This section of lecture notes is devoted to the discussion of Abstract Classes in Java:

Thanks to Prof. Satish Singhal for allowing me to include these excerpts from his notes on Java Programming:
Abstract Classes

Object oriented languages such as Java and C++ use a concept of abstract classes, so that an object of certain type can be conceptualized and then extended further. For example, take the concept of round shapes in geometry. The roundness is a property that is present in many different forms in real objects. The circle, spheres, and, elliptical objects have roundness. However, the concept roundness is a bit abstract, so if we wrote a class to capture this concept, we may never be able to truly find a real object for which the class Roundness can be a template. The concepts and behaviors, that apply to other objects, but are too abstract to make concrete, are often written in the form of abstract classes. In java, the abstract classes have following main properties.

**Abstract Class**

1. The abstract class has at least one abstract method. An abstract method is a method, which has no body or definition. It just has a signature and return type.
2. The usefulness of abstract class lies in it being extended by other classes or other classes being derived from it.
3. An abstract class cannot be instantiated, though it can be used as a reference type for the sub class objects.

**FIG. 10.24**

Example of an abstract class and its extension.
Let us consider the following inheritance hierarchy (Figure 10.25).
A shape is a general abstract concept. We know that a shape can have area and in some cases volume. A shape certainly has a name. However, shape is abstract enough that we can define an abstract class called Shape to model it. We would never need an instance of class Shape because there is no real object in the world that is called “Shape”.

Now there can be many types of shapes. A point is a shape, which is infinitesimal in size. However, it has properties such as coordinates x and y in the Cartesian system. Therefore, the Class Point can extend the general concept of Shape to a concrete form. Hence, we derive the class Point from Shape.

A Square has a center point just like a Point and a class Square can inherit the coordinates x and y from class Point. Nevertheless, Square shape has additional property like a length or edge. Therefore, we derive the class Square from Point.

Finally, a Cube has the same properties or fields as the Square does – like a central point, and an edge, but has a volume, which square does not have. Yet, no new fields are needed if we were to derive the class Cube from class Square.

**Converting the Inheritance Design Into The Source Code:**
Now we convert the above design into source code, first by writing various classes in the inheritance hierarchy and then writing a driver class program to test them.
The class `Shape` below gives the code for the abstract class `Shape`.

```java
public abstract class Shape
{
    public double area()
    {
        return 0;
    }

    public double volume()
    {
        return 0;
    }

    public abstract String getName();
}
```

**Listing 10.12**

The class `Shape` is an abstract class because it has one un-implemented (abstract) method `getName()`. The syntax requires that the keyword “abstract” be included in the declaration of method signature.

A class can also become abstract if it derives from an abstract class and does not provide implementation for the abstract methods for its parent. The keyword `abstract` would be then required in class declaration.
The source code for the class Point is given below in the Listing 10.13

```java
public class Point extends Shape {
    protected int x;
    protected int y;

    public Point() {
        this(0);
    }

    public Point(int a) {
        this(a, 0);
    }

    public Point(int a, int b) {
        this.x = a;
        this.y = b;
    }

    public String getName() {
        return "Point";
    }

    public void setPoint(int a, int b) {
        this.x = a;
        this.y = b;
    }

    public void setX(int x1) {
        this.x = x1;
    }

    public void setY(int y1) {
        this.y = y1;
    }

    public int getX() {
        return this.x;
    }
}
```

**class Point**

Fields are x and y coordinates for the Point.

Default and explicit constructors, all chained together.

abstract method from the parent class Shape implemented.

Other methods of class Point.
class Square
The source code for the class Square is given below in the Listing 10.14.

```java
public class Square extends Point {
    protected double edge;

    public Square() {
        this(0.0);
    }

    public Square(double edge1) {
        this(edge1, 0, 0);
    }

    public Square(double edge1, int a) {
        this(edge1, a, 0);
    }

    public Square(double m_edge, int a, int b) {
        super(a, b);
        setEdge(m_edge);
    }

    public void setEdge(double m_edge) {
    }
}
```

New field edge is added. It inherits x and y.

Constructors

Methods

Overrides the toString method from Object class.

Comment [p4]: Figure 10.25
this.edge = (m_edge >= 0 ? m_edge : 0);
}

public double getEdge()
{
    return this.edge;
}

public double area()
{
    return this.edge*this.edge;
}

public String toString()
{
    return "center = " + super.toString() + "; Edge = " + this.edge;
}

public String getName()
{
    return "Square";
}

}//End of class Square

Listing 10.14

class Cube
The source code for the class Square is given below in the Listing 10.15. Does not
need any new fields.

