Introduction to Object Classes with Class Coordinate

Mitsu Ogihara

Department of Computer Science
University of Miami

Table of Contents

Defining a New Class for Storing Information

So far we have written only codes that use only static methods

We have seen three types of classes:

- Classes that provide `main` and thus are executable
- Classes without `main` that provide static methods that can be called from other classes, e.g., `Math`
- **Classes for storing, accessing, and modifying information**
  $\cdots$ `Random`, `File`, `Scanner`, etc.

We will learn to write a class of the last type

Reasons to Define New Container Classes

- Want to group a number of data items (homogeneous or hetrogeneous) that collectively represent something
- Want to add new methods to an existing class

Coordinates of a Point of a Plane

- The point on the xy-plane consists of tis x- and y-coordinates
- Write a class for storing a point on the xy-plane
- Combine two `double` values `x` and `y`

Table of Contents

Definiting a Class Coordinate

The following `Coordinate` captures the idea of bringing together two values

```
1  public class Coordinate {
2    double x, y;
3
4    Coordinate( double xValue, double yValue ) {
5      x = xValue;
6      y = yValue;
7    }
```

The class header ... no difference

Definiting a Class Coordinate

The following `Coordinate` captures the idea of bringing together two values

```
1  public class Coordinate {
2    double x, y;
3
4    Coordinate( double xValue, double yValue ) {
5      x = xValue;
6      y = yValue;
7    }
```

`x` and `y` are the double variables representing the coordinates
They are declared outside methods and without the `static` attribute
They are called `instance variables` or `field variables`

Definiting a Class Coordinate

The following `Coordinate` captures the idea of bringing together two values

```
1  public class Coordinate {
2    double x, y;
3
4    Coordinate( double xValue, double yValue ) {
5      x = xValue;
6      y = yValue;
7    }
```

A container class has methods with the same name as the class
They are called **constructors**
Constructors are non-static and may take some parameters

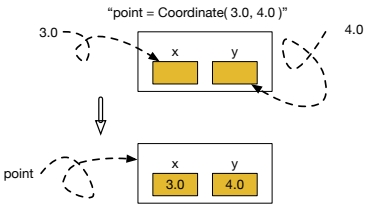
Definiting a Class Coordinate

The following `Coordinate` captures the idea of bringing together two values

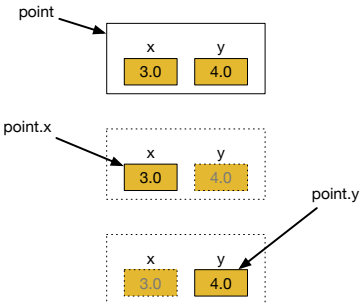
```
1  public class Coordinate {
2    double x, y;
3
4    Coordinate( double xValue, double yValue ) {
5      x = xValue;
6      y = yValue;
7    }
```

The constructor here assigns the value to the instance variables specified in the two parameters
Here this act as the method for initializing the instance variables

Constructor



Accessing the Field Variables



The First Code Using `Coordinate`

- Create one `Coordinate` object, `point`, by receiving the coordinates from the user
- Print the coordinates of `point` by directly accessing the instance variables `x` and `y`, by `point.x` and `point.y`
- Compute the distaance between the origin and `point` and print the distance

PlayWithCoordinate1

```
1  import java.util.Scanner;
2  public class PlayWithCoordinate1 {
3    public static void main( String[] args ) {
4      Scanner console = new Scanner( System.in );
5      System.out.print( "Enter x and y of the point: " );
6      double xVal = console.nextDouble();
7      double yVal = console.nextDouble();
8      Coordinate point = new Coordinate( xVal, yVal );
9      System.out.printf( "The point is (%f,%f)%n",
10         point.x, point.y );
11     double distance = Math.sqrt(
12         point.x * point.x + point.y * point.y );
13     System.out.printf(
14         "The distance from the origin is %f%n", distance );
15   }
16 }
```

Prompt the user to enter values for variables `xVal` and `yVal` and receive the values

