am a student";
}

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

public String determineResult() {
    return "";
}
```

Then we create the child class: `UnderGraduateStudent`



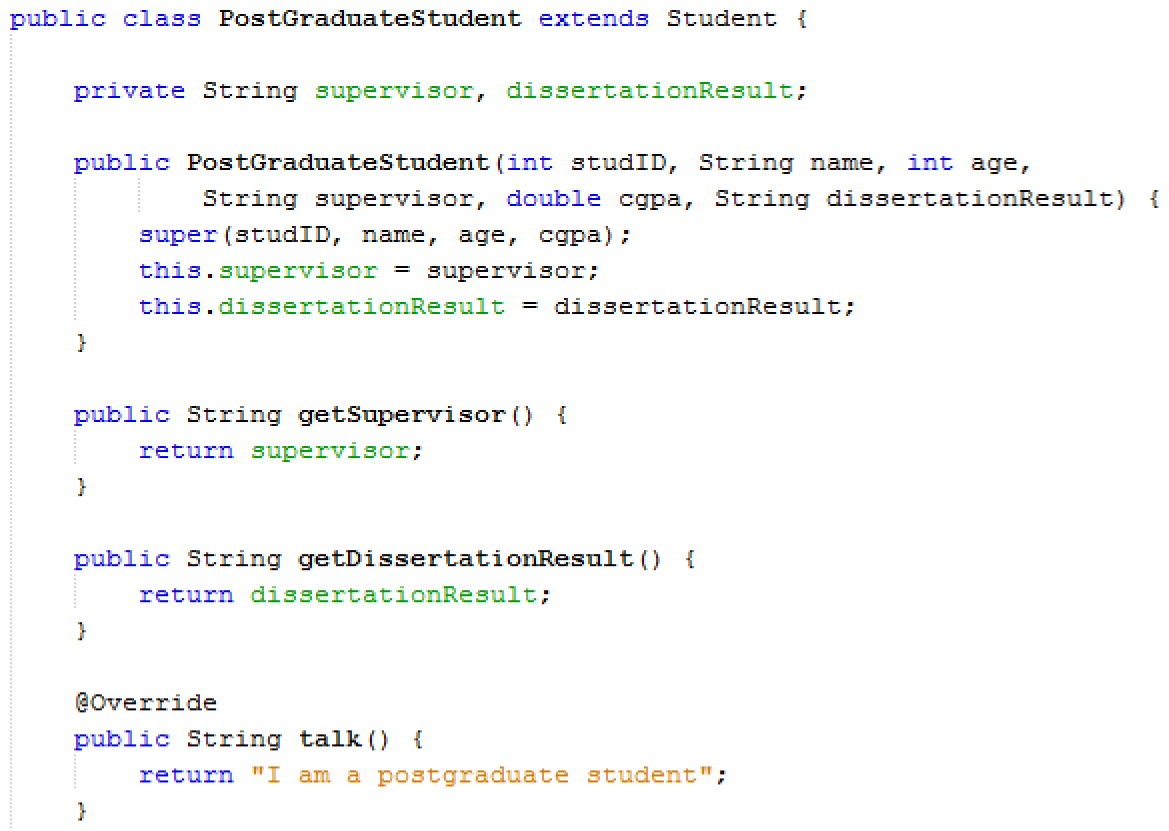
```java
public class UnderGraduateStudent extends Student {

    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)



