Objects and Classes

• Our approach to studying Java
  – We will use an “objects-first” approach
  – We will use the BlueJ development environment
  – You will also learn how to make programs run on any machine without using BlueJ

• Our pedagogical approach
  – Introduce a concept using an existing program
  – Manipulate the objects in the program to understand the available methods
  – Study the source code for the program
  – Expand the functionality of the program
The Shapes Example

- The shapes are a *Circle*, *Square*, and *Triangle* that can be painted on a *Canvas*

- Things that you will learn
  - Creating one or more instances of an object
  - Calling a method
  - Passing parameter values to a method
  - Different data types
  - Signature of a method
  - The state of an object
  - Interaction between objects
  - Returning values from an object
  - Using objects as parameter values
Activities you will complete - 1

• Learn how to manipulate the objects in the shapes class
• Develop a sequence of commands to draw a picture of a “house”
• Put this sequence of commands into a main method so your program can run outside of BlueJ
• Study the source code for the Shapes class
• Perform some simple animations
Activities you will complete – 2

• Learn about a non-graphical class called Student that uses a text window to display results
• Show how output data can be displayed in a text window
• Show how methods can return data values
• Manipulate data for the student example
• Study the source code for the Student class
• Extend the functionality for the student example
Visit the BlueJ Development Environment

• We will visit BlueJ as a class with everyone having a chance to manipulate some of the geometric shapes

• Make sure you understand how to:
  – Create an instance of an object
  – Call methods on each instance separately
  – Supply parameters to a method, if required, to customize the method
  – Examine the current state of each instance of an object and find out the values of the fields associated with the instance
Attributes Associated With Each Shape

• xPosition – an integer value for the horizontal screen coordinate specified in pixels
• yPosition – an integer value for the vertical screen coordinate specified in pixels
• color – the color of an object represented as a String, such as “red”
• isVisible – a boolean value (true or false) specifying whether the object is currently visible or not
Ways You Can Move a Shape

• `moveRight` or `moveLeft` by a fixed distance
• `moveUp` or `moveDown` by a fixed distance
• `moveHorizontal` or `moveVertical` by a specified distance, the sign of the distance value indicates the direction of movement
• `slowMoveHorizontal` or `slowMoveVertical`
  – The object appears to move the specified distance but does this “slowly” rather than one big jump
  – This is accomplished visually by drawing the object, erasing the object, moving a small fixed distance, redrawing the object, and repeating this process until the desired traversal is completed
Ways to Change an Object’s Appearance

• makeVisible or makeInvisible – determines whether you will see the object on the canvas or not

• changeSize
  – Make an object appear to be bigger or smaller
  – The size parameters will depend on the type of object being displayed

• changeColor – the parameter is a String value giving the new color name, as in “blue”

• erase – removes the object from the canvas
Using BlueJ on Your Own

- You are to draw a picture of a house with a sun overhead, something like the one shown below, but I am sure you can be more imaginative.
- When you are happy with your picture of the house, you should write down the precise sequence of method calls that produced your picture.
- Following the directions on the lab description provided, store your method calls to draw your house in the “main” method and export your file out of BlueJ to be run independently. Compile and run your program.
import java.awt.*;
import java.awt.geom.*;
java.awt.geom.*;
public class Circle{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new circle at default position with default color.
     */
    public Circle()
    {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }

    // methods go here
}
/**
 * Make this circle visible. If it was already visible, do nothing.
 */
public void makeVisible()
{
    isVisible = true;
    draw();
}
/**
 * Make this circle invisible. If it was already invisible, do nothing.
 */
public void makeInvisible()
{
    erase();
    isVisible = false;
}
/**
 * Move the circle a few pixels to the right.
 */
public void moveRight()
{
    moveHorizontal(20);
    }
/**
 * Move the circle a few pixels to the left.
 */
public void moveLeft()
{
    moveHorizontal(-20);
}
/**
 * Move the circle a few pixels up.
 */
public void moveUp()
{
    moveVertical(-20);
}
/**
 * Move the circle a few pixels down.
 */
public void moveDown()
{
    moveVertical(20);
}
/**
 * Move the circle horizontally by 'distance' pixels.
 */
public void moveHorizontal(int distance)
{
    erase();
    xPosition += distance;
    draw();
}
/**
 * Move the circle vertically by 'distance' pixels.
 */
public void moveVertical(int distance)
{
    erase();
    yPosition += distance;
    draw();
}
/**
 * Slowly move the circle horizontally by 'distance' pixels.
 */

public void slowMoveHorizontal(int distance)
{
    int delta;

    if(distance < 0)
    {
        delta = -1;
        distance = -distance;
    }
    else
    {
        delta = 1;
    }

    for(int i = 0; i < distance; i++)
    {
        xPosition += delta;
        draw();
    }
}
/**
 * Change the size to the new size (in pixels). Size must be >= 0.
 */

public void changeSize(int newDiameter)
{
    erase();
    diameter = newDiameter;
    draw();
}