PlayWithCoordinate1

```
1  import java.util.Scanner;
2  public class PlayWithCoordinate1 {
3    public static void main( String[] args ) {
4      Scanner console = new Scanner( System.in );
5      System.out.print( "Enter x and y of the point: " );
6      double xVal = console.nextDouble();
7      double yVal = console.nextDouble();
8      Coordinate point = new Coordinate( xVal, yVal );
9      System.out.printf( "The point is (%f,%f)%n",
10         point.x, point.y );
11     double distance = Math.sqrt(
12         point.x * point.x + point.y * point.y );
13     System.out.printf(
14         "The distance from the origin is %f%n", distance );
15   }
16 }
```

Create a `Coordinate` object `point` out of the two values

PlayWithCoordinate1

```
1  import java.util.Scanner;
2  public class PlayWithCoordinate1 {
3    public static void main( String[] args ) {
4      Scanner console = new Scanner( System.in );
5      System.out.print( "Enter x and y of the point: " );
6      double xVal = console.nextDouble();
7      double yVal = console.nextDouble();
8      Coordinate point = new Coordinate( xVal, yVal );
9      System.out.printf( "The point is (%f,%f)%n",
10         point.x, point.y );
11     double distance = Math.sqrt(
12         point.x * point.x + point.y * point.y );
13     System.out.printf(
14         "The distance from the origin is %f%n", distance );
15   }
16 }
```

Print the coordinates by directly accessing the field variables

PlayWithCoordinate1

```
1  import java.util.Scanner;
2  public class PlayWithCoordinate1 {
3    public static void main( String[] args ) {
4      Scanner console = new Scanner( System.in );
5      System.out.print( "Enter x and y of the point: " );
6      double xVal = console.nextDouble();
7      double yVal = console.nextDouble();
8      Coordinate point = new Coordinate( xVal, yVal );
9      System.out.printf( "The point is (%f,%f)%n",
10         point.x, point.y );
11     double distance = Math.sqrt(
12         point.x * point.x + point.y * point.y );
13     System.out.printf(
14         "The distance from the origin is %f%n", distance );
15   }
16 }
```

Compute the distance fromt the root by accessing the field variables

PlayWithCoordinate1

```
1  import java.util.Scanner;
2  public class PlayWithCoordinate1 {
3    public static void main( String[] args ) {
4      Scanner console = new Scanner( System.in );
5      System.out.print( "Enter x and y of the point: " );
6      double xVal = console.nextDouble();
7      double yVal = console.nextDouble();
8      Coordinate point = new Coordinate( xVal, yVal );
9      System.out.printf( "The point is (%f,%f)%n",
10         point.x, point.y );
11     double distance = Math.sqrt(
12         point.x * point.x + point.y * point.y );
13     System.out.printf(
14         "The distance from the origin is %f%n", distance );
15   }
16 }
```

Print the distance

Classes as Information Containers

- A container class needs **constructor**, a creator of the information container
- A container class may have **instance methods** $\cdots$ methods for executing certain tasks using the data stored in the object
  Two special types of instance methods:
  - **accessors** $\cdots$ methods for retrieving the instance variables or some properties (e.g., `getAbsolutePath` and `exists` for class `File`)
  - **mutators** $\cdots$ methods for modifying the instance variables or some properties (e.g., `setTeam`)

Instance Methods That Return a `Coordinate` object

```
9   public Coordinate plus( Coordinate o ) {
10    return new Coordinate( x+o.x, y+o.y );
11  }
12  public Coordinate minus( Coordinate o ) {
13    return new Coordinate( x-o.x, y-o.y );
14  }
15  public Coordinate scale( double scalor ) {
16    return new Coordinate( x*scalor, y*scalor );
17  }
```

The method `plus` takes another `Coordinate` object as a parameter and returns a new `Coordinate` object that is shifted from the current one by the coordinate values of the parameter

Instance Methods That Return a `Coordinate` object

```
9   public Coordinate plus( Coordinate o ) {
10    return new Coordinate( x+o.x, y+o.y );
11  }
12  public Coordinate minus( Coordinate o ) {
13    return new Coordinate( x-o.x, y-o.y );
14  }
15  public Coordinate scale( double scalor ) {
16    return new Coordinate( x*scalor, y*scalor );
17  }
```

