Topic 23
Classes – Part I

"A 'class' is where we teach an 'object' to behave."

-Rich Pattis

Based on slides for Building Java Programs by Reges/Stepp, found at http://faculty.washington.edu/stepp/book/

An example that benefits from new object types

City distance program

- Given an input file named `cities.txt` that contains x/y coordinates of many cities, like this:

```
6
50 20
90 60
10 72
74 98
5 136
150 91
```

- Write a program that prints distance information:

```
Type a city's x/y coordinates: 10 72
for the city at (10, 72):
    you are 65.6 miles from the city at (50, 20)
    you are 80.89 miles from the city at (90, 60)
    you are 69.08 miles from the city at (74, 98)
    you are 64.2 miles from the city at (5, 136)
    you are 141.28 miles from the city at (150, 91)
    you are 72.69 miles from the origin at (0, 0)
```

The Point type

- Java has a type of objects named Point, in the `java.awt` package.
- Constructing a Point object, general syntax:

```
Point <name> = new Point(<x>, <y>);
```

or

```
Point <name> = new Point();  // the origin, (0, 0)
```

  - Example:

```
Point p1 = new Point(5, -2);
Point p2 = new Point();  // 0, 0
```

- Point objects are useful:
  - An array of Points is useful to store many x/y pairs.
  - In programs that do a lot of 2D graphics, it can be nice to be able to return an (x, y) pair from a method.
  - Points have several useful geometric methods we can call in our programs.

Point object methods

- Data fields of Point objects:

Field name	Description
x	Point's x-coordinate
y	Point's y-coordinate

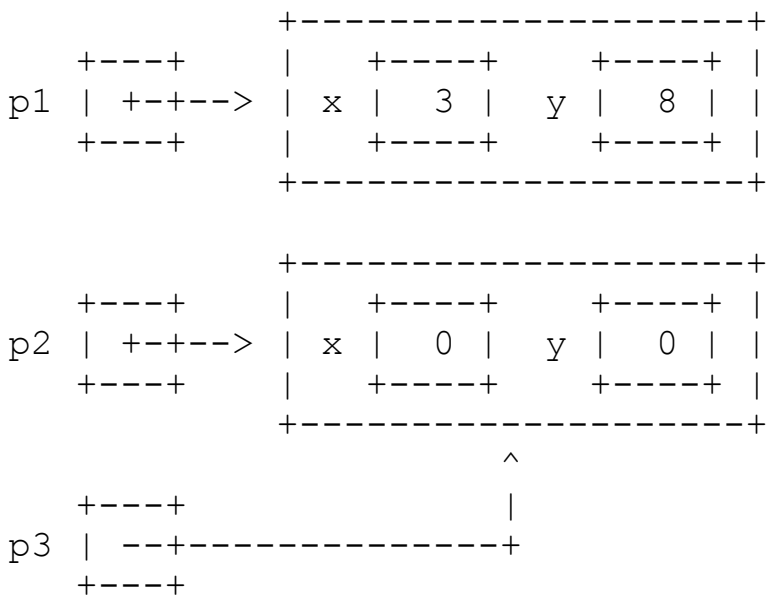
- Useful methods of Point objects:

Method name	Description
distance(*Point*)	how far apart these two Points are
equals(*Point*)	whether the two Points have the same (x, y) coordinates
setLocation(*x*, *y*)	changes this Point's x and y to be the given values
toString()	converts the Point into a String such as "java.awt.Point[x=5,y=-2]"
translate(*dx*, *dy*)	adjusts this Point's x and y by the given difference amounts

Reminder: references

- Remember that variables of Object types store references to the actual object. Here we have 3 variables that refer to 2 unique objects:

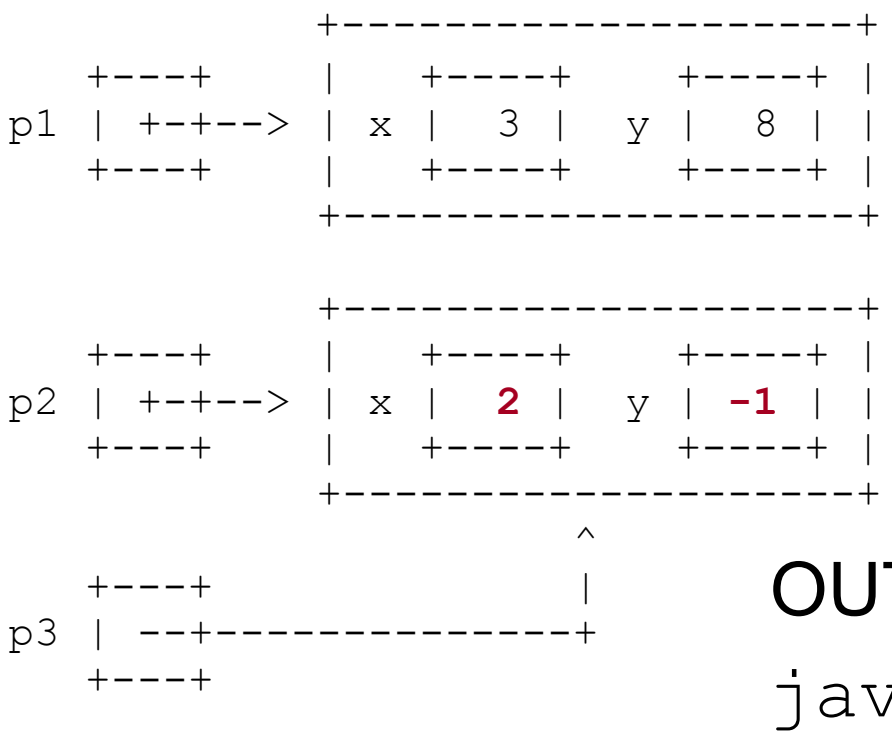
```
Point p1 = new Point(3, 8);
Point p2 = new Point();
Point p3 = p2;
```



Reference semantics

- If two variables refer to the same object, modifying one of them will also make a change in the other:

```
p3.setLocation(2, -1);
System.out.println(p2.toString());
```



OUTPUT:

```
java.awt.Point[x=2,y=-1]
```

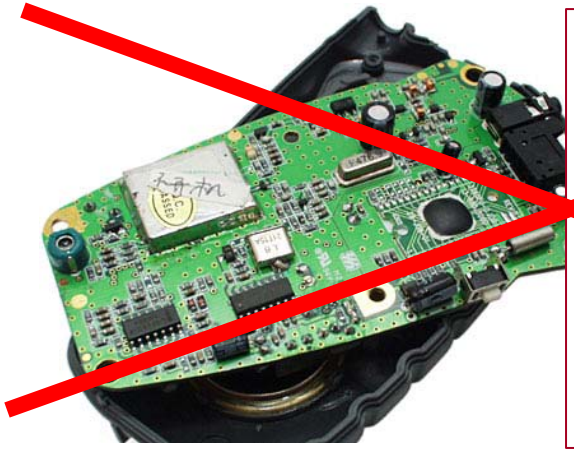
Objects and Object Oriented Programming

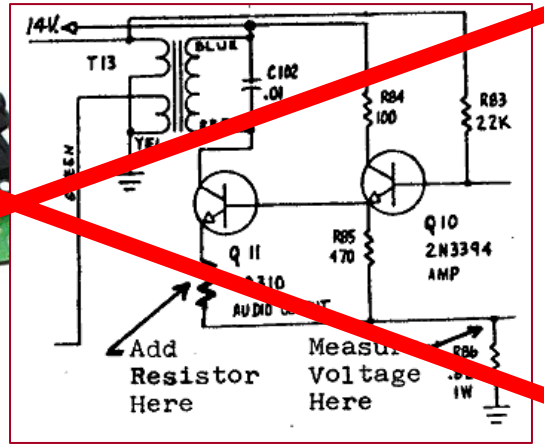
- **object**: An encapsulation of data and behavior.
- **object-oriented programming (OOP)**: Writing programs that perform most of their useful behavior through interactions with objects.

- So far, we have interacted with objects of the following data types:
  - String
  - Point
  - DrawingPanel
  - Graphics
  - Color
  - Scanner
  - Random
  - File

