Inheritance

• Describe a new class in terms of how it differs from a previously defined class.

• Promotes modularity of programs.

• Promotes reuse of modules from one application to another.

Abstract Classes

• An abstract class is a partially specified class.
• For some methods only the signature, but no implementation, is provided.
• An abstract class may not be instantiated
• The \texttt{new} operator may not be used to create instances of an abstract class.
• Another class may extend an abstract class.
• If the extended class is complete (not abstract) it may be instantiated.
FacesApplication

- Define classes to implement drawings of faces in various moods.
- Start with a basic `Face` class.
- Leave the mouth shape unspecified.
- Define `HappyFace` as a `Face` with a smiling mouth.
- Define `SadFace` as a `Face` with a frowning face.
- Define `AlertFace` to have special eyes.
- Define `ScarFace` to have an additional feature.

```java
package facesapplication;
import acm.graphics.*;

public abstract class Face extends GCompound {

    protected static final double LEFT = 0;
    protected static final double TOP = 0;
    protected static final double HEIGHT = 600;
    protected static final double WIDTH = 600;

    protected GObject head;
    protected GObject nose;
    protected GObject leftEye;
    protected GObject rightEye;
    protected GObject mouth;

    public Face() { this(LEFT, TOP, WIDTH, HEIGHT);  }
    public Face(double left, double top, double width, double height)  {
        makeObjects(left,top,height,width);
        addObjects();
    }
    // ... Omitted ...
}
```

The `Face` class is intended to represent faces in various moods. The class is abstract. Some methods will be declared, but not implemented.

```java
    void makeObjects(double left, double top, double width, double height) {
        head = makeHead(left,top,HEIGHT);
        double centerX = left+width/2;
        double centerY = top+height/2;
        nose = makeNose(centerX-width/10,centerY-height/10,width/5,height/5);
        leftEye = makeEye(centerX-3*width/10,centerY-3*width/10,width/5,height/5);
        rightEye = makeEye(centerX+1*width/10,centerY-3*width/10,width/5,height/5);
        mouth = makeMouth(left+width/5,top+2*height/3,3*width/5,height/6);
    }
    void addObjects() {
        add(head);
        add(nose);
        add(leftEye);
        add(rightEye);
        add(mouth);
    }
```

These helpers are called by the `Face` constructor. The parameters to `makeObjects` define a rectangle in which the face will be placed. This method in turn calls helpers to make the nose, eyes and mouth, passing each a description of the rectangle in which the object should be placed.
The head, nose and eyes are defined (respectively) by the `makeHead`, `makeNose`, `makeEye` helper methods, which are called by the `Face` constructor. The `makeMouth` method is left undefined. It is labeled "abstract". It will be defined in classes derived from `Face` via inheritance.

The `HappyFace` class extends the `Face` class. The `HappyFace` constructor uses the `super` keyword to invoke its base class (`Face`) constructor. The `makeMouth` method (left abstract in `Face`) is defined in `HappyFace` to return a `GArc` object in the shape of a smile.

The `SadFace` class extends the `Face` class. The `SadFace` constructor uses the `super` keyword to invoke its base class (`Face`) constructor. The `makeMouth` method (left abstract in `Face`) is defined in `SadFace` to return a `GArc` object in the shape of a frown.
package facesapplication;
import acm.graphics.*;

public class AlertFace extends HappyFace {
    public AlertFace(double left, double top, double width, double height) {
        super(left, top, width, height);
    }
    GObject makeEye(double left, double top, double width, double height) {
        return new Eye(left, top, width, height);
    }
}

The AlertFace class extends the HappyFace class. The AlertFace constructor uses the super keyword to invoke its base class (HappyFace) constructor. It includes a new definition of makeEye, which was defined previously in HappyFace. The new definition (in AlertFace) overrides the old one (in HappyFace).

class Eye extends GCompound {
    protected GObject eyeBall;
    protected GObject iris;
    public Eye(double left, double top, double width, double height) {
        eyeBall = new GOval(left, top, width, height);
        add(eyeBall);
        GOval temp = new GOval(left+width/4, top+height/4, width/2, height/2);
        temp.setFilled(true);
        iris = temp;
        add(iris);
    }
}

The Eye class implements an object representing the eye of an AlertFace.

package facesapplication;
import acm.graphics.*;

public class ScarFace extends SadFace {
    protected GObject scar;
    public ScarFace(double left, double top, double width, double height) {
        super(left, top, width, height);
    }
    void makeObjects(double left, double top, double width, double height) {
        super.makeObjects(left, top, width, height);
        scar = makeScar(left+3*width/4, top+height/2, width/8, height/10);
    }
    void addObjects() {
        super.addObjects();
        add(scar);
    }
    GObject makeScar(double left, double top, double width, double height) {
        return new GLine(left+width, top, left, top+height);
    }
}