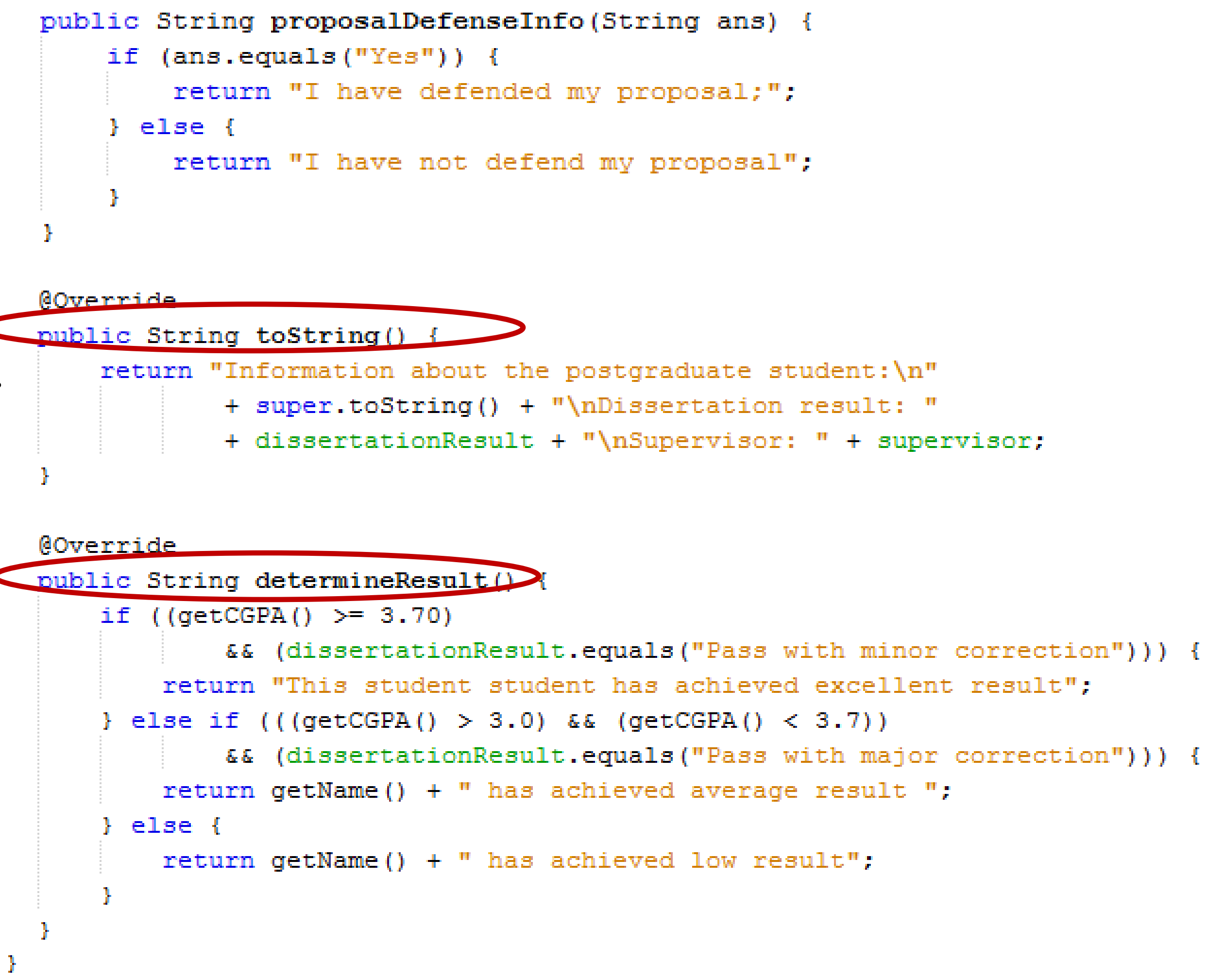
```
    @Override
    public String determineResult() {
        if (getCGPA() >= 3.5) {
            return getName() + " will get the Dean List Award";
        } else {
            return getName() + " will not get the Dean List Award";
        }
    }
}
```

Then we create another child class: **`PostGraduateStudent`**



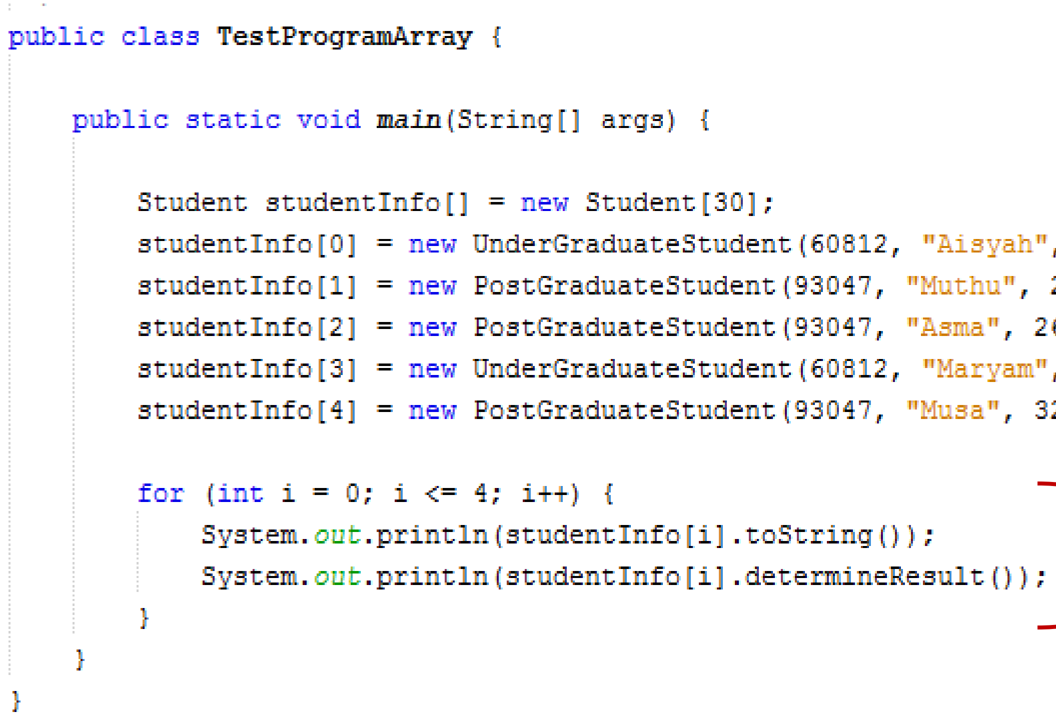
```java
public class PostGraduateStudent extends Student {

    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";
    }
}
}
```

