CLASSES AND OOP

OVERVIEW

OVERVIEW

- **In this section, we will see how to design, implement and use classes in object oriented programs**

- **What is a class?**
  - A class is a user defined abstract data type (ADT) that contain variables (called attributes) and a collection of operations on these variables (called methods)
  - The primary advantage of classes is that they give us a natural way to create robust and reliable code that can be reused in a wide range of applications

OVERVIEW

- **A class is normally created by one programmer and used by many other programmers**
  - Only the creator needs to know implementation details
  - Users can ignore details and build code on top of the class
  - This allows teams of programmers to work on separate classes to build very large and complex applications

- **Class libraries**
  - Java contains over 4000 general purpose class libraries that can be used in any program
  - We have already been using the String, Scanner, and System.out, Math, Arrays classes in our programs

OVERVIEW

- **To design a class**
  - Select appropriate names and data types for the data fields
  - Decide on names and parameters for the class methods
  - This defines the user interface for the class
- **To implement a class**
  - Implement constructor methods to initialize data fields
  - Implement other methods to perform data operations
- **To use a class**
  - Declare objects of the class
  - Call methods on these objects

OVERVIEW

- **Lesson objectives:**
  - Learn how to create and use simple classes
  - Learn how to create and use composite classes
  - Study example programs with classes
  - Complete online labs on classes
  - Complete programming project using classes

CLASSES

PART 1

DESIGNING CLASSES

DESIGNING CLASSES

- **The main purpose of a class is to bundle together the data and operations that make up an abstract data type (ADT)**
  - We must declare variables to store the data fields that make up the abstract data type
  - We must declare methods to implement all operations that are possible on these data fields

- **We must also specify how the class can be used**
  - We must specify which of the variables and methods are public and can be accessed directly by users of this class
  - We must also specify which of the variables and methods are private and hidden from users of this class

DESIGNING CLASSES

- **Overview of Java's class syntax**

```
public class class_name
{
   // Private variables
   private data_type variable_name;
   private data_type variable_name;
   …
```

We give the name of the class here

DESIGNING CLASSES

- **Overview of Java's class syntax**

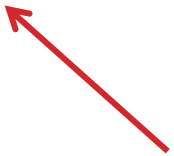
```
public class class_name
{
   // Private variables
   private data_type variable_name;
   private data_type variable_name;
   …
```

These variable declarations define the data fields inside the class that make up the abstract data type

DESIGNING CLASSES

- **Overview of Java's class syntax**

```
public class class_name
{
   // Private variables
   private data_type variable_name;
   private data_type variable_name;
   …
```



The keyword private says that these variables are hidden from users of the class can can not be accessed directly

DESIGNING CLASSES

First we declare constructor methods that initialize the private data fields

```
…
   // Constructors
   public class_name() {  }
   public class_name( parameter_list ) {  }

   // Methods
   public return_type method_name( parameter_list ) {  }
   public return_type method_name( parameter_list ) {  }
   public return_type method_name( parameter_list ) {  }
}
```

DESIGNING CLASSES

Next we declare the public methods that implement operations on the data fields

```
…
   // Constructors
   public class_name() {  }
   public class_name( parameter_list ) {  }

   // Methods
   public return_type method_name( parameter_list ) {  }
   public return_type method_name( parameter_list ) {  }
   public return_type method_name( parameter_list ) {  }
}
```

DESIGNING CLASSES

For now the implementation of these methods are empty. We can fill in the code later.

```
…
   // Constructors
   public class_name() {  }
   public class_name( parameter_list ) {  }

   // Methods
   public return_type method_name( parameter_list ) {  }
   public return_type method_name( parameter_list ) {  }
   public return_type method_name( parameter_list ) {  }
}
```

TIME CLASS EXAMPLE

- **Consider the problem of keeping track of the time of day in a program**
  - We need an integer hour value [0..23]
  - We need an integer minute value [0..59]
  - We need an integer second value [0..59]

- **We need to operations that safely manipulate the hour, minute, second values**
  - Provide methods to access/modify time values
  - Provide methods to input/output time values
  - Make sure the user can not create invalid times

TIME CLASS EXAMPLE

- **First we declare private variables to store the data fields inside the Time class**
  - Use integers for hour, minute, second values
- **Then we declare the constructor methods and all public methods of the Time class**
  - Use “get” methods to access each of the Time data fields
  - Use “set” methods to modify each of the Time data fields
  - Use “read” method to input all Time data fields
  - Use “print” method to output all Time data fields

TIME CLASS EXAMPLE

- **Where do we put the Time class?**
  - By convention, the declaration of a class is placed in a java file with the same name as the class
  - For example, the Time class would be stored in Time.java

- **How can we use the Time class in our program?**
  - We simply put the Time.java file in the same folder as our program and compile both using “javac” or similar tool
  - The Java run time system will automatically combine the Java class files when you run the program

TIME CLASS EXAMPLE

```
public class Time
{
   // Private variables
   private int hour;
   private int minute;
   private int second;
…
```

These variable declarations define the data fields inside the Time class

TIME CLASS EXAMPLE

```
…
  // Constructors
  public Time() {  }
  public Time(int h, int m, int s) {  }

  // Setter methods
  public void setHour(int h) {  }
  public void setMinute(int m) {  }
  public void setSecond(int s) {  }
…
```

The constructor methods will initialize the data fields

TIME CLASS EXAMPLE

```
…
   // Constructors
   public Time() {  }
   public Time(int h, int m, int s) {  }

   // Setter methods
   public void setHour(int h) {  }
   public void setMinute(int m) {  }
   public void setSecond(int s) {  }
…
```

The setter methods will let users change the data fields. The name of the data field is normally part of the method name

TIME CLASS EXAMPLE

```
…
   // Getter methods
   public int getHour() {  }
   public int getMinute() {  }
   public int getMinute() {  }

   // Other methods
   public void read() {  }
   public void print() {  }
}
```

The getter methods will let users access the values of data fields. The name of the data field is normally part of the method name

TIME CLASS EXAMPLE

```
…
   // Getter methods
   public int getHour() {  }
   public int getMinute() {  }
   public int getMinute() {  }

   // Other methods
   public void read() {  }
   public void print() {  }
}
```

We define other methods that use or manipulate data fields here

TIME CLASS EXAMPLE

- **It is possible to extend the Time class in many ways**
  - Add more data fields (eg. days, microseconds)
  - Add more methods to manipulate Time values
  - A method to print time in military time
  - A method to compare two time values
  - A method to add H hours, M minutes, S seconds
  - A method to subtract H hours, M minutes, S seconds

SUMMARY

- **A Java class is used to bundle together data and operations that make up an abstract data type (ADT)**
  - Data fields are stored in class variables
  - Operations on this data are defined by methods