The ScarFace class extends the SadFace class. It includes a new definition of makeObjects, which was defined previously in Face. The new definition of makeObjects uses the super keyword to invoke the inherited version of this method. Then it calls a helper to add an additional feature to the object.
package facesapplication;
import acm.program.*;

public class FacesApplication extends GraphicsProgram {
    public void run() {
        Face happyFace = new HappyFace(25,25,250,250);
        add(happyFace);
        Face sadFace = new SadFace(325,25,250,250);
        add(sadFace);
        Face alertFace = new AlertFace(25,325,250,250);
        add(alertFace);
        Face scarFace = new ScarFace(325,325,250,250);
        add(scarFace);
    }
    public static void main(String[] args) {
        new FacesApplication().start(args);
    }
}

The FacesApplication constructs and displays a HappyFace, a SadFace, an AlertFace and a ScarFace.

Inheritance Hierarchy
HouseApplication

- Design a class hierarchy for pictures of houses.
- **House**: A rectangle with a triangle on top.
- **FancyHouse**: A house with a garage.
- **OpenHouse**: A house with windows.
- **[FancyOpenHouse]**: A house with a garage and windows.

House Hierarchy

```
    House
     /\       \
    /  \      \      
FancyHouse  OpenHouse
```

House Hierarchy

- **House**: A `GCompound` including a rectangle base and a polygon forming a roof.
- **Open House**: Just like a House, but with windows.
- **Fancy House**: Just like a house, but with a garage.
package house;
import acm.graphics.*;
import java.awt.*;
public class House extends GCompound {
    public House(int left, int bottom, int width, int height, Color color) {
        super();
        int top = bottom - height;
        int right = left + width;
        int midPointX = left + width / 2;
        int peakY = top - height;
        GRect base = new GRect(left, top, width, height);
        base.setFillColor(color);
        base.setFilled(true);
        add(base);
        GPolygon roof = new GPolygon();
        roof.addVertex(left, top);  
        roof.addVertex(midPointX, peakY);
        roof.addVertex(right, top);
        roof.setFillColor(color);
        roof.setFilled(true);
        add(roof);
    }
}

The House constructor creates a GRect base and a GPolygon roof inside the rectangle defined by the constructor parameters. Notice that we start by calling the base class constructor.

package house;
import java.awt.*;
public class FancyHouse extends House {
    public FancyHouse(int left, int bottom, int width, int height, Color color) {
        super(left, bottom, width, height, color);
        House garage = new House(left + width, bottom, width / 2, height / 2, color);
        add(garage);
    }
}

The FancyHouse constructor invokes its base class constructor (to create the base and roof of the house) and then creates a whole new (smaller) House for the garage.

package house;
import acm.graphics.*;
import java.awt.*;
public class OpenHouse extends House {
    public OpenHouse(int left, int bottom, int width, int height, Color color, Color wColor) {
        super(left, bottom, width, height, color);
        int wWidth = width / 6;  int wHeight = height / 6;
        int lX = left + wWidth;  int rX = left + 4 * wWidth;
        int bY = bottom - 2 * wHeight;  int tY = bottom - 5 * wHeight;
        GRect w1 = new GRect(lX, bY, wWidth, wHeight);
        w1.setFilled(true); w1.setFillColor(wColor);
        add(w1);
        GRect w2 = new GRect(rX, bY, wWidth, wHeight);
        w2.setFilled(true); w2.setFillColor(wColor);
        add(w2);
        GRect w3 = new GRect(lX, tY, wWidth, wHeight);
        w3.setFilled(true); w3.setFillColor(wColor);
        add(w3);
        GRect w4 = new GRect(rX, tY, wWidth, wHeight);
        w4.setFilled(true);
        w4.setFillColor(wColor);
        add(w4);
    }
}

The OpenHouse constructor invokes its base class constructor (to create the base and roof of the house) and then creates GRect objects for the windows.
package housesapplication;
import acm.program.*;
import java.awt.*;
import house.*;

public class HousesApplication extends GraphicsProgram {
    public void run() {
        House house = new House(100, 300, 100, 100, Color.RED);
        add(house);

        FancyHouse fancyHouse = new FancyHouse(300, 300, 100, 100, Color.ORANGE);
        add(fancyHouse);

        OpenHouse openHouse = new OpenHouse(500, 300, 100, 100,
                Color.GREEN, Color.LIGHT_GRAY);
        add(openHouse);
    }

    public static void main(String[] args) {
        new HousesApplication().start();
    }
}