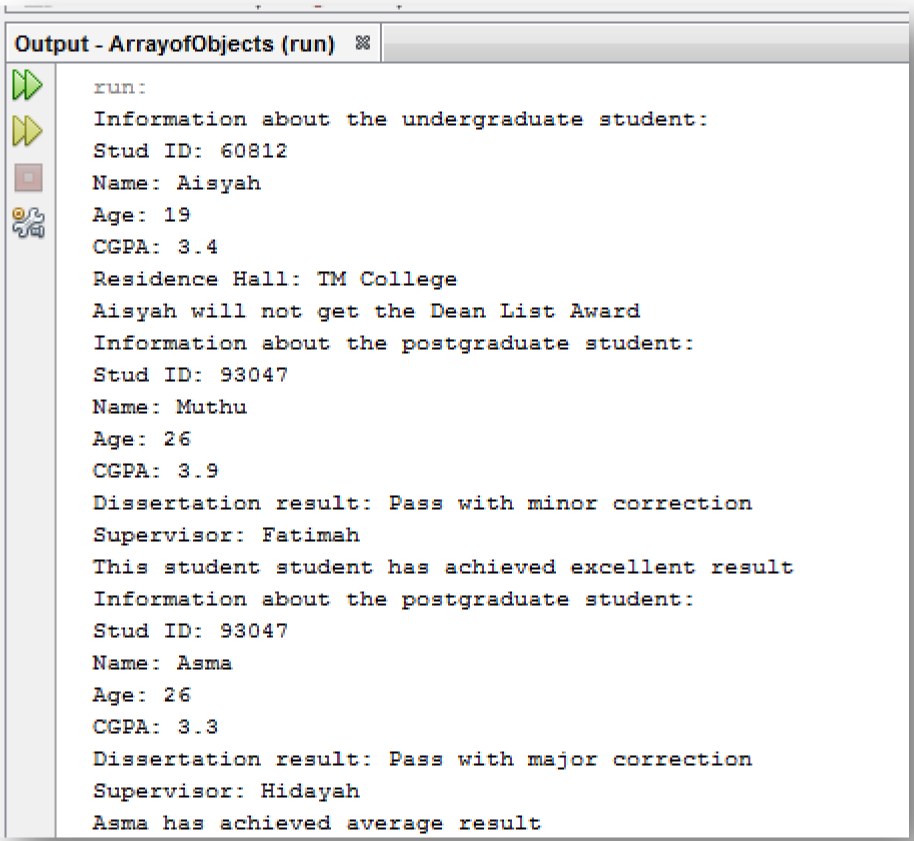
To display students' information and determine their result in the `studentInfo`:



```java
public class TestProgramArray {

    public static void main(String[] args) {

        Student studentInfo[] = new Student[30];
        studentInfo[0] = new UnderGraduateStudent(60812, "Aisyah", 19, "TM College", 3.4);
        studentInfo[1] = new PostGraduateStudent(93047, "Muthu", 26, "Fatimah", 3.9, "Pass with minor correction");
        studentInfo[2] = new PostGraduateStudent(93047, "Asma", 26, "Hidayah", 3.3, "Pass with major correction");
        studentInfo[3] = new UnderGraduateStudent(60812, "Maryam", 20, "BSN College", 3.0);
        studentInfo[4] = new PostGraduateStudent(93047, "Musa", 32, "Lim", 3.8, "Pass with minor correction");

        for (int i = 0; i <= 4; i++) {
            System.out.println(studentInfo[i].toString());
            System.out.println(studentInfo[i].determineResult());
        }
    }
}
```

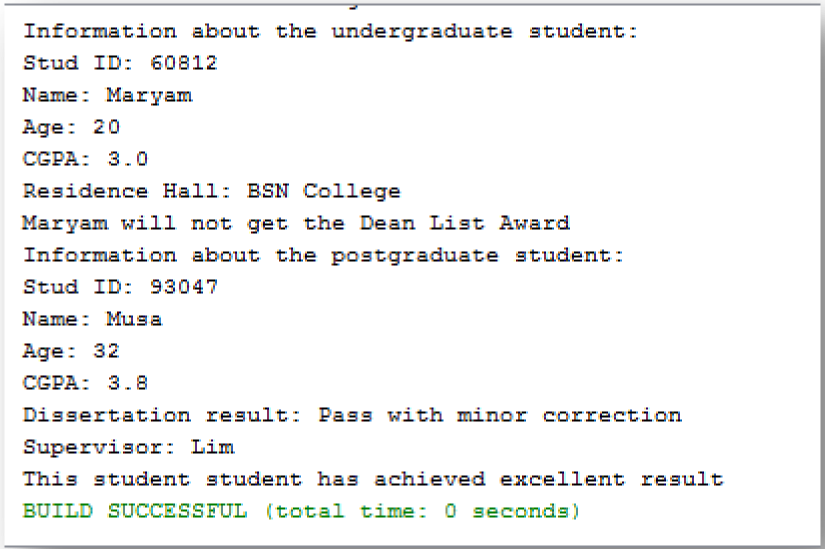
The `toString()` and `determineResult()` methods are executed based on the current type of object (either `PostGraduateStudent` or `UnderGraduateStudent`)

Output:



Output - ArrayofObjects (run)

```
run:
Information about the undergraduate student:
Stud ID: 60812
Name: Aisyah
Age: 19
CGPA: 3.4
Residence Hall: TM College
Aisyah will not get the Dean List Award
Information about the postgraduate student:
Stud ID: 93047
Name: Muthu
Age: 26
CGPA: 3.9
Dissertation result: Pass with minor correction
Supervisor: Fatimah
This student student has achieved excellent result
Information about the postgraduate student:
Stud ID: 93047
Name: Asma
Age: 26
CGPA: 3.3
Dissertation result: Pass with major correction
Supervisor: Hidayah
Asma has achieved average result
```



```
Information about the undergraduate student:
Stud ID: 60812
Name: Maryam
Age: 20
CGPA: 3.0
Residence Hall: BSN College
Maryam will not get the Dean List Award
Information about the postgraduate student:
Stud ID: 93047
Name: Musa
Age: 32
CGPA: 3.8
Dissertation result: Pass with minor correction
Supervisor: Lim
This student student has achieved excellent result
BUILD SUCCESSFUL (total time: 0 seconds)
```