- **The class definition also tells us how to use a class**
  - public methods (and variables) can be accessed
  - private variables (and methods) are hidden from users
  - The Java compiler will give us error messages if we attempt to break these rules

CLASSES

PART 2

IMPLEMENTING CLASSES

IMPLEMENTING CLASSES

- **To complete the implementation of a Java class we must fill in the body of the methods we declared above**
  - We are allowed to access the method parameters and define local variables to perform calculations
  - We are also allowed to access and modify all of the private variables in this class

- **Information hiding**
  - We are not allowed to directly access or modify private variables from another class
  - Other classes are not allowed to directly access or modify the private variables in this class

IMPLEMENTING CLASSES

- **The first methods we implement are the constructors**
  - The default constructor has no parameters, so we have to choose a sensible default value for each private variable

```
public Time()
{
    hour = 0;
    minute = 0;
    second = 0;
}
```

IMPLEMENTING CLASSES

- **The first methods we implement are the constructors**
  - The second constructor typically has parameters to initialize each of the private variables

```
public Time(int h, int m, int s)
{
    hour = h;
    minute = m;
    second = s;
}
```

IMPLEMENTING CLASSES

- **Next we implement are the setters to store the parameter values in the private variables**
  - We can add error checking later to ensure that the values being stored are valid (e.g. minutes between 0..59)

```
public void setHour(int h)
{
    hour = h;
}

public void setMinute(int m)
{
    minute = m;
}
…
```

IMPLEMENTING CLASSES

- **Next we implement are the getters to return the current value of private variables**
  - The return type of each getter should match the data type of the corresponding private variable

```
public int getHour()
{
    return hour;
}

public int getMinute()
{
    return minute;
}
…
```

IMPLEMENTING CLASSES

- **Finally we implement the remaining methods in the class**

```
public void read()
{
   Scanner scnr = new Scanner(System.in);
   System.out.print("Enter hour: ");
   hour = scnr.nextInt();
   System.out.print("Enter minute: ");
   minute = scnr.nextInt();
   System.out.print("Enter second: ");
   second = scnr.nextInt();
}
```

IMPLEMENTING CLASSES

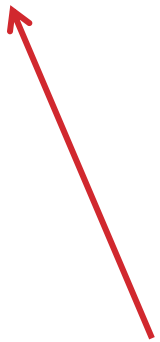
- **Finally we implement the remaining methods in the class**

```
public void print()
{
   // Basic output of time variables
   System.out.println("Hour: "  + hour);
   System.out.println("Minute: " + minute);
   System.out.println("Second: " + second);
}
```

IMPLEMENTING CLASSES

- **Finally we implement the remaining methods in the class**

```
public void print()
{
   // Formatted output of Time variables
   System.out.printf("%02d:%02d:%02d",
      hour, minute, second);
}
```



Each time variable is printed as an integer in a field 2 characters wide and with leading zeros

Example: 12:04:07

CODE DEMO

Time1.java

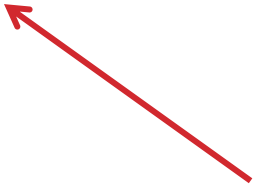
ERROR CHECKING

- **How should we test that a Time value is valid?**
  - We need to check that the hour, minute, second values are always within their expected 0..23, 0..59, 0..59 ranges

- **How should we correct invalid Time values?**
  - Simple solution uses modulo arithmetic to "wrap around" any overflow that occurs (10:66:90 becomes 10:06:30)
  - Fancy solution "wraps around and carries" the hour, minute, second values (10:66:90 becomes 11:07:30)
  - Change the hour/min/sec values to min/max value (10:66:90 becomes 10:59:59)

ERROR CHECKING

```
// Simple time validation
hour = hour % 24;
minute = minute % 60;
second = second % 60;
```

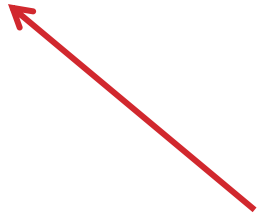


This method will “wrap around” any value overflows

It will not change valid hour, minute, second values

ERROR CHECKING

```
// Fancy time validation
minute = minute + second / 60;
second = second % 60;
hour = hour + minute / 60;
minute = minute % 60;
hour = hour % 24;
```



This method will “wrap around and carries” any value overflows

It will not change valid hour, minute, second values

ERROR CHECKING

- **Private variables can be updated in several places in the Time class (the constructor method and setter methods)**
  - We can put error checking code in a helper method
  - This helper method is not intended for users of this class, so we can make it a private method
  - We can call helper method in any method in the class

```
private void correctTime()
{
    // Put error checking code here
}
```

ERROR CHECKING

- **We call the helper function when Time value is set**

```
public Time(int h, int m, int s)
{
    hour = h;
    minute = m;
    second = s;
    correctTime();
}
```

This will fix any errors in hour, minute, second when the Time object is created

ERROR CHECKING

- **We also call helper function when Time value is changed**

```
public void setSecond(int s)
{
    second = s;
    correctTime();
}
```

This will fix any errors in hour, minute, second when the second value is changed

SUMMARY

- **To complete the implementation of Java class we fill in the bodies of all of the methods in the class**
  - Start with constructors and getters and setters
  - Then complete the read and print methods
  - Make sure these compile before working on other methods
  - Finally add error checking/correction after all of the basic operations are completed

- **Incremental development**
  - It is almost always faster and easier to edit, compile, debug methods one at a time than all at once

CLASSES

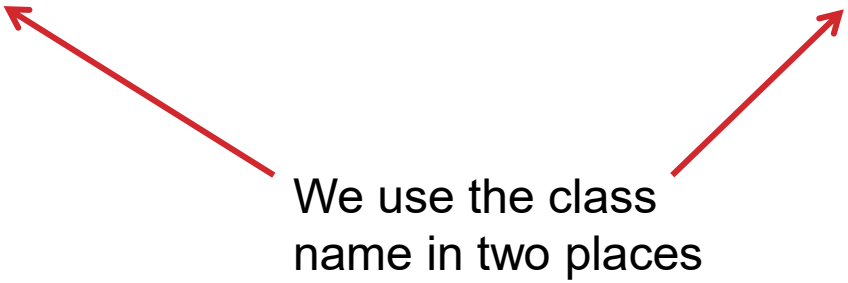
PART 3

USING CLASSES

CREATING OBJECTS

- **Java objects are essentially variables of the abstract data type we defined in our Java class**
  - In Java parlance, an object is an “instance of the class”

- **We create objects in Java as follows:**
  - class_name object_name = new class_name(params);



CREATING OBJECTS

- **Java objects are essentially variables of the abstract data type we defined in our Java class**
  - In Java parlance, an object is an “instance of the class”