Abstractions

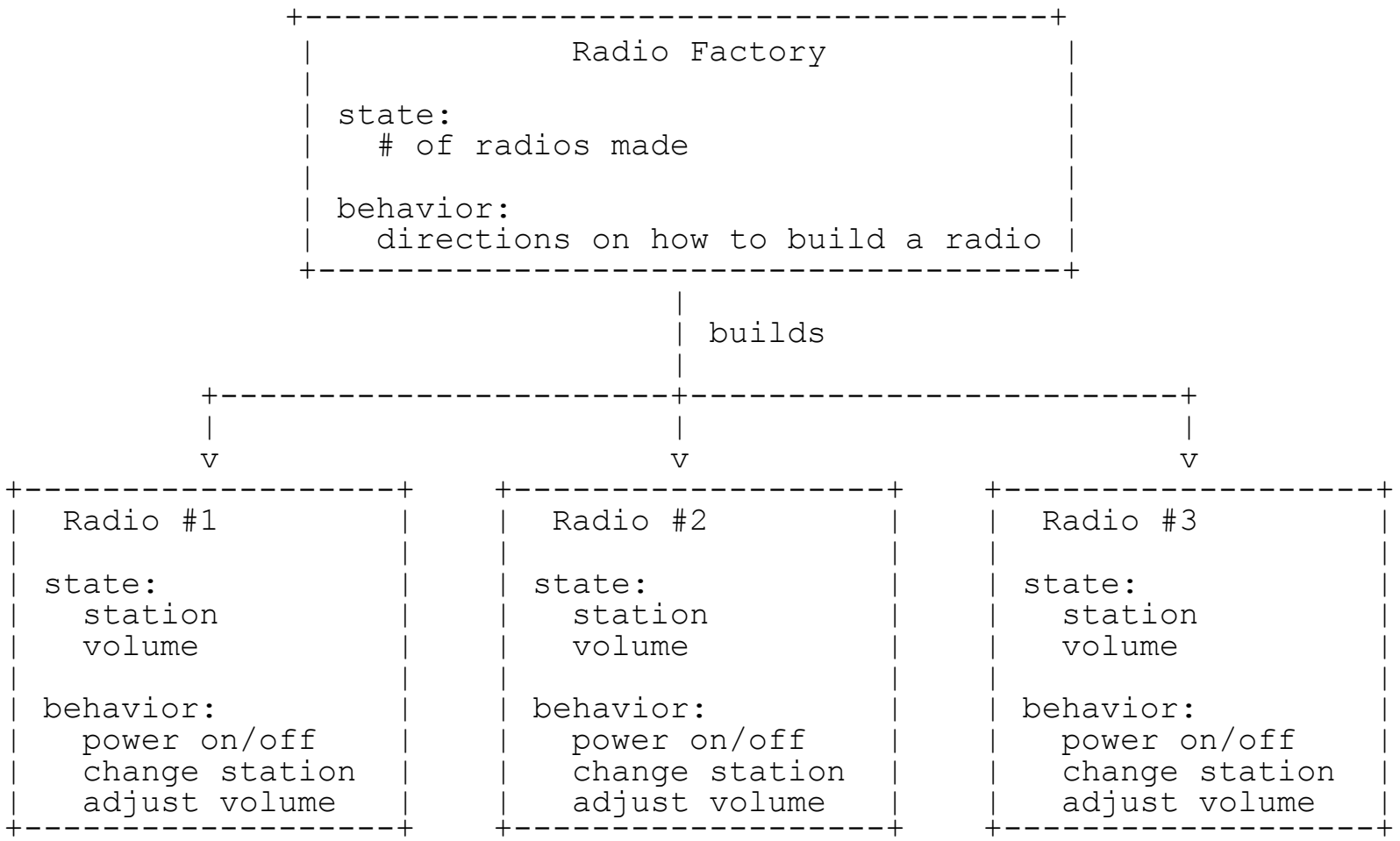
- **abstraction**: A distancing between ideas and details.
  - The objects in Java provide a level of abstraction, because we can use them without knowing how they work.
- You use abstraction every day when interacting with technological 'objects' such as a radio.
  - You understand the external behavior of the radio (volume knobs/buttons, station dial, etc.)
  - You might not understand the inner workings of the radio (capacitors, wires, etc.)
    - You don't *need* to understand the inner workings to **use** the radio
    - You do need to understand the inner workings if you want to **build** a radio





