8.3 Polymorphism

- Dynamic Binding
- Type Checking and Dynamic Binding
- Dynamic Binding with `toString`
- Polymorphism

Static and Dynamic Binding

- *Binding* : determining the memory addresses for jumps
- *Static*: done at
  - » also called *offline*
- *Dynamic*: done at
- Compilation is done *offline*
  - » it is a separate operation done before running a program
- Binding done at compile time is, therefor, static, and
- Binding done at run time is dynamic
  - » also called

Example of Dynamic Binding: General Description

- Derived classes call a method in their parent class which calls a method that is overridden (defined) in each of the derived classes
  - » the parent class is compiled separately and before the derived classes are even written
  - » the compiler cannot possibly know to use
  - » therefore the address must be determined (bound) at

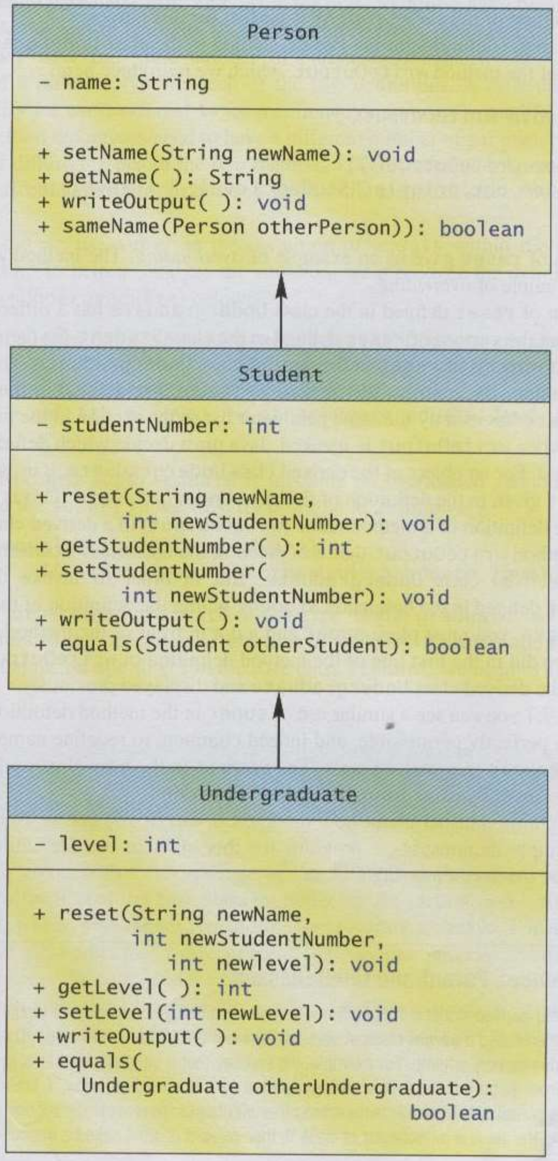
Listing 8.6 A Demo of Polymorphism

```
public class PolymorphismDemo
{
        public static void main(String[] args)
        {
                Person[] people = new Person[4];

                people[0] = new Undergraduate("Cotty, Manny", 4910, 1);
                people[1] = new Undergraduate("Kick, Anita", 9931, 2);
                people[2] = new Student("DeBanque, Robin", 8812);
                people[3] = new Undergraduate("Bugg, June", 9901, 4);

                for (Person p : people)
                {
                        p.writeOutput();
                        System.out.println();
                }
        }
}
```

This code would output:

```
Name: Cotty, Manny
Student Number: 4910
Student Level: 1

Name: Kick, Anita
Student Number: 9931
Student Level: 2

Name: DeBanque, Robin
Student Number: 8812

Name: Bugg, June
Student Number: 9901
Student Level: 4
```

■ DISPLAY 7.8 Some More Details of a UML Class Hierarchy



```
Person
- name: String
+ setName(String newName): void
+ getName( ): String
+ writeOutput( ): void
+ sameName(Person otherPerson)): boolean

Student
- studentNumber: int
+ reset(String newName,
        int newStudentNumber): void
+ getStudentNumber( ): int
+ setStudentNumber(
        int newStudentNumber): void
+ writeOutput( ): void
+ equals(Student otherStudent): boolean

Undergraduate
- level: int
+ reset(String newName,
         int newStudentNumber,
            int newLevel): void
+ getLevel( ): int
+ setLevel(int newLevel): void
+ writeOutput( ): void
+ equals(
    Undergraduate otherUndergraduate):
                               boolean
```

Polymorphism

- Polymorphism : (from Greek)
- Using the process of to allow different objects to use different method actions for the same method name
- Now the term usually refers to use of dynamic binding – overriden method