- **We create objects in Java as follows:**
  - class_name object_name = new class_name(params);

CREATING OBJECTS

- **Time class example**

```
// Creating time1 = 00:00:00
Time time1 = new Time();

// Creating time2 = 01:02:03
Time time2 = new Time(1, 2, 3);
```

CALLING METHODS

- **To call a public method on an object, we must tell the Java compiler which object to send to the method**
  - This is done with the following “dot notation”
  - object_name.method_name(params);

- **Java will send the specified object into the method as an “implicit parameter”**
  - This allows the method to have access to the private data fields in the object

CALLING METHODS

- **The Java compiler will only let us access public methods**
  - You will get errors if you attempt to access the private data fields in the class using the dot notation

```
Time time3 = new Time(4,5,6);
time3.minute = 42;
```

This will cause a compiler error

CALLING METHODS

- **Time class example**

```
// Creating time1 = 00:00:00
Time time1 = new Time();

// Change time values
time1.setHour(11);
time1.setSecond(42);

// Print time values
time1.print();
System.out.println("minute = " + time1.getMinute());
```

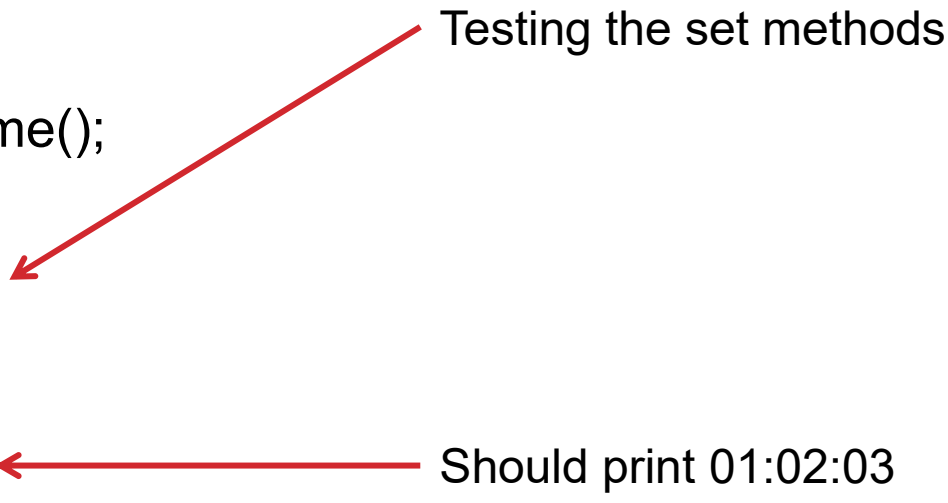
UNIT TESTING

- **You should always test all of the methods in a class before using it in another program**
  - One way to do this is to add a “unitTest” method in the class when it is being implemented
  - This method should be a static method (which does not have direct access to private variables of an object)
  - The “unitTest” method should call all methods with typical parameters, and verify that they are working correctly

UNIT TESTING

- **Time class example**

```
public static unitTest()
{
    Time time1 = new Time();
    time1.setHour(1);
    time1.setMinute(2);
    time1.setSecond(3);
    time1.print();
…
```



UNIT TESTING

Testing the get methods

```
…
   System.out.println(“hour = ” + time1.getHour());
   System.out.println(“minute = ” + time1.getMinute());
   System.out.println(“second = ” + time1.getSecond());

   Time time2 = new Time(1, 2, 321);
   time2.print();
…
```

Testing error correction in the constructor method

Should print 01:07:21

SUMMARY

- **In this section, we saw how to create Java objects and call methods using these objects**
  - Must use “dot notation” to call methods
  - We can only access public methods in a class
  - Java will stop us from using private variables directly

- **Unit testing is strongly recommended**
  - Test all of the methods with normal parameter values
  - Test error checking code by calling methods with abnormal parameter values

CLASSES

PART 3

SIMPLE CLASS EXAMPLES

SIMPLE CLASS EXAMPLES

- **The goal of object oriented programming is to create applications that build upon a collection of a classes**

- **There are three steps to this design process:**
  - Decide what information is needed to describe object
    - What private variables to declare
  - Decide what operations on the object are necessary
    - What public methods to create
  - Decide how to build applications using class
    - How to create and use objects in a program

SIMPLE CLASS EXAMPLES

- **In this section, we will illustrate object oriented programming by creating two simple classes:**
  - Student class
    - Stores basic information about a student
    - Very basic operations to access information
    - Could be used as part of large university database
  - Linear class
    - Store information about linear equations
    - Classic mathematical operations for linear equations
    - Could be used in an engineering application

STUDENT CLASS

- **What student information might be of interest?**
  - Student ID number (int)
  - First name, middle name, last name (string)
  - Home address, campus address (string)
  - ACT, SAT test scores (int)
  - Undergraduate major (string)
  - Current GPA (float)

- **We store information in private variables in the class**

STUDENT CLASS

- **What operations could we perform on a student?**
  - Change address
  - Update test scores
  - Change major
  - Update GPA
  - Print all information

- **We use get and set methods and other methods to implement operations**

STUDENT CLASS

```
pubic class Student
{
   // Private variables
   private int ID;
   private String Name;
   private String Address;
   private float GPA;
…
```

These variable declarations define the data fields inside the Student class

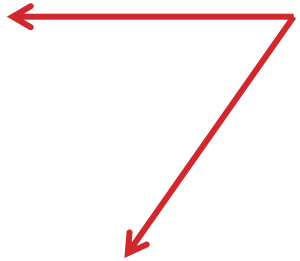
STUDENT CLASS

```
// Default constructor
public Student()
{
   ID = 0;
   Name = "name";
   Address = "address";
   GPA = 0.0;
}
```

We choose default values for all variables

STUDENT CLASS

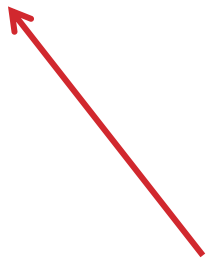
```
// Getters
public int getID() { return ID; }
public String getName() { return Name; }
public String getAddress() { return Address; }
public float getGPA() { return GPA; }

// Setters
public void setID(int id) { ID = id; }
public void setName(String name) { Name = name; }
public void setAddress(String address) { Address = address; }
public void setGPA(float gpa) { GPA = gpa; }
```



One line getters and setters save space in the program

STUDENT CLASS

```
// Print method
public void print()
{
    System.out.println("ID: " + ID);
    System.out.println("GPA: " + GPA);
    System.out.println("Name: " + Name);
    System.out.println("Address: " + Address);
}
```



The format of the output may depend on the needs of the application

STUDENT CLASS

```
// Main program
public static void main(String[] args)
{
   System.out.println("Testing the Student class");
   Student test = new Student();
   test.setID(123456);
   test.setName("John Gauch");
   test.setAddress("518 JB Hunt");
   test.setGPA(3.14);
   test.print();
}
```