The field variables of the `o` can be accessed by `o.x` and `o.y`

Instance Methods That Return a `Coordinate` object

```
9   public Coordinate plus( Coordinate o ) {
10    return new Coordinate( x+o.x, y+o.y );
11  }
12  public Coordinate minus( Coordinate o ) {
13    return new Coordinate( x-o.x, y-o.y );
14  }
15  public Coordinate scale( double scalor ) {
16    return new Coordinate( x*scalor, y*scalor );
17  }
```

The method `minus` takes another `Coordinate` object as a parameter and returns a new `Coordinate` object that is negatively shifted from the current one by the coordinate values of the parameter

Instance Methods That Return a `Coordinate` object

```
9   public Coordinate plus( Coordinate o ) {
10    return new Coordinate( x+o.x, y+o.y );
11  }
12  public Coordinate minus( Coordinate o ) {
13    return new Coordinate( x-o.x, y-o.y );
14  }
15  public Coordinate scale( double scalor ) {
16    return new Coordinate( x*scalor, y*scalor );
17  }
```

The method `scale` takes a double value `scalor` as a parameter and returns a new `Coordinate` object whose coordinate values are the current values multiplied by `scalor`

Instance Methods That Return a double value

```
19    public double distance( Coordinate o ) {
20      return Math.sqrt( Math.pow( x-o.x, 2 ) + Math.pow( y-o.y, 2 ) );
21    }
22    public double distance() {
23      return Math.sqrt( Math.pow( x, 2 ) + Math.pow( y, 2 ) );
24    }
25    public double innerProduct( Coordinate o ) {
26      return x*o.x + y*o.y;
27    }
28  }
```

One version of the method `distance` takes another `Coordinate` object as a parameter and returns the Euclidean distance between the two

Instance Methods That Return a double value

```
19    public double distance( Coordinate o ) {
20      return Math.sqrt( Math.pow( x-o.x, 2 ) + Math.pow( y-o.y, 2 ) );
21    }
22    public double distance() {
23      return Math.sqrt( Math.pow( x, 2 ) + Math.pow( y, 2 ) );
24    }
25    public double innerProduct( Coordinate o ) {
26      return x*o.x + y*o.y;
27    }
28  }
```

Another version of the method `distance` takes no parameter and returns a new `Coordinate` object that is negatively shifted from the current one by the coordinate values of the parameter
This is by way of **method overloading**

Instance Methods That Return a double value

```
19    public double distance( Coordinate o ) {
20      return Math.sqrt( Math.pow( x-o.x, 2 ) + Math.pow( y-o.y, 2 ) );
21    }
22    public double distance() {
23      return Math.sqrt( Math.pow( x, 2 ) + Math.pow( y, 2 ) );
24    }
25    public double innerProduct( Coordinate o ) {
26      return x*o.x + y*o.y;
27    }
28  }
```

The method `innerProduct` takes another `Coordinate` object `o`, and returns the inner product of the vector going to the present coordinate and the one going to `o`

The method `scale` takes a double value `scalor` as a parameter and returns a new `Coordinate` object whose coordinate values are the current values multiplied by `scalor`

Using Instance Methods

```
1  import java.util.Scanner;
2
3  public class PlayWithCoordinate2 {
4    public static void main( String[] args ) {
5      Scanner console = new Scanner( System.in );
6      System.out.print( "Enter x and y for point1: " );
7      double x = console.nextDouble();
8      double y = console.nextDouble();
9      Coordinate point1 = new Coordinate( x, y );
10
11     System.out.print( "Enter x and y for point2: " );
12     x = console.nextDouble();
13     y = console.nextDouble();
14     Coordinate point2 = new Coordinate( x, y );
```

Declaring a console Scanner

Using Instance Methods

```
1  import java.util.Scanner;
2
3  public class PlayWithCoordinate2 {
4    public static void main( String[] args ) {
5      Scanner console = new Scanner( System.in );
6      System.out.print( "Enter x and y for point1: " );
7      double x = console.nextDouble();
8      double y = console.nextDouble();
9      Coordinate point1 = new Coordinate( x, y );
10
11     System.out.print( "Enter x and y for point2: " );
12     x = console.nextDouble();
13     y = console.nextDouble();
14     Coordinate point2 = new Coordinate( x, y );
```