/**
 * Change the color. Valid colors are "red", "yellow", "blue", "green",
 * "magenta" and "black".
 */

public void changeColor(String newColor)
{
    color = newColor;
    draw();
}
/**
 * Draw the circle with current specifications on screen.
 */
private void draw()
{
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color, new Ellipse2D.Double(xPosition, yPosition, diameter, diameter));
        canvas.wait(10);
    }
}

/**
 * Erase the circle on screen.
 */
private void erase()
{
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.erase(this);
    }
}
Types and Signatures

• Each data type has a set of allowed values
  – An int is from the set \{ \ldots , -3, -2, -1, 0, 1, 2, 3, \ldots \} 
  – A String is any sequence of characters, such as “Hello”, and includes the empty sequence, “”
  – A boolean can only be true or false

• Signature of a method (or a constructor)
  – The name of the method and the sequence of types (left to right) in the parameter list 
  – method1(String) is different than method1(int)
  – method2(String, int) is different than method2(int, String)
Other Classes

• The classes for *Square* and *Triangle* are very similar and will not be repeated here
• The class *Canvas* will not be discussed at this time
• Here are some of the things we have learned from looking at the code for the *Circle* class
  – We can import from other classes, such as the package java.awt.geom, documentation can be found online
  – A constructor initializes the object when created
  – The coordinate system is in screen pixels
  – (preview) an if statement selected between alternatives
  – (preview) a for statement causes a sequence of statements to be repeated
Adding a Picture class

- The picture will contain a very simple “house” and a sun in the sky
- The picture will be constructed by using method calls to the Square, Triangle, and Circle classes
- These method calls will be part of a method in Picture called draw
- There will be two other methods in Picture
  - setBlackAndWhite that only uses black & white
  - setColor that puts the color back in the picture
Visit the BlueJ Development Environment

- Experiment with the Picture class
- What do you think is happening inside the draw method?
- Try changing colors by calling the methods setBlackAndWhite and setColor
- How do you think the change color is being accomplished?
- Double click on the Picture class icon to display the source code
- Add code at the end of the draw method to make the sun sink to the horizon and then disappear
/**
 * This class represents a simple picture. You can draw the picture using
 * the draw method. But wait, there's more: being an electronic picture, it
 * can be changed. You can set it to black-and-white display and back to
 * colors (only after it's been drawn, of course).
 */

class Picture
{
    private Square wall;
    private Square window;
    private Triangle roof;
    private Circle sun;

    /**
     * Constructor for objects of class Picture
     */
    public Picture()
    {
        // nothing to do... instance variables are automatically set to null
    }

    // the methods go here
}
/**
 * Draw this picture.
 */

public void draw()
{
    wall = new Square();
    wall moveVertical(80);
    wall changeSize(100);
    wall makeVisible();

    window = new Square();
    window changeColor("black");
    window moveHorizontal(20);
    window moveVertical(100);
    window makeVisible();

    roof = new Triangle();
    roof changeSize(50, 140);
    roof moveHorizontal(60);
    roof moveVertical(70);
    roof makeVisible();

    sun = new Circle();
    sun changeColor("yellow");
    sun moveHorizontal(180);
    sun moveVertical(-10);
    sun changeSize(60);
    sun makeVisible();
}
/**
 * Change this picture to black/white display
 */
public void setBlackAndWhite()
{
    if(wall != null)    // only if it's painted already...
    {
        wall.changeColor("black");
        window.changeColor("white");
        roof.changeColor("black");
        sun.changeColor("black");
    }
}

/**
 * Change this picture to use color display
 */
public void setColor()
{
    if(wall != null)    // only if it's painted already...
    {
        wall.changeColor("red");
        window.changeColor("black");
        roof.changeColor("green");
        sun.changeColor("yellow");
    }
}
The Sun and the Moon

• Cut and paste the code you developed to draw your house from your previous program to the draw method in Picture
• Verify that your house draws correctly
• Add additional code that
  – Make the sun set and then disappear, change the colors to black and white
  – Make the moon appear on the opposite side of the house at the horizon, rise above the house, move across above the house, and then move back down to the horizon
  – Make the sun appear on the opposite side, change to full color, and make the sun rise in the sky
Students and Lab Classes