We can test the Student class by calling each of the methods

STUDENT CLASS

Testing the Student class

ID: 123456

GPA: 3.14

Name: John Smith

Address: 518 JB Hunt

CODE DEMO

Student.java

LINEAR CLASS

- **How can we represent a linear equation?**
  - Slope intercept formula: $y = mx + b$
    - Store m, b values
  - Geometric formula: $(n_x,n_y) \cdot (x,y) = d$
    - Store normal (n_x,n_y) and distance from origin d
  - Parametric formula: $(x_1,y_1) + t\,(x_2-x_1,y_2-y_1)$
    - Store points on line (x_1,y_1) and (x_2,y_2)
  - Classic formula: $ax + by + c = 0$
    - Store a, b, c values

- **We can store the linear equation in one way, and convert to any of the other representations as needed**

LINEAR CLASS

- **What operations could we perform on a linear equation?**
  - Get and set the line equation coefficients
  - Print the line in y=mx+b or ax+by+c=0 format
  - Check if line is vertical or horizontal
  - Check if two lines are parallel or perpendicular
  - Solve for x when given y
  - Solve for y when given x
  - Calculate the intersection point of two lines
  - Calculate distance from a point to the line

- **Users of this class do not need to know how these operations are implemented – just how to call them**

LINEAR CLASS

```
pubic class Linear
{
    // Private variables
    private double A;
    private double B;
    private double C;
    …
```

We are using the
Ax + By + C = 0
line representation

LINEAR CLASS

```
public Linear() {
        A = 0;
        B = 0;
        C = 0;
}
public Linear(double a, double b, double c) {
        A = a;
        B = b;
        C = c;
}
```

The default constructor sets line equation to 0x + 0y + 0 = 0

The non-default constructor sets line equation to ax + by + c = 0

LINEAR CLASS

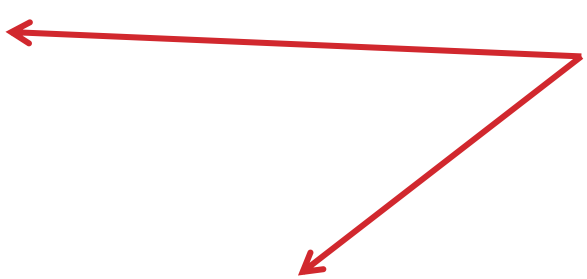
```
// Setter methods
public void setA(double a) { A = a; }
public void setB(double b) { B = b; }
public void setC(double c) { C = c; }

// Getter methods
public double getA() { return A; }
public double getB() { return B; }
public double getC() { return C; }
```

LINEAR CLASS

```
// Check if line is vertical
public Boolean isVertical()
{
   return (B == 0);
}
// Check if line is horizontal
public Boolean isHorizontal()
{
   return (A == 0);
}
```

LINEAR CLASS

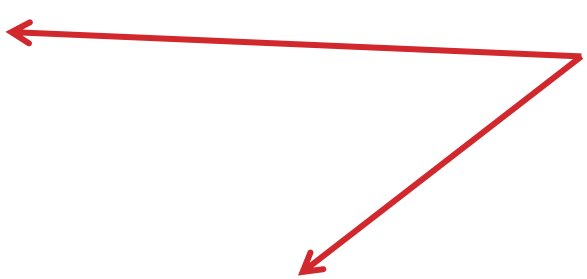
```
// Solve Ax + By + C = 0 for x given y
public double solveForX(double y)
{
   if (A == 0)
      return 0;
   else
      return -(B * y + C) / A;
}
```



We must check value of A to avoid a divide by zero

LINEAR CLASS

```
// Solve Ax + By + C = 0 for y given x
public double solveForY(double x)
{
   if (B == 0)
      return 0;
   else
      return -(A * x + C) / B;
}
```



We must check value of B
to avoid a divide by zero

LINEAR CLASS

```
// Print methods
public String toString()
{
   return String.format("%3.2fx + %3.2fy + %3.2f = 0", A, B, C);
}


public void print()
{
   System.out.println(toString());
}
```

Here we create formatted string for the line equation

Here we print the formatted string out to the screen

LINEAR CLASS

```
// Main program
public static void main(String[] args)
{
   System.out.println("Testing the Linear class");
   Linear eq1 = new Linear(1,2,3);
   System.out.println("\nLine equation: " + eq1.toString());
   System.out.println("\nLine vertical = " + eq1.isVertical());
   System.out.println("\nLine horizontal = " + eq1.isHorizontal());
…
```

LINEAR CLASS

```
   Linear eq2 = new Linear(0,1,2);
   System.out.println("\nLine equation: " + eq2.toString());
   System.out.println("\nLine  vertical = " + eq2.isVertical());
   System.out.println("\nLine  horizontal = " + eq2.isHorizontal());

   Linear eq3 = new Linear(3,0,1);
   System.out.print("\nLine equation: "); eq3.println();
   System.out.println("\nLine  vertical = " + eq3.isVertical());
   System.out.println("\nLine  horizontal = " + eq3.isHorizontal());
}
```

LINEAR CLASS

Testing the Linear class

Line equation: 1.00x + 2.00y + 3.00 = 0

Line vertical = false

Line horizontal = false

LINEAR CLASS

...

Line equation: 0.00x + 1.00y + 2.00 = 0

Line vertical = false

Line horizontal = true

Line equation: 3.00x + 0.00y + 1.00 = 0

Line vertical = true

Line horizontal = false

CODE DEMO

Linear.java

SUMMARY

- **In this section, we showed how two simple classes could be defined, implemented, and used in a program**
  - The Student class illustrated how separate get/set methods could be used for each private variable
  - The Student class methods do not have any error checking, but this could be added (eg. GPA < 4.0)
  - The Linear class uses get/set methods with multiple parameters to access/store private variables
  - The Linear class illustrated how the toString method can simplify print methods