Constructing a `Coordinate` object `point1` by receiving input from the user

Using Instance Methods

```
1  import java.util.Scanner;
2
3  public class PlayWithCoordinate2 {
4    public static void main( String[] args ) {
5      Scanner console = new Scanner( System.in );
6      System.out.print( "Enter x and y for point1: " );
7      double x = console.nextDouble();
8      double y = console.nextDouble();
9      Coordinate point1 = new Coordinate( x, y );
10
11     System.out.print( "Enter x and y for point2: " );
12     x = console.nextDouble();
13     y = console.nextDouble();
14     Coordinate point2 = new Coordinate( x, y );
```

Constructing a `Coordinate` object `point2` by receiving input from the user

Using Instance Methods (cont'd)

```
16      Coordinate sum = point1.plus( point2 );
17      Coordinate diff = point1.minus( point2 );
18      System.out.print( "Enter a scale factor: " );
19      double scaleFactor = console.nextDouble();
20      Coordinate scaled = point1.scale( scaleFactor );
21
22      System.out.printf( "point1 is (%f,%f)%n", point1.x, point1.y );
23      System.out.printf( "point2 is (%f,%f)%n", point2.x, point2.y );
24      System.out.printf( "point1 + point2 is (%f,%f)%n", sum.x, sum.y );
25      System.out.printf( "point1 - point2 is (%f,%f)%n",
26          diff.x, diff.y );
27      System.out.printf( "%f * point1 is (%f,%f)%n", scaleFactor,
28          scaled.x, scaled.y );
```

Compute the sum and the diff

Using Instance Methods (cont'd)

```
16      Coordinate sum = point1.plus( point2 );
17      Coordinate diff = point1.minus( point2 );
18      System.out.print( "Enter a scale factor: " );
19      double scaleFactor = console.nextDouble();
20      Coordinate scaled = point1.scale( scaleFactor );
21
22      System.out.printf( "point1 is (%f,%f)%n", point1.x, point1.y );
23      System.out.printf( "point2 is (%f,%f)%n", point2.x, point2.y );
24      System.out.printf( "point1 + point2 is (%f,%f)%n", sum.x, sum.y );
25      System.out.printf( "point1 - point2 is (%f,%f)%n",
26          diff.x, diff.y );
27      System.out.printf( "%f * point1 is (%f,%f)%n", scaleFactor,
28          scaled.x, scaled.y );
```

Receive a scale and compute the scaled version of `point1`

Using Instance Methods (cont'd)

```
16      Coordinate sum = point1.plus( point2 );
17      Coordinate diff = point1.minus( point2 );
18      System.out.print( "Enter a scale factor: " );
19      double scaleFactor = console.nextDouble();
20      Coordinate scaled = point1.scale( scaleFactor );
21
22      System.out.printf( "point1 is (%f,%f)%n", point1.x, point1.y );
23      System.out.printf( "point2 is (%f,%f)%n", point2.x, point2.y );
24      System.out.printf( "point1 + point2 is (%f,%f)%n", sum.x, sum.y );
25      System.out.printf( "point1 - point2 is (%f,%f)%n",
26          diff.x, diff.y );
27      System.out.printf( "%f * point1 is (%f,%f)%n", scaleFactor,
28          scaled.x, scaled.y );
```

Print the coordinate values of the two input points

Using Instance Methods (cont'd)

```
16      Coordinate sum = point1.plus( point2 );
17      Coordinate diff = point1.minus( point2 );
18      System.out.print( "Enter a scale factor: " );
19      double scaleFactor = console.nextDouble();
20      Coordinate scaled = point1.scale( scaleFactor );
21
22      System.out.printf( "point1 is (%f,%f)%n", point1.x, point1.y );
23      System.out.printf( "point2 is (%f,%f)%n", point2.x, point2.y );
24      System.out.printf( "point1 + point2 is (%f,%f)%n", sum.x, sum.y );
25      System.out.printf( "point1 - point2 is (%f,%f)%n",
26          diff.x, diff.y );
27      System.out.printf( "%f * point1 is (%f,%f)%n", scaleFactor,
28          scaled.x, scaled.y );
```