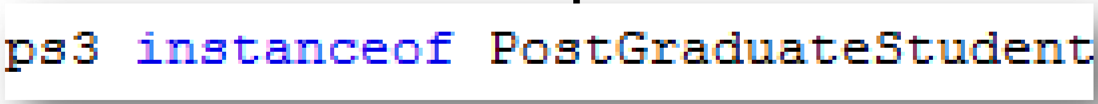
The `instanceof` Operator

Can help us to test if an object is of a particular class.
The reserved word `instanceof` is used to test an object

Syntax:

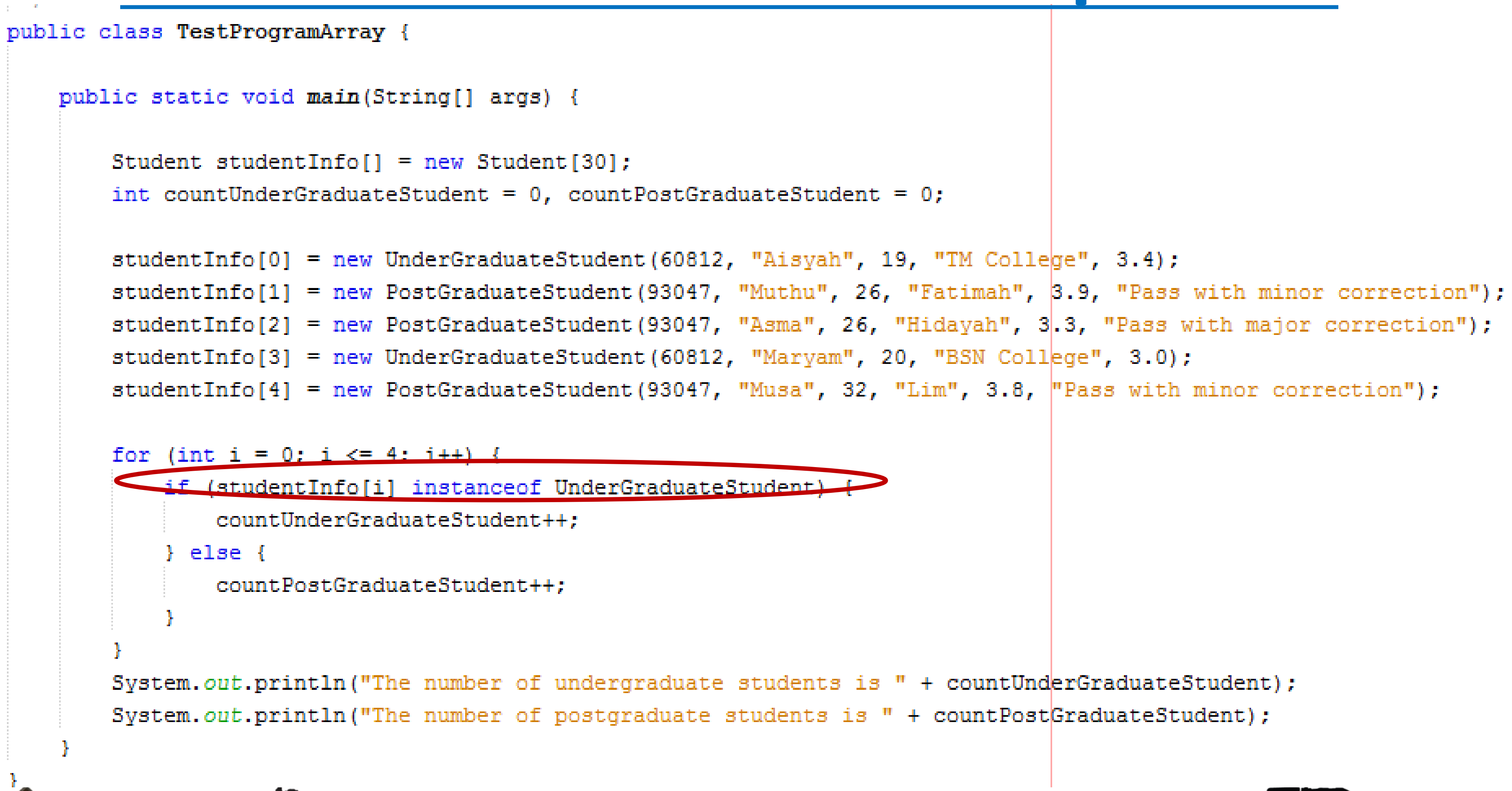
```
<object_reference_name> instanceof <class_name>
```

Example:



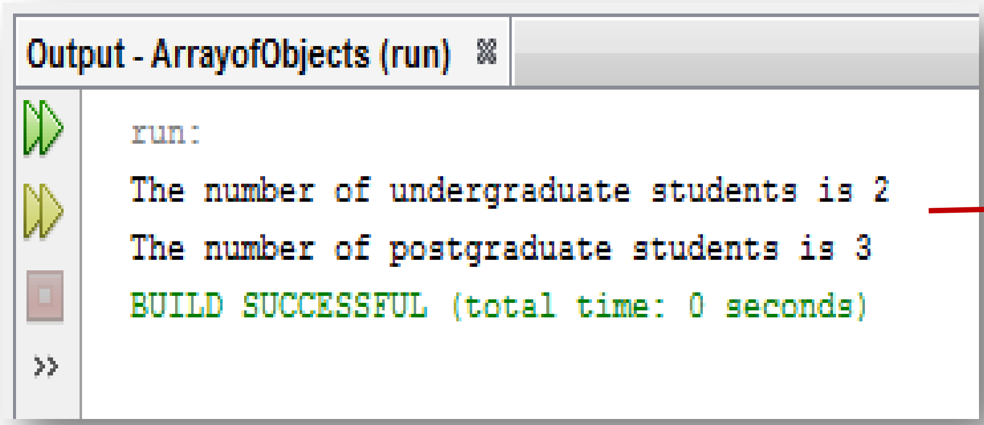
```
ps3 instanceof PostGraduateStudent
```

The Use of `instanceof` Operator



```java
public class TestProgramArray {

    public static void main(String[] args) {

        Student studentInfo[] = new Student[30];
        int countUnderGraduateStudent = 0, countPostGraduateStudent = 0;

        studentInfo[0] = new UnderGraduateStudent(60812, "Aisyah", 19, "TM College", 3.4);
        studentInfo[1] = new PostGraduateStudent(93047, "Muthu", 26, "Fatimah", 3.9, "Pass with minor correction");
        studentInfo[2] = new PostGraduateStudent(93047, "Asma", 26, "Hidayah", 3.3, "Pass with major correction");
        studentInfo[3] = new UnderGraduateStudent(60812, "Maryam", 20, "BSN College", 3.0);
        studentInfo[4] = new PostGraduateStudent(93047, "Musa", 32, "Lim", 3.8, "Pass with minor correction");

        for (int i = 0; i <= 4; i++) {
            if (studentInfo[i] instanceof UnderGraduateStudent) {
                countUnderGraduateStudent++;
            } else {
                countPostGraduateStudent++;
            }
        }
        System.out.println("The number of undergraduate students is " + countUnderGraduateStudent);
        System.out.println("The number of postgraduate students is " + countPostGraduateStudent);
    }
}
```

Output:



```
run:
The number of undergraduate students is 2
The number of postgraduate students is 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

The number of undergraduate and postgraduate objects in `studInfo` array are calculated

Casting Objects

A process of assigning one object reference into another object reference.

Example:

```
Object o = new Student(); //implicit casting
Student b = (Student)o; //explicit casting
```

Types of Object Casting

UPCASTING

•Casting an instance of a child class to a variable of a parent class.

•Casting can be done implicitly or explicitly

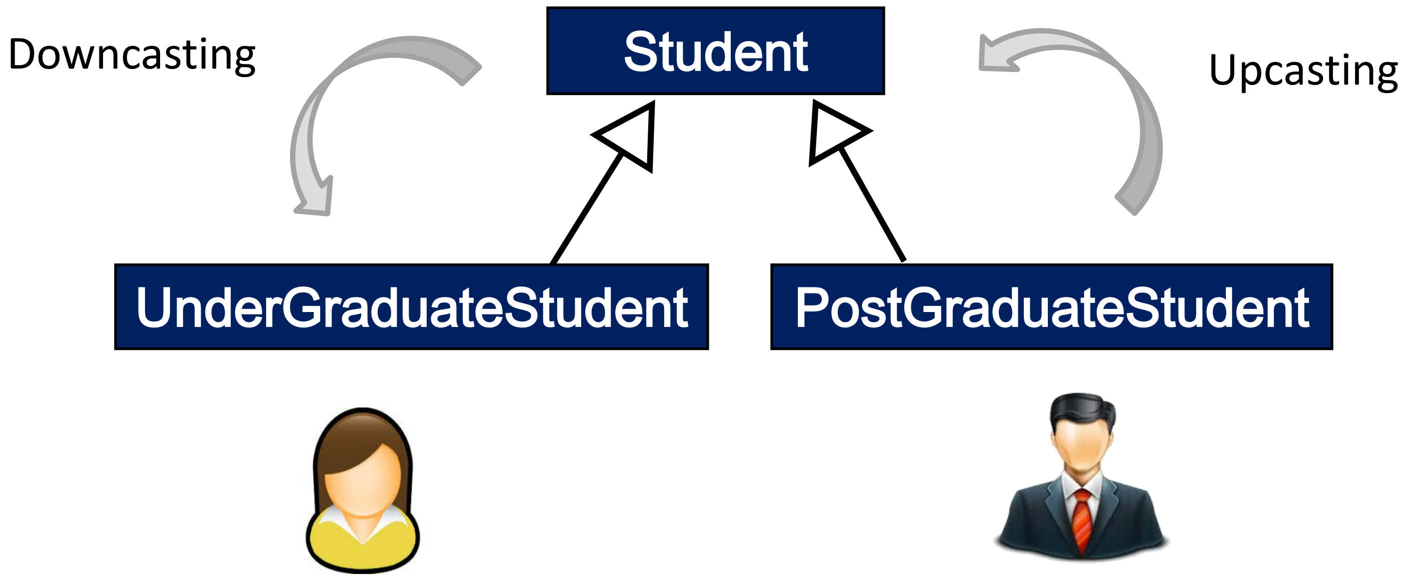
DOWNCASTING

•Casting an instance of a parent class to a variable of its child class

•Casting must be done explicitly.

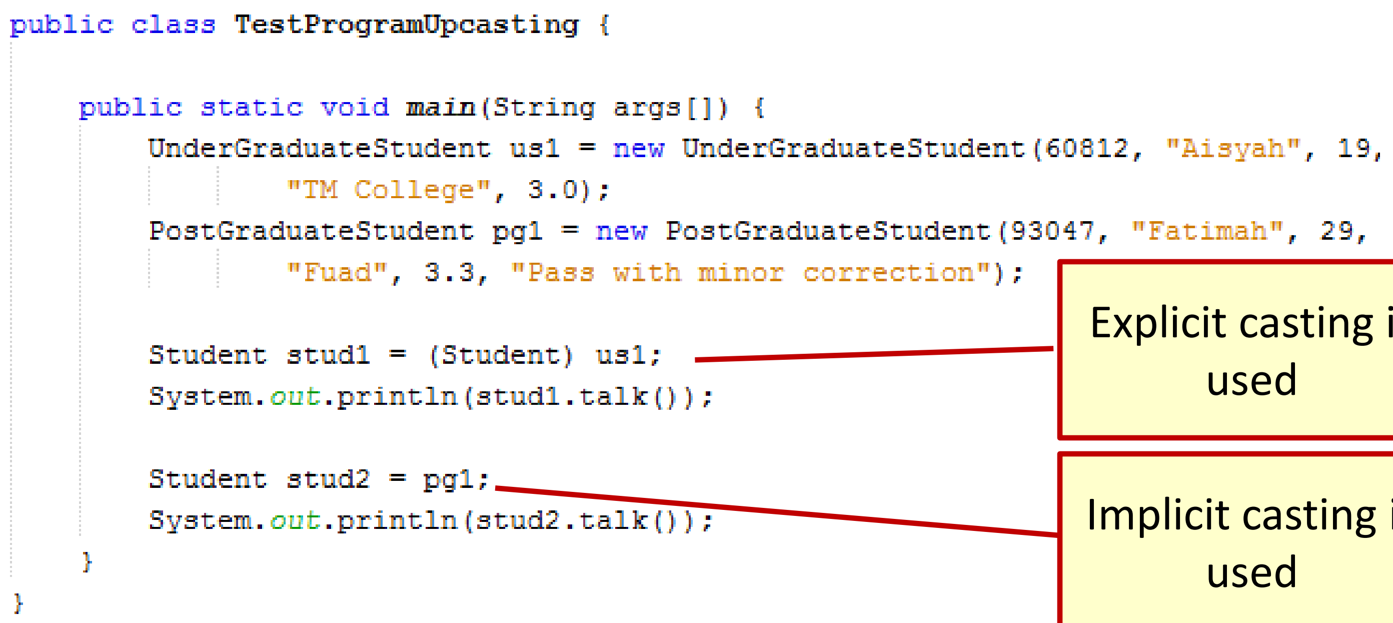
Casting Example

Consider these classes again:



Upcasting Example

Using the existing `Student`, `UnderGraduateStudent` and `PostGraduateStudent` classes, a `TestProgramUpcasting` is created:

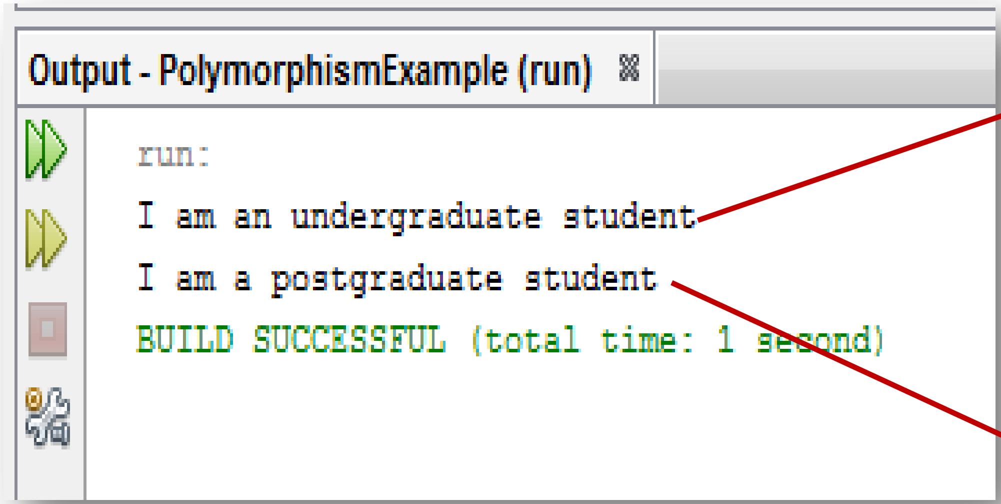


```
public class TestProgramUpcasting {

    public static void main(String args[]) {
        UnderGraduateStudent us1 = new UnderGraduateStudent(60812, "Aisyah", 19,
                "TM College", 3.0);
        PostGraduateStudent pg1 = new PostGraduateStudent(93047, "Fatimah", 29,
                "Fuad", 3.3, "Pass with minor correction");