CLASSES

PART 4

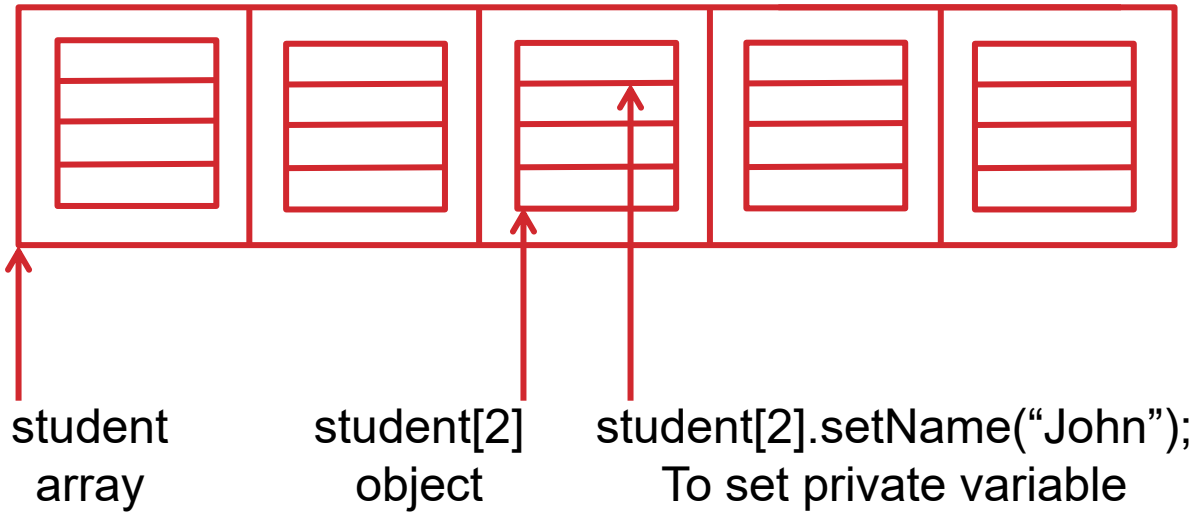
ADVANCED CLASSES

ADVANCED CLASSES

- **Now that we have created a class, we can use it as a building block to create more complex classes**
  - We can create arrays of objects
  - We can nest objects within other objects
  - We can pass objects as parameters into methods
  - We can return objects from methods
  - We can define private “helper” methods in a class
  - We can define public constants or variables in a class
  - We can copy objects with a “copy constructor method”
  - We can compare objects with a “compare method”

ARRAYS OF OBJECTS

- **An array of objects can be used to store data**
  - ”Student[] student = new Student[5]” creates an array of five objects to store all student information

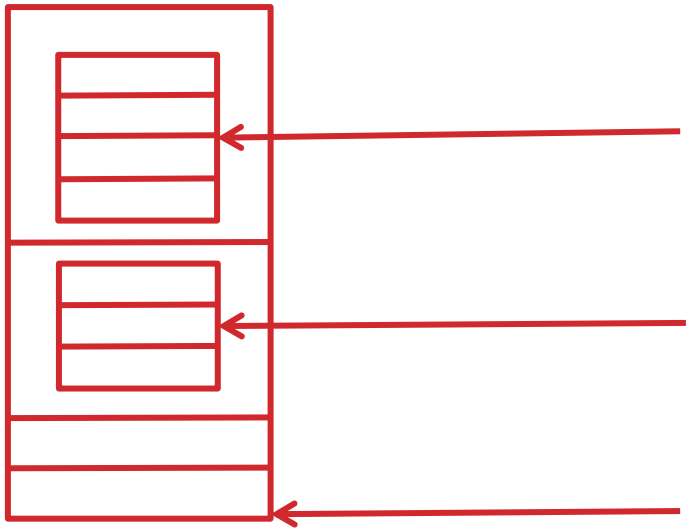


ARRAYS OF OBJECTS

```
// Main program
public static void main(String[] args)
{
   System.out.println("Testing the Student class");
   Student student[] = new Student[5];
   student[2].setID(123456);
   student[2].setName("John");
   student[2].setAddress(”123 Main Street");
   student[2].setGPA(3.45);
   student[2].print();
}
```

OBJECTS WITHIN OBJECTS

- **A class can contain other objects as private variables**
  - By nesting classes we can build more complex ADTs
  - For example, we can store track and field race results using a class that contains two other classes



Person class keeps track of (name, address, age, gender)

Time class keeps track of (hour, minute seconds)

Result class keeps track of all track and field race results

OBJECTS WITHIN OBJECTS

```
pubic class Person
{
   private String name;
   private String address;
   private int age;
   private char gender;
…
}
```

OBJECTS WITHIN OBJECTS

```
pubic class Time
{
   private int hour;
   private int minute;
   private int second;
…
}
```

OBJECTS WITHIN OBJECTS

```
pubic class Result
{
   private Person person;
   private String event_name;
   private Time event_time;
   private boolean disqualified;
   private int event_position;
…
}
```

OBJECTS AS PARAMETERS

- **Objects can be passed as parameters into methods**
  - A reference to the object is sent to the method
  - The method can access and change attributes of the object by calling methods of the object's class

- **Example:**
  - The Event class described above has Person and Time objects as private variables.
  - To store or manipulate these objects in the Event class they need to passed into methods as parameters

OBJECTS AS PARAMETERS

```
pubic class Result
{
   public void setPerson(Person p)
   {
      person = new Person(p);
   }
   public void setEventTime(Time t)
   {
      event_time = new Time(t);
   }
```

We should not use “person = p” because that would not make a copy of the Person object

Similarly, we should use the Time copy constructor to save the time value

RETURNING OBJECTS

- **An object can also be used as a return type for a method**
  - This lets us return more information from a method
  - We need to create and initialize the return object
  - We can use the returned object in our program

- **Example:**
  - The Event class described above has Person and Time objects as private variables.
  - The getPerson and getEventTime methods in the Event class need to return these objects

RETURNING OBJECTS

```
pubic class Result
{
   public Person getPerson()
   {
      return person;
   }
   public Person getEventTime()
   {
      return event_time;
   }
```

PRIVATE METHODS

- **We are allowed to make methods in a class private**
  - Use the keyword “private” when declaring the method
  - Private methods can be called by other methods in the class, but can not be called from outside the class
  - This is useful for error checking operations we need to implement the class, but the user does not need

- **Example:**
  - The “private void correctTime()” method in the Time class ensures that the hour, minute, second are valid

PUBLIC VARIABLES

- **We are allowed to make variables in a class public**
  - Use the keyword “public” when declaring the variable
  - Public variables can be read and modified by users of the class in the main program
  - Some programmers will do this on purpose to avoid the overhead/inconvenience of get/set methods
  - Public variables will break the data hiding principal in object oriented programming, so it is not recommended

PUBLIC CONSTANTS

- **We can declare public constants in a class**
  - We use “public static final” to declare a constant
  - By convention, the name of the constant should be in be in capital letters (e.g. public static final int MAGIC = 42;)

- **Static constants can be used to:**
  - Specify the size of a private array
  - Specify the min/max values on private variables
  - Specify Boolean flags for debugging or printing
  - Specify mathematical constants (e.g. PI)