public class Cube extends Square
{
    public Cube()
    {
        this(0,0);
    }
    public Cube(double edge1)
    {
        this(edge1,0,0);
    }
    public Cube(double edge1, int a)
    {
        this(edge1, a, 0);
    }
    public Cube(double m_edge, int a, int b)
public double volume()
{
    return this.edge*this.edge*this.edge;
}

public double area()
{
    return 6*this.edge*this.edge;
}

public String toString()
{
    return super.toString();
}

public String getName()
{
    return "Cube";
}

//end of class Cube

Listing 10.15

The UML (Unified Modeling Language) diagram for all these classes is given on next page (Figure 10.26).
Testing of Classes in Figure 10.26
Listing 10.16 provides a main method to test the classes in Figure 10.26, where the class on the top of hierarchy, Shape, is an abstract class.

```java
00001 import javax.swing.JOptionPane;
00002 import java.text.*;
00003
00004 public class TestShapes
00005 {
```
```java
public static void main(String[] args) {
    // Initialize fields
    boolean done = false;
    String Input = "";
    String Output = "";

    while(!done) {
        Input = JOptionPane.showInputDialog("Please enter the "+"x coordinate of the shape.\n");
        int x_cord = Integer.parseInt(Input);
        Input = JOptionPane.showInputDialog("Please enter the "+"y coordinate of the shape.\n");
        int y_cord = Integer.parseInt(Input);
        Input = JOptionPane.showInputDialog("Please enter the edge of the shape. Enter 0 for the point.\n");
        double edge = new Double(Input).doubleValue();
        Shape Gen_Shape = null;
        if(edge == 0) {
            Gen_Shape = new Point(x_cord,y_cord);
            Output += Gen_Shape.getName():" + Gen_Shape.toString():" ; "
            +"Area of Point = " + precision2.format(Gen_Shape.area())+" ; "
            +"Volume of Point = " + precision2.format(Gen_Shape.volume());
        }
        else if("S".equals(Input)) {
            Gen_Shape = new Square(edge,x_cord,y_cord);
            Output += Gen_Shape.getName():" + Gen_Shape.toString():" ; "
            +"Square Area = " + precision2.format(Gen_Shape.area())+" ; "
            +"Square Volume = " + precision2.format(Gen_Shape.volume());
        }
        else if("C".equals(Input)) {
            Gen_Shape = new Cube(edge,x_cord,y_cord);
            Output += Gen_Shape.getName():" + Gen_Shape.toString():" ; "
            +"Cube Area = " + precision2.format(Gen_Shape.area())+" ; "
            +"Cube Volume = " + precision2.format(Gen_Shape.volume());
        }
        else {
            done = true;
        }
    }
}
```
The purpose of this program is to get from the user the properties of shapes and then display properties, such as central coordinates of shape, and if relevant then display its area and volume. The coordinates of the center of a Shape, be it a point, square or cube would not change with its type. In Lines (L10.16#15-20), user input for Shape’s center x, y coordinates are accepted and parsed. The user is prompted to input the edge of the Shape or zero if the Shape is a point (L10.16#21-23).

Understand that all instantiable objects such as Point, Square and Cube are derived from super class Shape. Therefore the reference type Shape can hold the address of any of them. We declare a general Shape type reference Gen_Shape (L10.16#24). If data for edge entered by the user were zero, then it is certain that the shape is a point. In that case the first if block is executed (L10.16#24-31). Inside the if block the final point object is constructed with user provided center coordinates x_coord and y_coord (L10.16#27), and a super class reference Gen_Shape points to this derived class Point object. An Output String is built to describe the properties of this Shape object, first by getting its name, and central coordinates by calling member methods getName and toString (L10.16#28). Then to add its area and volume to the overall description, the methods area and volume are called (L10.16#29-30). Understand that class Point does not actually have the coded area and volume methods, and it automatically binds to its super class Shape’s implemented methods area and volume for their execution. The JOptionPane outside the else block displays the properties of the Shape (L10.16#52). Figure 10.27 shows the system of displays when the user constructs a Point shape.
FIG. 10.27
User is given, through the display, all the correct properties of Point type object constructed by them (Figure 10.27). Notice that even though point object created on Line (L10.16#27) was represented by a Shape type reference (Gen_Shape), due to the property of dynamic binding, Java correctly recognizes the object to be of Point type and only calls the methods from Point class to build the property String Output.

If edge value selected is not zero then user may wish to construct either a square or a cube (both these shapes having an edge). In that case the else block in Listing 10.16 (L10.16#32-50) is executed. And if after entering the center coordinates of 10,
and 10, the user also entered an edge value of 10 for a cube object, the displays of Figure 10.28 print the properties of Cube constructed by the user.

FIG. 10.28
Here also though reference of type Shape (Gen_Shape) holds the address of Cube object created at line (L10.16#45), Java does the correct dynamic binding by calling the appropriate methods for area and volume for the Shape object Cube. We shall soon discuss advantages of extending other Java abstract classes called adapter classes, which implement some methods for a Java interface but leave one of more methods unimplemented. Adapter classes simplifies coding in creating some graphical user interface programs.