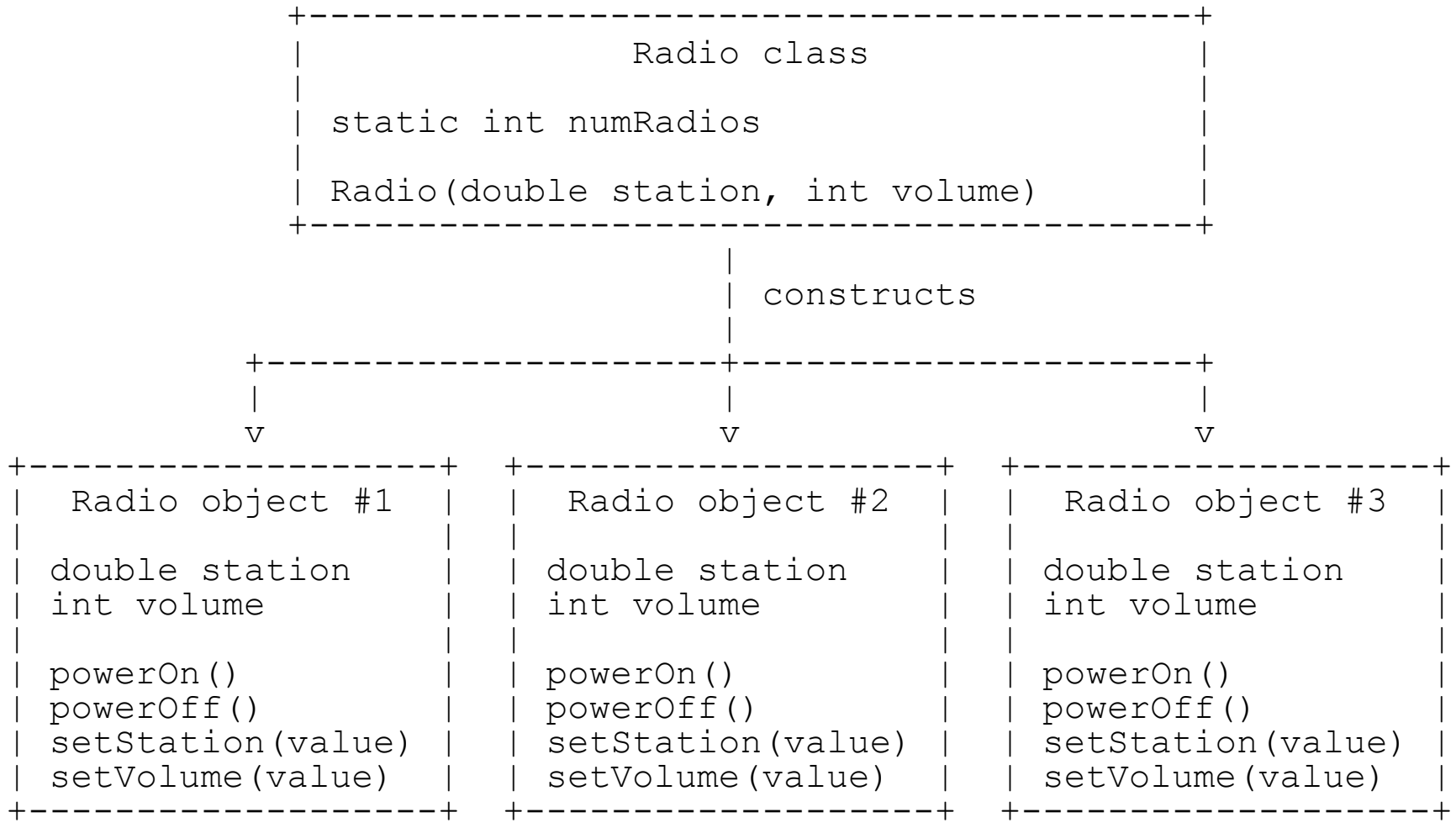
Creation of real objects

- In real life, a factory can create many similar 'objects':



Creation of Java objects

- The analogous entity in Java to the 'factory' is a class.