COPY CONSTRUCTOR

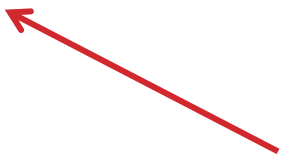
- **Assignment of objects**
  - Assume thing1 and thing2 are objects of the same class
  - We are allowed to type “thing2 = thing1;”
  - This does NOT do a field-by-field copy of thing1 to thing2
  - Instead, thing2 now refers to thing1
  - Any change we make to thing2 will really change thing1
  - Any change we make to thing1 will be visible to thing2

- **We need to implement a copy constructor to make a field-by-field copy of one object into another**
  - Eg. thing2 = new Thing(thing1);

COPY CONSTRUCTOR

- **The copy constructor must copy all data fields from the input object into the data fields of object being created**

```
public Thing(Thing thing)
{
   field1 = thing.field1;
   field2 = thing.field2;
   field3 = thing.field3;
   …
}
```



We can access the data fields of thing because we are inside the Thing class

COMPARE METHOD

- **Comparison of two objects**
  - Assume thing1 and thing2 are objects of the same class
  - We are allowed to type “if (thing2 == thing1)”
  - Unfortunately this does NOT do a field-by-field comparison
  - It tests to see if thing1 and thing2 refer to the same object

- **In order to compare two objects on a field-by-field basis we need to implement a compare method**
  - Eg: if (thing1.compare(thing2) == 0) // returns 0 if equal

COMPARE METHOD

- **The implementation of a compare method should compare all of the the fields of the input object and return 1, 0, -1**

```
public int compare(Thing thing)
{
   if (field1 - thing.field1 > 0) return 1;
   if (field1 - thing.field1 < 0) return -1;
   if (field2 - thing.field2 > 0) return 1;
   if (field2 - thing.field2 < 0) return -1
…
   return 0;
}
```

We return 1 if object is larger and -1 if the parameter is larger

We return 0 if all object fields are equal to each other

COMPARE METHOD

- **The implementation of a compare method should compare all of the the fields of the input object and return 1, 0, -1**

```
public int compare(Thing thing)
{
   if (field1 > thing.field1) return 1;
   if (field1 < thing.field1) return -1;
   if (field2 > thing.field2) return 1;
   if (field2 < thing.field2) return -1
…
   return 0;
}
```

We return 1 if object is larger and -1 if the parameter is larger

We return 0 if all object fields are equal to each other

SUMMARY

- **In this section, we discussed the following:**
  - Composite classes (arrays of objects, nested objects)
  - Assignment and comparison of objects
  - Using objects as parameters and return values
  - Private methods and public variables
  - Static constants
  - Copy constructors
  - Compare methods

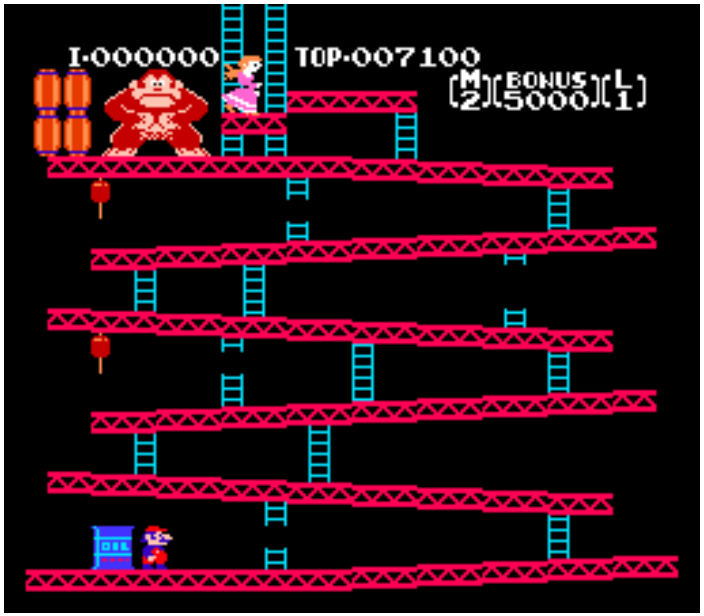
CLASSES

PART 5

ADVANCED CLASS EXAMPLES

ADVANCED CLASS EXAMPLES

- **Consider the problem of creating a 2D platform video game like Super Mario Bros or Donkey Kong**



Sample game images from Wikipedia

ADVANCED CLASS EXAMPLES

- **What do we need to know to implement this game?**
  - We need to know the location of players on the screen
  - We need geometric models for platforms and objects
  - We need images of players, clouds, trees, etc.

- **We can use a collection of classes to store geometric information and implement operations on this data**
  - We can build models using Points, Lines and Polygons
  - These classes will demonstrate many of the advanced Java features discussed in the previous section

POINT CLASS

- **What data do we need to store?**
  - For a 2D point we need the (x,y) coordinates

- **What operations do we need to implement?**
  - Basic get and set methods
  - Some way to print or display points
  - Distance between two points
  - Geometric transformations (translate, rotate, scale)

POINT CLASS

```
public class Point
{
    // Private variables
    private double X;
    private double Y;

    // Constructors
    public Point()
    {
        X = 0;
        Y = 0;
    }
…
```

POINT CLASS

…

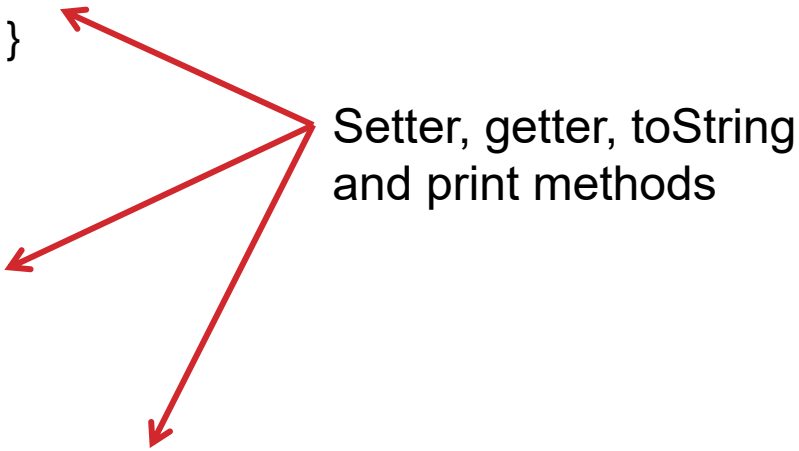
```
public Point(double x, double y)
{
    X = x;
    Y = y;
}

public Point(Point p)
{
    X = p.X;
    Y = p.Y;
}
```

← Constructor with parameters

← Copy constructor

POINT CLASS

```
// Setter methods
public void setX(double x) { X = x; }
public void setY(double y) { Y = y; }

// Getter methods
public double getX() { return X; }
public double getY() { return Y; }

// Print methods
public String toString() { return String.format("(%3.2f, %3.2f)", X, Y); }
public void print() { System.out.print(toString()); }
public void println() { System.out.println(toString()); }
```