Print the coordinate values of the sum and the diff

Using Instance Methods (cont'd)

```
16      Coordinate sum = point1.plus( point2 );
17      Coordinate diff = point1.minus( point2 );
18      System.out.print( "Enter a scale factor: " );
19      double scaleFactor = console.nextDouble();
20      Coordinate scaled = point1.scale( scaleFactor );
21
22      System.out.printf( "point1 is (%f,%f)%n", point1.x, point1.y );
23      System.out.printf( "point2 is (%f,%f)%n", point2.x, point2.y );
24      System.out.printf( "point1 + point2 is (%f,%f)%n", sum.x, sum.y );
25      System.out.printf( "point1 - point2 is (%f,%f)%n",
26          diff.x, diff.y );
27      System.out.printf( "%f * point1 is (%f,%f)%n", scaleFactor,
28          scaled.x, scaled.y );
```

Print the coordinate values of the scaled point

Using Instance Methods (cont'd)

```
30      double distance1 = point1.distance( point2 );
31      double distance2 = point1.distance();
32      double inner = point1.innerProduct( point2 );
33      System.out.printf( "| point1 - point2 | is %f%n", distance1 );
34      System.out.printf( "| point1 - O | is %f%n", distance2 );
35      System.out.printf( "( point1, point2 ) is %f%n", inner );
36    }
37  }
```

Compute the distance between the two

Using Instance Methods (cont'd)

```
30      double distance1 = point1.distance( point2 );
31      double distance2 = point1.distance();
32      double inner = point1.innerProduct( point2 );
33      System.out.printf( "| point1 - point2 | is %f%n", distance1 );
34      System.out.printf( "| point1 - O | is %f%n", distance2 );
35      System.out.printf( "( point1, point2 ) is %f%n", inner );
36    }
37  }
```

Compute the distance of the first from the origin

Using Instance Methods (cont'd)

```
30      double distance1 = point1.distance( point2 );
31      double distance2 = point1.distance();
32      double inner = point1.innerProduct( point2 );
33      System.out.printf( "| point1 - point2 | is %f%n", distance1 );
34      System.out.printf( "| point1 - O | is %f%n", distance2 );
35      System.out.printf( "( point1, point2 ) is %f%n", inner );
36    }
37  }
```

Compute the inner product between the two

Using Instance Methods (cont'd)

```
30      double distance1 = point1.distance( point2 );
31      double distance2 = point1.distance();
32      double inner = point1.innerProduct( point2 );
33      System.out.printf( "| point1 - point2 | is %f%n", distance1 );
34      System.out.printf( "| point1 - O | is %f%n", distance2 );
35      System.out.printf( "( point1, point2 ) is %f%n", inner );
36    }
37  }
```

Report the results

Table of Contents

Information Hiding

In the previous example, the user of the Coordinate class had direct access to the fields by attaching a period and a field name:

```
System.out.println( point1.x );
```

Information Hiding

In the previous example, the user of the Coordinate class had direct access to the fields by attaching a period and a field name:

```
System.out.println( point1.x );
```

A possible issue with such a design:

- Whenever there is a change in the `Coordinate` class (changing the name of the field from `x` to `valueX`), the class that uses `Coordinate` code on any class that uses it (we call such a class a **client**) be some changes in the user code

Information Hiding

In the previous example, the user of the Coordinate class had direct access to the fields by attaching a period and a field name:

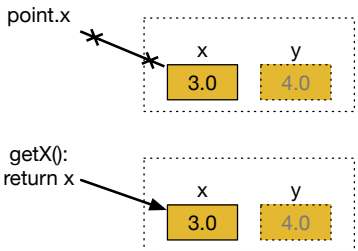
```
System.out.println( point1.x );
```

A possible issue with such a design:

- Whenever there is a change in the `Coordinate` class (changing the name of the field from `x` to `valueX`), the class that uses `Coordinate` code on any class that uses it (we call such a class a **client**) be some changes in the user code

Coding will be a bit simpler if the direct access to fields are prohibited
$\rightarrow$ turning the Coordinate class a black box

Accessing the Field Variables via getX()



Why Use Information Hiding

- Two separate people can work on the code development project: one for the client class and the other for the container class

Why Use Information Hiding

- Two separate people can work on the code development project: one for the client class and the other for the container class
- As long as the parameter signature is maintained for the constructors and the accessors
  - The client class developer can change the code without having to tell the container class developer, vice versa

Why Use Information Hiding

- Two separate people can work on the code development project: one for the client class and the other for the container class
- As long as the parameter signature is maintained for the constructors and the accessors
  - The client class developer can change the code without having to tell the container class developer, vice versa
- This information hiding concept comes very central to large-scale code development, such as API development

Coordinate - an Information-Hidden Implementation of Coordinates

- Attach to the instance variables an attribute of `private`, which means that only the methods in the class have access to them
- Provide methods for return the values of the instance variables (not necessarily in the same data type) (called **Accessors**)
- Provide methods for modifying the values of the instance variables (not necessarily in the same data type) (called **Mutators**)

```
1  public class CoordinateHidden {
2    private double x, y;
3
4    CoordinateHidden( double xValue, double yValue ) {
5      x = xValue;
6      y = yValue;
7    }
8
9    public CoordinateHidden plus( CoordinateHidden o ) {
10     return new CoordinateHidden( x+o.x, y+o.y );
11   }
12   public CoordinateHidden minus( CoordinateHidden o ) {
13     return new CoordinateHidden( x-o.x, y-o.y );
14   }
15   public CoordinateHidden scale( double scalor ) {
16     return new CoordinateHidden( x*scalor, y*scalor );
17   }
```

Now the instance variables have the `private` attribute

Coordinate - an Information-Hidden Implementation of Coordinates

- Attach to the instance variables an attribute of `private`, which means that only the methods in the class have access to them
- Provide methods for return the values of the instance variables (not necessarily in the same data type) (called **Accessors**)
- Provide methods for modifying the values of the instance variables (not necessarily in the same data type) (called **Mutators**)

```
1  public class CoordinateHidden {
2    private double x, y;
3
4    CoordinateHidden( double xValue, double yValue ) {
5      x = xValue;
6      y = yValue;
7    }
8
9    public CoordinateHidden plus( CoordinateHidden o ) {
10     return new CoordinateHidden( x+o.x, y+o.y );
11   }
12   public CoordinateHidden minus( CoordinateHidden o ) {
13     return new CoordinateHidden( x-o.x, y-o.y );
14   }
15   public CoordinateHidden scale( double scalor ) {
16     return new CoordinateHidden( x*scalor, y*scalor );
17   }
```

The existing methods have no difference other that the class name has been

Information Hiding: this

```
29    public double getX() {
30      return x;
31    }
32    public double getY() {
33      return y;
34    }
35    public void setX( double x ) {
36      this.x = x;
37    }
38    public void setY( double y ) {
39      this.y = y;
40    }
```

The accessor for the x-coordinate

Information Hiding: this

```
29    public double getX() {
30      return x;
31    }
32    public double getY() {
33      return y;
34    }
35    public void setX( double x ) {
36      this.x = x;
37    }
38    public void setY( double y ) {
39      this.y = y;
40    }
```

The accessor for the y-coordinate

Information Hiding: this

```
29    public double getX() {
30      return x;
31    }
32    public double getY() {
33      return y;
34    }
35    public void setX( double x ) {
36      this.x = x;
37    }
38    public void setY( double y ) {
39      this.y = y;
40    }
```

The mutators for the x- and y-coordinates

Information Hiding: this

```
29    public double getX() {
30      return x;
31    }
32    public double getY() {
33      return y;
34    }
35    public void setX( double x ) {
36      this.x = x;
37    }
38    public void setY( double y ) {
39      this.y = y;
40    }
```

Here `this.` refers to the object to which the method is to applied, to distinguish it from the `x` given as the parameter