        Student stud1 = (Student) us1;
        System.out.println(stud1.talk());

        Student stud2 = pg1;
        System.out.println(stud2.talk());
    }
}
```

Explicit casting is used

Implicit casting is used

Output



```
run:
I am an undergraduate student
I am a postgraduate student
BUILD SUCCESSFUL (total time: 1 second)
```

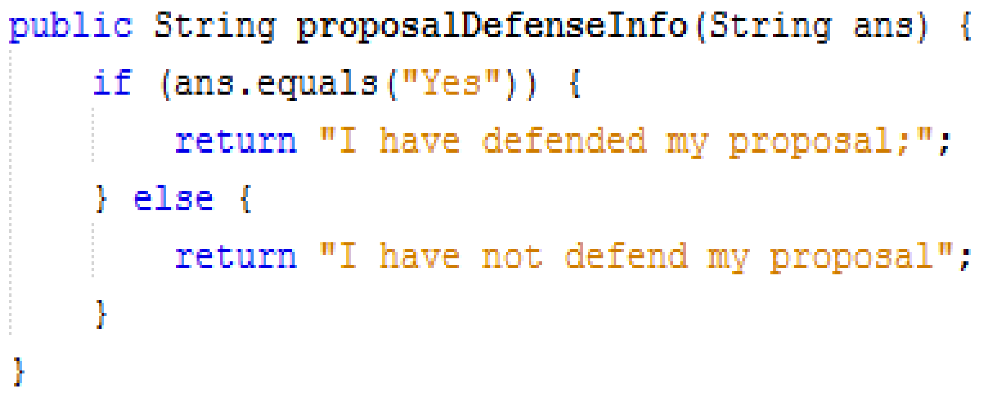
The `talk()` method in `UnderGraduateStudent` class is called even though the object reference variable is of type `Student`

The `talk()` method in `PostGraduateStudent` class is called even though the object reference variable is of type `Student`

This is because the variables `stud1` and `stud2` point to objects of `UnderGraduateStudent` and `PostGraduateStudent` respectively as a result of UPCASTING. Thus, the respective child class methods are called

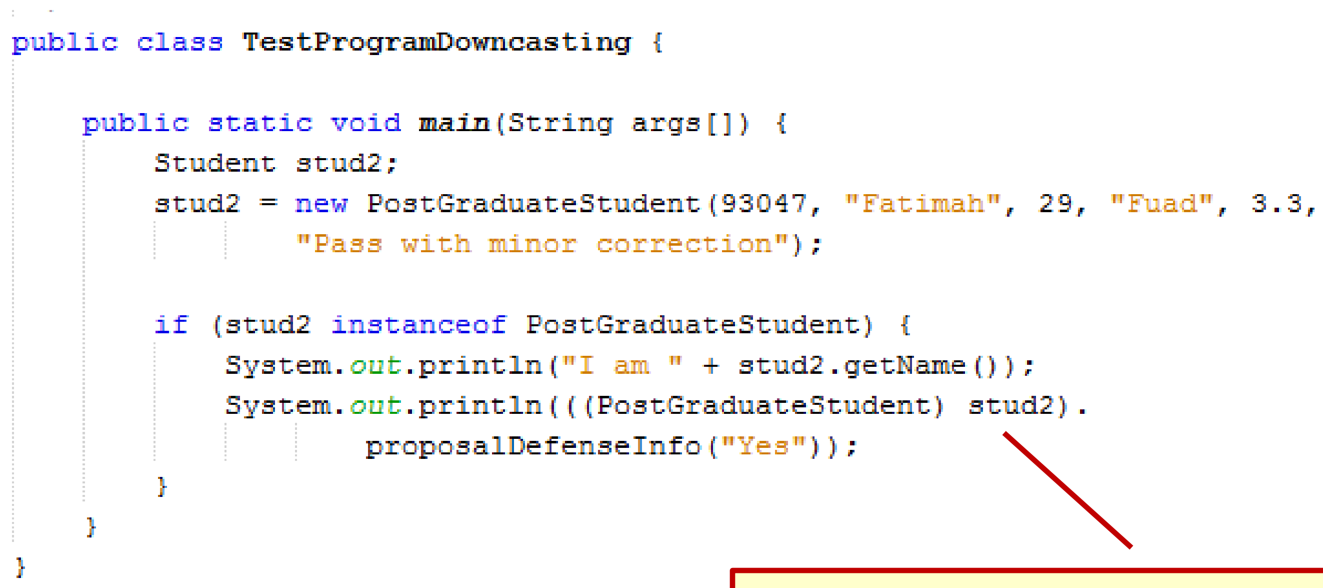
Downcasting Example

Considering previous example, lets say we have a unique method in class `PostGraduateStudent`: `proposalDefenseInfo()`



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

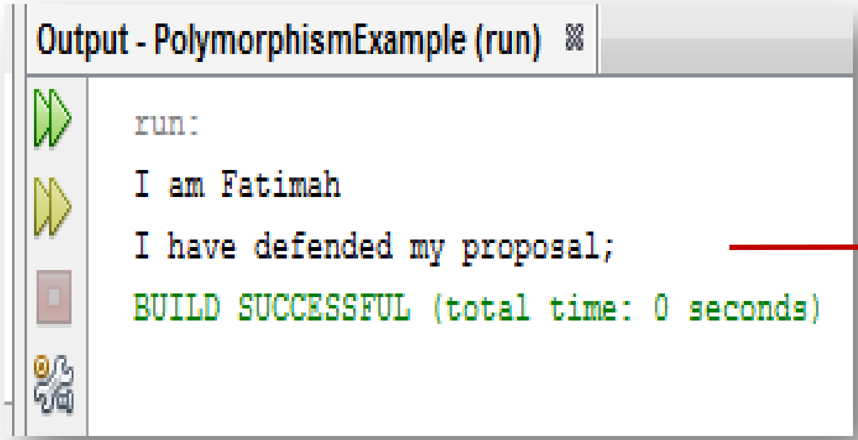
Next we create a test program: **`TestProgramDowncasting`**