POINT CLASS

```
public double distance(Point point)
{
        double dx = X - point.X;
        double dy = Y - point.Y;
        return Math.sqrt(dx*dx + dy*dy);
}
```

Calculate distance between two Points

```
public void translate(double dx, double dy)
{
        X += dx;
        Y += dy;
}
```

Translate the (x,y) coordinates of Point

POINT CLASS

```
public void rotate(double angle)
{
        double newX = X * Math.cos(angle) - Y * Math.sin(angle);
        double newY = X * Math.sin(angle) + Y * Math.cos(angle);
        X = newX;
        Y = newY;
}
```

Rotate the (x,y) coordinates of Point

```
public void scale(double sx, double sy)
{
        X *= sx;
        Y *= sy;
}
```

Scale the (x,y) coordinates of Point

POINT CLASS

```
public static void main(String[] args)
{
        System.out.println("\nTesting the Point class");

        // Test constructors and print methods
        Point p1 = new Point();
        System.out.println("p1 = " + p1.toString());
        Point p2 = new Point( 3,7 );
        System.out.println("p2 = " + p2.toString());
        Point p3 = new Point( p2 );
        System.out.print("p3 = "); p3.println();
…
```

POINT CLASS

```
…
// Test distance calculations
double distance = p1.distance(p2);
System.out.println("p1.distance(p2) = " + distance);
System.out.println("p1.distance(p3) = " + p1.distance(p3));
System.out.println("p2.distance(p3) = " + p2.distance(p3));
…
```

POINT CLASS

```
…
// Test geometric operations
p1.translate(1, -2);
System.out.println("p1.translate(1, -2) = " + p1.toString());
p2.rotate(Math.PI/2);
System.out.println("p2.rotate(PI/2) = " + p2.toString());
p3.scale(1.5, 0.5);
System.out.print("p3.scale(1.5, 0.5) = "); p3.println();
}
```

POINT CLASS

Sample Program Output

Testing the Point class

p1 = (0.00, 0.00)

p2 = (3.00, 7.00)

p3 = (3.00, 7.00)

p1.distance(p2) = 7.615773105863909

p1.distance(p3) = 7.615773105863909

p2.distance(p3) = 0.0

p1.translate(1, -2) = (1.00, -2.00)

p2.rotate(PI/2) = (-7.00, 3.00)

p3.scale(1.5, 0.5) = (4.50, 3.50)

CODE DEMO

Point.java

LINE CLASS

- **What data do we need to store?**
  - Lines can be defined in terms of two Points on the line
  - From this, we can derive Ax+By+C=0 line equation

- **What operations do we need to implement?**
  - Basic get and set methods
  - Some way to print or display lines
  - Geometric transformations (translate, rotate, scale)
  - Distance between points and a line
  - Intersection of two lines

LINE CLASS

```
public class Line
{
   // Private variables
   private Point point1;
   private Point point2;

   // Constructors
   public Line()
   {
      point1 = new Point(0,0);
      point2 = new Point(0,0);
   }
…
```

Private variables of Line are two Point objects

Constructor creates two Points that define Line

LINE CLASS

```
…
   public Line(Point p1, Point p2)
   {
      point1 = new Point(p1);
      point2 = new Point(p2);
   }

   public Line(Line line)
   {
      point1 = new Point(line.point1);
      point2 = new Point(line.point2);
   }
```

Constructor with two Points that define Line

Copy constructor with Line parameter

LINE CLASS

```
…
   public Line(double x1, double y1, double x2, double y2)
   {
      point1 = new Point(x1, y1);
      point2 = new Point(x2, y2);
   }
```

Constructor with four Point coordinates that define Line

LINE CLASS

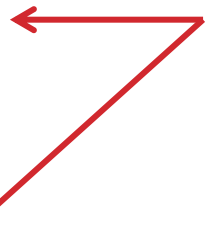
```
…
   // Setter methods
   public void setP1(Point p) { point1 = new Point(p); }
   public void setP2(Point p) { point2 = new Point(p); }

   // Getter methods
   public Point getP1() { return new Point(point1); }
   public Point getP2() { return new Point(point2); }
```

We must make copies of the Point parameters

We must return new Points instead of returning references to private variables

LINE CLASS

```
…
   // Print methods
   public String toString()
   {
      return String.format("%s -> %s",
         point1.toString(), point2.toString());
   }

   public void print() { System.out.print(toString()); }
   public void println() { System.out.println(toString()); }
```



Defining toString and print methods

LINE CLASS

```
…
   // Geometric methods
   public void rotate(double angle)
   {   point1.rotate(angle);
       point2.rotate(angle); }

   public void translate(double dx, double dy)
   {   point1.translate(dx,dy);
       point2.translate(dx,dy); }

   public void scale(double sx, double sy)
   {   point1.scale(sx,sy);
       point2.scale(sx,sy); }
```

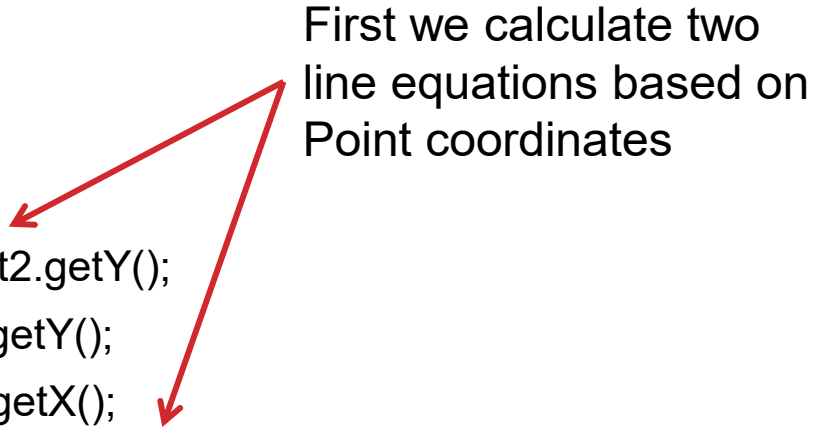
Here we call Point methods to implement geometric operations

LINE CLASS

```
…
    // Distance method
    public double distance(Point point)
    {
        double x = point.getX();
        double y = point.getY();
        double A = point2.getY() - point1.getY();
        double B = point1.getX() - point2.getX();
        double C = - A * point2.getX() - B * point2.getY();
        return (A*x + B*y + C) / Math.sqrt(A*A + B*B);
    }
```