Dynamic Binding: Specific Example

Parent class: `Figure`

- » Defines methods: `drawAt` and `drawHere`
- » `drawAt` calls `drawHere`

Derived class: `Box extends Figure`

- » Inherits `drawAt`
- » **Redefines (overrides) `drawHere`**
- » Calls `drawAt`
  - – uses the parent's `drawAt` method
  - – which must call this, the derived class's, `drawHere` method

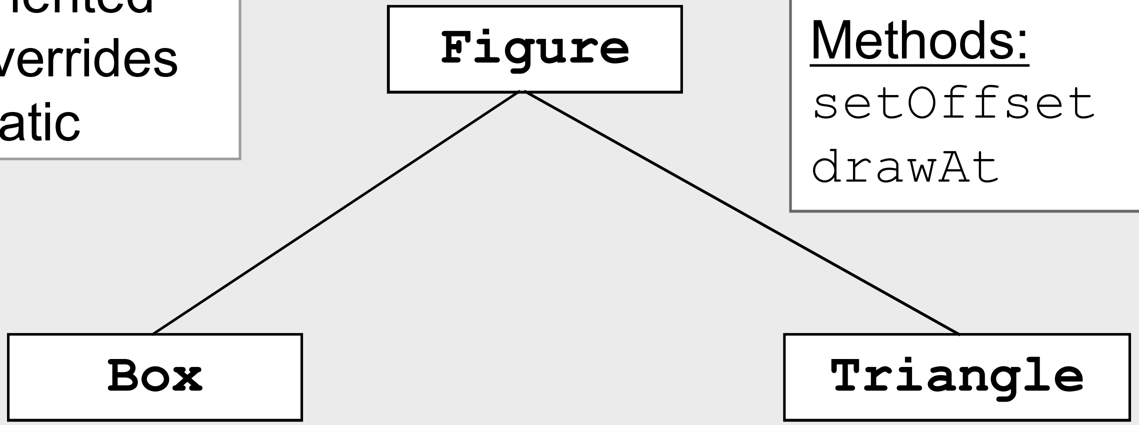
- `Figure` is compiled before `Box` is even written, so **the address of `drawHere`(in the derived class `Box`) cannot be known** then
  - » it must be determined during e, i.e.

Character Graphics Example

🟦 Inherited

🟥 Overrides

🟩 Static



Instance variables:

offset

Methods:

setOffset getOffset

drawAt drawHere

Box

Instance variables:

offset height width

Methods:

setOffset **getOffset**

drawAt **drawHere**

reset drawHorizontalLine

drawSides drawOneLineOfSides

spaces

Triangle

Instance variables:

offset base

Methods:

setOffset **getOffset**

drawAt **drawHere**

reset drawBase

drawTop **spaces**

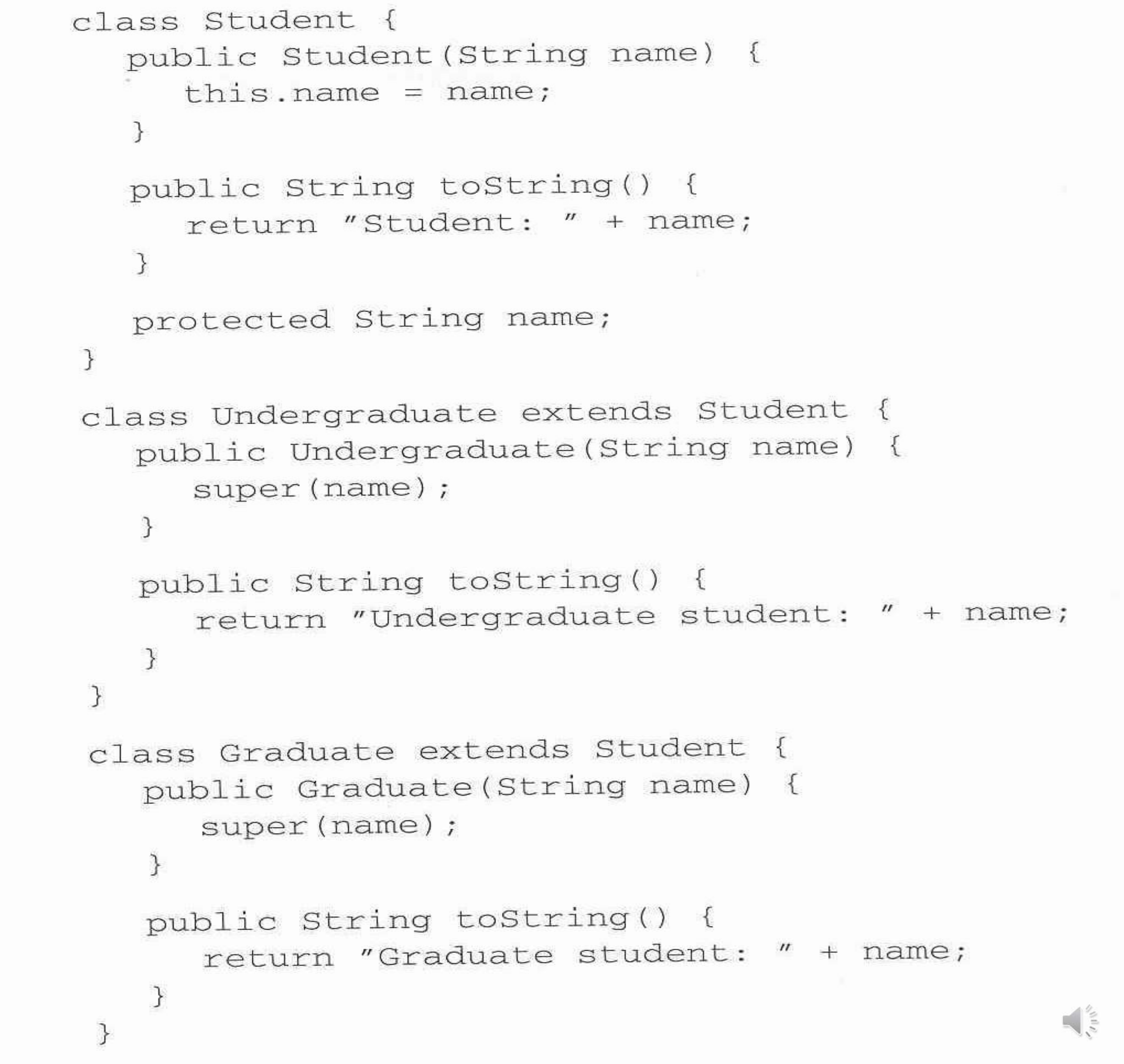
Dynamic Binding: Specific Example

- Ex)

```
Figure f ;
Box b = new Box(1,4,4);
f = b;
f.drawAt(2);
Triangle t = new Triangle(1,21);
f = t;
f.drawAt(2)
```

```
public class Figure
{


    ...............

    /**
     Draws the figure at lineNumber lines down
     from the current line.
    */
    public void drawAt(int lineNumber)
    {
        int count;
        for (count = 0; count < lineNumber; count++)
            System.out.println( );
        drawHere( );
    }
    /**
     Draws the figure at the current line.
    */
    public void drawHere( )
    {
        int count;
        for (count = 0; count < offset; count++)
            System.out.print(' ');
        System.out.println("*");
    }
 }
```

```
public class Box extends Figure
{

     …………..


 public void reset(int newOffset, int newHeight, int newWidth)
    {
        setOffset(newOffset);
        height = newHeight;
        width = newWidth;
    }

    /**
     Draws the figure at the current line.
    */
    public void drawHere( )
    {
        drawHorizontalLine( );
        drawSides( );
        drawHorizontalLine( );
    }
```



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

   public String toString() {
      return "Student: " + name;
   }

   protected String name;
}

class Undergraduate extends Student {
   public Undergraduate(String name) {
      super(name);
   }

   public String toString() {
      return "Undergraduate student: " + name;
   }
}

class Graduate extends Student {
   public Graduate(String name) {
      super(name);
   }

   public String toString() {
      return "Graduate student: " + name;
   }
}
```

```
public class Course {
  public void enroll(Student s) {
    if (s != null && count < CAPACITY)
      students[count++] = s;
  }

  public void list() {
    for (int i = 0; i < count; i++)
      System.out.println(students[i].toString());
  }