• This is a non-graphical example that involves text with output displayed in a text window

• There are two objects
  – *Student* with the attributes of name (a String type), id (a String type), and credits (an int type)
  – *LabClass* with attributes *instructor* (a String type), *room* (a String type), *timeAndDay* (a String type), *students* (a built-in collections called a List), and *capacity* (an int type)

• There can be multiple instances of each object
  – There can be multiple students in the same lab
  – There can be multiple labs, some students might be in several labs
Methods in the Student Class

• accessor and mutator for name
  – getName - accessor returns the String value of the name
  – changeName - mutator that allows the name to be changed

• The ID field only has an accessor method getStudentID

• Methods involving credits
  – the accessor method getCredits
  – Credits can only be changed by adding more credits to the current total using the method addCredits

• A login name is a combination of the first 4 characters of the student's name and the first 3 characters of the student's ID number; it is accessed using getLoginName

• The method print outputs the student's name and ID number to the output terminal
Methods in the ClassLab Class

• The `enrolStudent` method accepts an instance of the student class as a parameter and adds this instance to the list of students in the lab

• `numberOfStudents` returns the number of students currently enrolled in the lab

• Some “setter” or mutator methods
  – `setRoom` has a single String parameter
  – `setTime` has a single String parameter
  – `setInstructor` has a single String parameter

• The method `printList` outputs the time and day, the instructor, and the complete list of enrolled students to the output terminal
public class Student
{
    // the student's full name
    private String name;
    // the student ID
    private String id;
    // the amount of credits for study taken so far
    private int credits;

    /**
     * Create a new student with a given name and ID number.
     */
    public Student(String fullName, String studentID)
    {
        name = fullName;
        id = studentID;
        credits = 0;
    }
    // methods go here
}
/**
 * Return the full name of this student.
 */
public String getName()
{
    return name;
}

/**
 * Set a new name for this student.
 */
public void changeName(String replacementName)
{
    name = replacementName;
}

/**
 * Return the student ID of this student.
 */
public String getStudentID()
{
    return id;
}
public void addCredits(int additionalPoints)
{
    credits += additionalPoints;
}

public int getCredits()
{
    return credits;
}

public String getLoginName()
{
    return name.substring(0,4) + id.substring(0,3);
}

public void print()
{
    System.out.println(name + "(" + id + ")");
}
import java.util.*;
/**
 * The LabClass class represents an enrolment list for one lab class. It stores
 * the time, room and participants of the lab, as well as the instructor's name.
 */
public class LabClass
{
    private String instructor;
    private String room;
    private String timeAndDay;
    private List students;
    private int capacity;

    /**
     * Create a LabClass with a maximum number of enrolments. All other details
     * are set to default values.
     */
    public LabClass(int maxNumberOfStudents)
    {
        instructor = "unknown";
        room = "unknown";
        timeAndDay = "unknown";
        students = new ArrayList();
        capacity = maxNumberOfStudents;
    }
    // methods go here
}
/**
 * Add a student to this LabClass.
 */

public void enrolStudent(Student newStudent)
{
    if(students.size() == capacity) {
        System.out.println("The class is full, you cannot enrol.");
    }
    else {
        students.add(newStudent);
    }
}

/**
 * Return the number of students currently enrolled in this LabClass.
 */

public int numberOfStudents()
{
    return students.size();
}

/**
 * Set the room number for this LabClass.
 */

public void setRoom(String roomNumber)
{
    room = roomNumber;
}
/**
 * Set the time for this LabClass. The parameter should define the day
 * and the time of day, such as "Friday, 10am".
 */
public void setTime(String timeAndDayString) {
    timeAndDay = timeAndDayString;
}
/**
 * Set the name of the instructor for this LabClass.
 */
public void setInstructor(String instructorName) {
    instructor = instructorName;
}
/**
 * Print out a class list with other LabClass details to the standard
 * terminal.
 */
public void printList() {
    System.out.println("Lab class " + timeAndDay);
    System.out.println("Instructor: " + instructor + "   room: " + room);
    System.out.println("Class list:");
    Iterator i = students.iterator();
    while(i.hasNext()) {
        Student student = (Student)i.next();
        student.print();
    }
    System.out.println("Number of students: " + numberOfStudents());
}
Exercises with LabClass and Student

• Do exercises 1.23, 1.24, and 1.25 on page 15 of the textbook

• Add two more fields in the Student class to store the major the student has declared (e.g., “Computer Science”) and the college for that major (e.g., “Arts and Sciences”)

• Add get and set methods for these two fields

• Change the print method in Student so that this additional information is printed in the class list