LINE CLASS

```
…
   // Intersection method
   public Point intersect(Line line)
   {
      double A1 = point2.getY() - point1.getY();
      double B1 = point1.getX() - point2.getX();
      double C1 = - A1 * point2.getX() - B1 * point2.getY();
      double A2 = line.point2.getY() - line.point1.getY();
      double B2 = line.point1.getX() - line.point2.getX();
      double C2 = - A2 * line.point2.getX() - B2 * line.point2.getY();
      double x = (B1 * C2 - C1 * B2) / (A1 * B2 - B1 * A2);
      double y = (A1 * C2 - C1 * A2) / (B1 * A2 - A1 * B2);
      return new Point(x, y);
   }
```



LINE CLASS

```
…
   // Intersection method
   public Point intersect(Line line)
   {
      double A1 = point2.getY() - point1.getY();
      double B1 = point1.getX() - point2.getX();
      double C1 = - A1 * point2.getX() - B1 * point2.getY();
      double A2 = line.point2.getY() - line.point1.getY();
      double B2 = line.point1.getX() - line.point2.getX();
      double C2 = - A2 * line.point2.getX() - B2 * line.point2.getY();
      double x = (B1 * C2 - C1 * B2) / (A1 * B2 - B1 * A2);
      double y = (A1 * C2 - C1 * A2) / (B1 * A2 - A1 * B2);
      return new Point(x, y);
   }
```

Next we calculate the line intersection Point (without error checking)

LINE CLASS

```
…
 public static void main(String[] args)
   {
      System.out.println("\nTesting the Line class");

      // Test constructors
      // Test getters and setters
      // Test geometric methods
      // Test intersection method

}
```

Finally we perform unit testing on Line class by calling all of the methods

CODE DEMO

Line.java

POLYGON CLASS

- **What data do we need to store?**
  - A polygon object is a closed sequence of line segments
  - We can define a polygon using an array of Points

- **What operations do we need to implement?**
  - Basic get and set methods
  - Some way to print or display points
  - Geometric transformations (translate, rotate, scale)
  - Eventually want methods to draw polygons in a game

POLYGON CLASS

```
public class Polygon
{
    // Private variables
    private static int MAX_POINTS = 10;
    private int point_count;
    private Point [ ] point_array;

    // Constructors
    public Polygon()
    {
        point_count = 0;
        point_array = new Point[MAX_POINTS];
    }
```

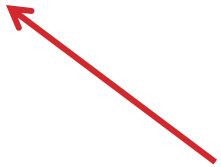
Constant defines maximum number of Points in a Polygon

We allocate empty array of Points and set point count to zero

POLYGON CLASS

…

```
public Polygon(Polygon poly)
{
    point_count = poly.point_count;
    point_array = new Point[MAX_POINTS];
    for (int index = 0; index < point_count; index++)
        point_array[index] = new Point(poly.point_array[index]);
}
```



The copy constructor creates a copy of the array of Points from the Polygon parameter

POLYGON CLASS

```
…
    // Setter method
    public void addPoint(Point point)
    {
        if (point_count < MAX_POINTS)
        {
            // point_array[point_count] = point;
            point_array[point_count] = new Point(point);
            point_count++;
        }
    }
```

We should not save the Point this way because changes to the Polygon would also change the Point object in the main program

Here we save a copy of the Point in the next available array location and increment the counter

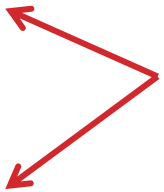
POLYGON CLASS

```
…
   // Getter method
   public Point getPoint(int index)
   {
      if ((index > 0) && (index < point_count))
         return new Point(point_array[index]);
      else
         return null;
   }
```

This method returns one Point from the Polygon array or the value null if index is out of bounds

POLYGON CLASS

```
…
   // Print methods
   public String toString()
   {
      String result = "";
      for (int index = 0; index < point_count; index++)
         result = result + point_array[index].toString() + " ";
      return result;
   }

   public void print() { System.out.print(toString()); }
   public void println() { System.out.println(toString()); }
```



Defining toString and print methods

POLYGON CLASS

```
…
    // Geometric methods
   public void translate(double dx, double dy)
   {   for (int index = 0; index < point_count; index++)
           point_array[index].translate(dx,dy); }

   public void rotate(double angle)
   {   for (int index = 0; index < point_count; index++)
           point_array[index].rotate(angle); }

   public void scale(double sx, double sy)
   {   for (int index = 0; index < point_count; index++)
           point_array[index].scale(sx,sy); }
```

Here we call Point methods to implement geometric operations

POLYGON CLASS

```
…
    public static void main(String[] args)
   {
       System.out.println("\nTesting the Polygon class");

       // Test Constructors and setters and getters
       Polygon poly1 = new Polygon();
       poly1.addPoint( new Point(3,7) );
       poly1.addPoint( new Point(6,1) );
       poly1.addPoint( new Point(4,5) );
       System.out.println("poly1 = " + poly1.toString());
       System.out.println("poly1.getPoint(1) = " + poly1.getPoint(1).toString());
```

POLYGON CLASS

…

```
// Test constructors and setters and getters
Polygon poly2 = new Polygon(poly1);
poly2.addPoint( new Point(2,8) );
poly2.addPoint( new Point(9,0) );
System.out.println("poly2 = " + poly2.toString());
System.out.println("poly2.getPoint(3) = " + poly2.getPoint(3).toString());
```

POLYGON CLASS

…

```
        // Test geometric methods
        poly1.translate(1,1);
        System.out.println("poly1.translate(1,1) = " + poly1.toString());
        poly1.scale(0.5,2);
        System.out.println("poly1.scale(0.5,2) = " + poly1.toString());
        poly2.rotate(Math.PI/2);
        System.out.println("poly2.rotate(PI/2) = " + poly2.toString());
    }
```

POLYGON CLASS

Sample Program Output

Testing the Polygon class

poly1 = (3.00, 7.00) (6.00, 1.00) (4.00, 5.00)

poly1.getPoint(1) = (6.00, 1.00)

poly2 = (3.00, 7.00) (6.00, 1.00) (4.00, 5.00) (2.00, 8.00) (9.00, 0.00)

poly2.getPoint(3) = (2.00, 8.00)

poly1.translate(1,1) = (4.00, 8.00) (7.00, 2.00) (5.00, 6.00)

poly1.scale(0.5,2) = (2.00, 16.00) (3.50, 4.00) (2.50, 12.00)

poly2.rotate(PI/2) = (-7.00, 3.00) (-1.00, 6.00) (-5.00, 4.00)

(-8.00, 2.00) (0.00, 9.00)

CODE DEMO

Polygon.java

SUMMARY

- **In this section, we described three advanced classes**
  - The Point class stores (x,y) coordinates
  - The Line class is defined using two Point objects
  - The Polygon class is defined using an array of Points
  - The geometric operations in the Line class and the Polygon class call methods in the Point class

- **We also illustrated how to do unit testing**
  - Write main program that calls all methods in a class
  - Print results and verify correctness by hand