  protected static final int CAPACITY = 40;
  protected Student students[] = new Student[CAPACITY];
  protected int count = 0;
}
```

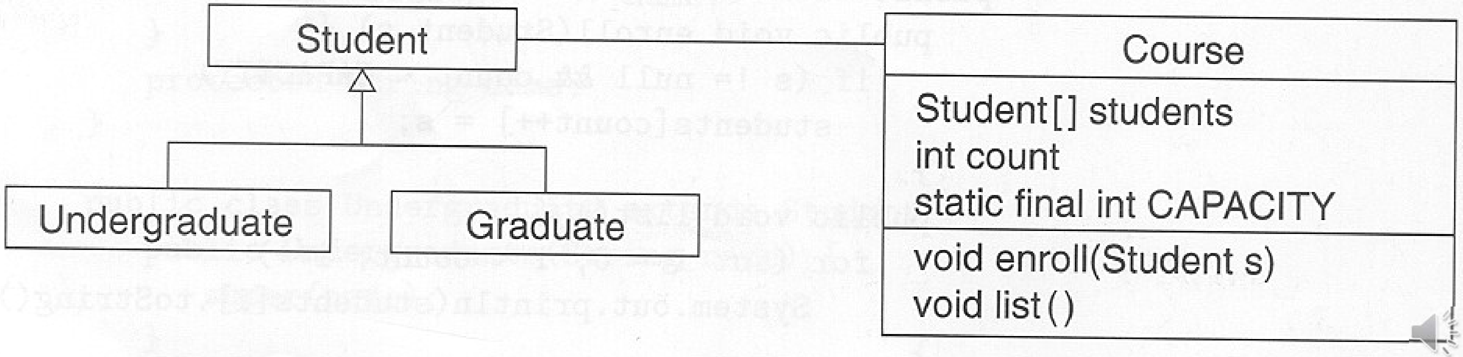
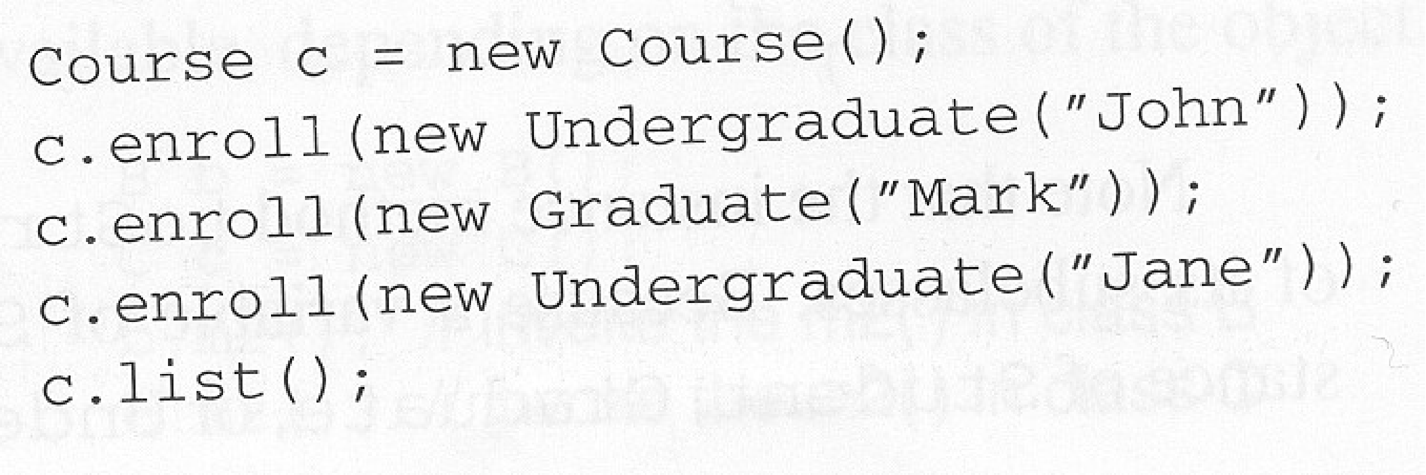
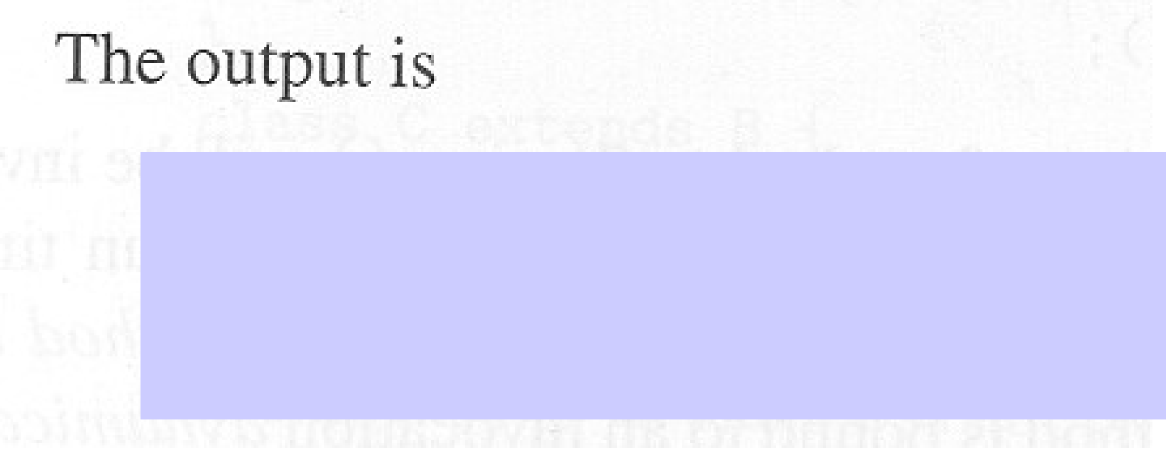


Figure 5.2

Students and courses.



```
Course c = new Course();
c.enroll(new Undergraduate("John"));
c.enroll(new Graduate("Mark"));
c.enroll(new Undergraduate("Jane"));
c.list();
```



The output is

끝