```
public class TestProgramDowncasting {

    public static void main(String args[]) {
        Student stud2;
        stud2 = new PostGraduateStudent(93047, "Fatimah", 29, "Fuad", 3.3,
                "Pass with minor correction");

        if (stud2 instanceof PostGraduateStudent) {
            System.out.println("I am " + stud2.getName());
            System.out.println(((PostGraduateStudent) stud2).
                    proposalDefenseInfo("Yes"));
        }
    }
}
```

Object `stud2` is type casted to `PostGraduateStudent`

Output

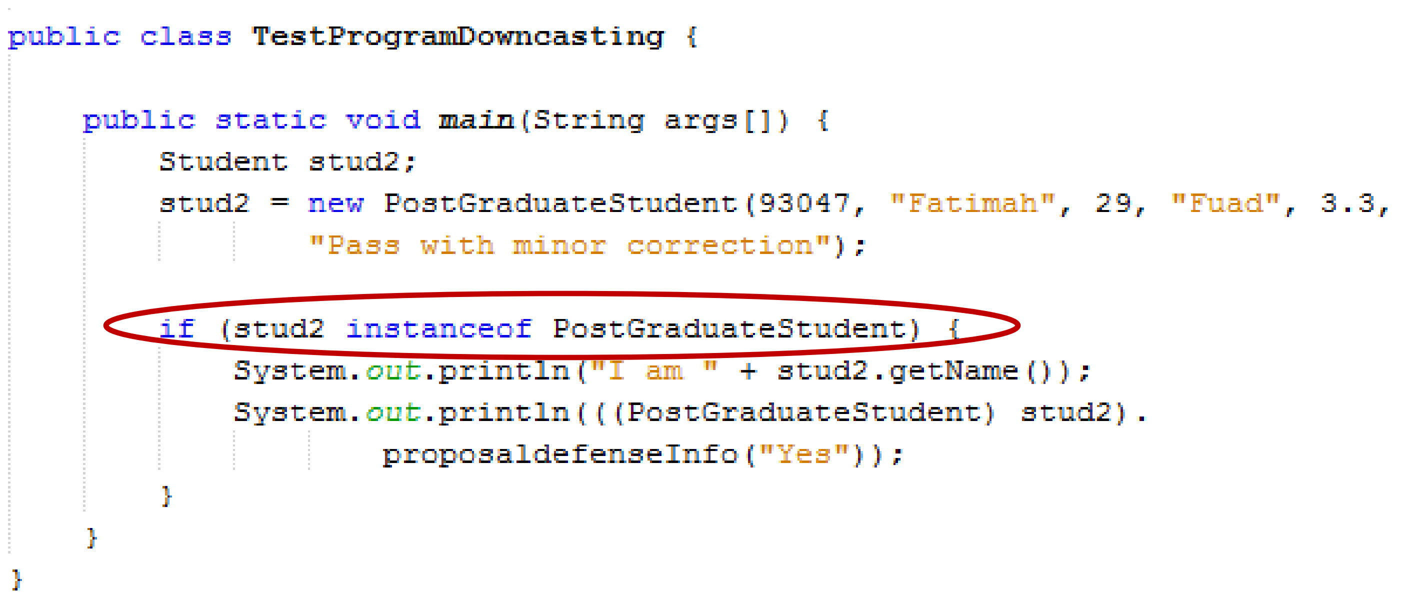

Output - PolymorphismExample (run)

```
run:
I am Fatimah
I have defended my proposal;
BUILD SUCCESSFUL (total time: 0 seconds)
```

`proposalDefenseInfo()` method is executed

Casting Objects and the instanceof Operator

For casting to be successful, we must ensure that the object to be casted `stud2` is an instance of the child class `PostGraduateStudent`



```
public class TestProgramDowncasting {

    public static void main(String args[]) {
        Student stud2;
        stud2 = new PostGraduateStudent(93047, "Fatimah", 29, "Fuad", 3.3,
                "Pass with minor correction");

        if (stud2 instanceof PostGraduateStudent) {
            System.out.println("I am " + stud2.getName());
            System.out.println(((PostGraduateStudent) stud2).
                    proposaldefenseInfo("Yes"));
        }
    }
}
```

Summary

- Two main concepts of polymorphism are polymorphic reference and polymorphic behaviour.
- Polymorphic reference allows a single variable to refer to objects from different child classes in the same inheritance hierarchy.
- Polymorphic behaviour allows the use of parent class variable to invoke the method in its child.
- The instanceof operator is used for ensuring the class of a particular object.
- There are two types of casting, which are upcasting (implicitly or explicitly) and downcasting (explicitly only).