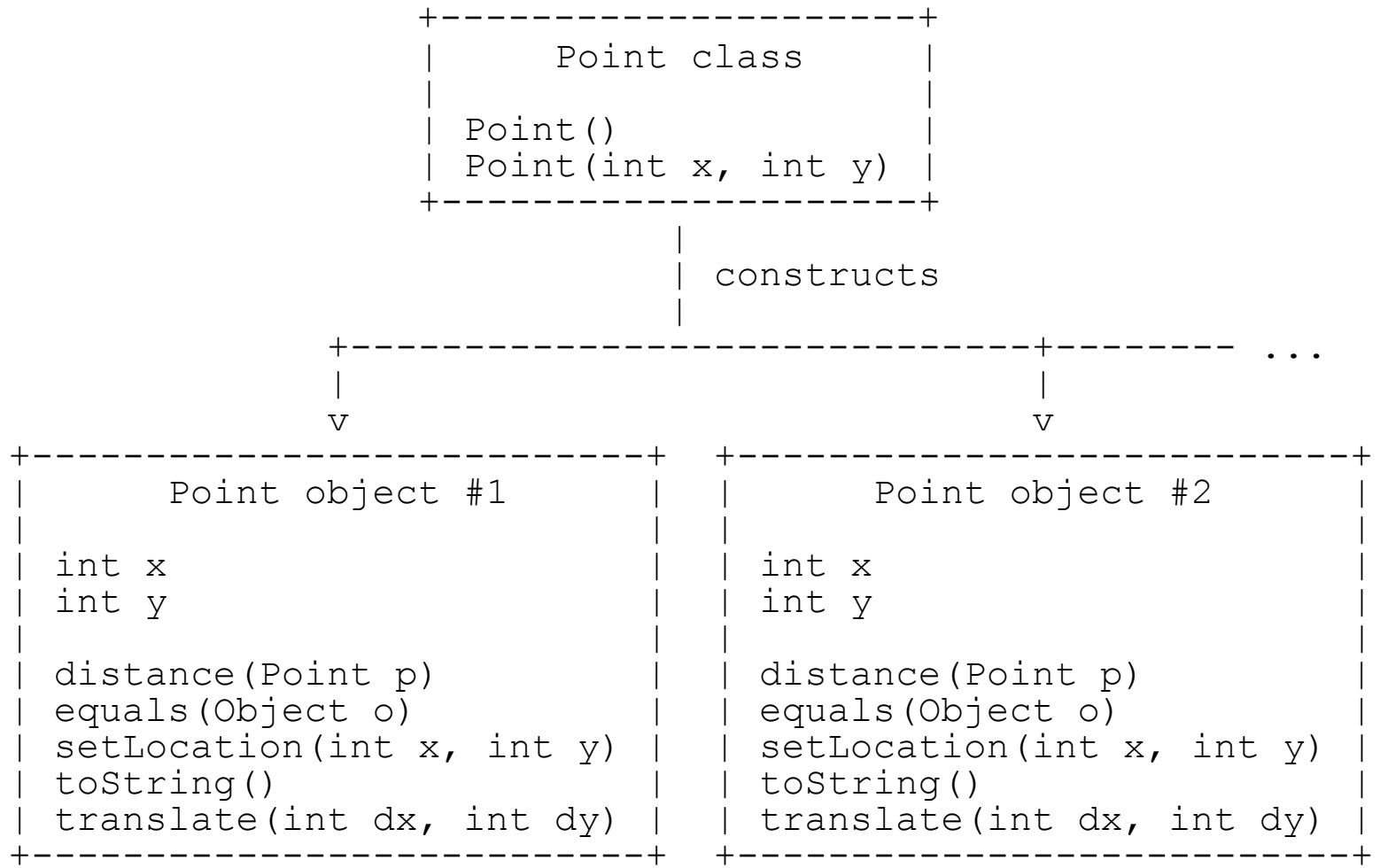


Classes, types, objects

- **class**:
  1. A file that can be run as a program, containing static methods and global constants.
  2. **A template for a type of objects.**
- We can write Java classes that are not programs in themselves, but instead are definitions of new types of objects.
  - We can use these objects in other programs.
- Why would we want to do this?
  - It could be useful to create the new type of objects because it is a valuable abstraction that we can use in another program.
  - a way of *managing complexity*

A Point class

- A class of Points might look something like this:
  - Each object contains its own data and methods.
  - The class has the instructions for how to construct individual objects.



A simple class (data fields)

- The following class creates a new type of objects named Point.
  - Each object contains two pieces of data:
    - an int named x,
    - and an int named y.
  - Point objects (so far) do not contain any behavior.

```
public class Point {
    int x;
    int y;
}
```

  - We'd save the above into a file named `Point.java`.

Data fields

- **data field**: A variable declared inside an object.
  - Each object of our type will have its own copy of the data field.

- Declaring a data field, general syntax:

  <type> <name> ***;***

  or,

  <type> <name> = ***<value>*** ***;***

  - Example:

```
public class Student {
      // each student object has a
      // name and gpa data field
     String name;
     double gpa;
}
```

Accessing data fields

- Code in another class can access your object's data fields. (for now)
  - Later in this chapter, we'll learn about *encapsulation*, which will change the way we access the data inside objects.

- Accessing or modifying a data field, general syntax:

  <variable name> . <field name>

  or

  <variable name> . <field name> = ***<value>*** ***;***

  - Examples:

```
System.out.println("the x-coord is " + p1.x);
p2.y = 13;
```

Client code: Using Point class

- The following code (stored in `UsePoint.java`) uses our Point class.
- **client code**: Code that uses our objects.

```
public class UsePoint {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 5;
        p1.y = -2;
        Point p2 = new Point();
        p2.x = -4;
        p2.y = 3;

        // print each point
        System.out.println("(" + p1.x + ", " + p1.y + ")");
        System.out.println("(" + p2.x + ", " + p2.y + ")");
    }
}
```

OUTPUT:

```
(5, -2)
(-4, 3)
```

Class with behavior (method)

- This second version of Point gives a method named translate to each Point object.
  - Each Point object now contains one method of behavior, which modifies its x and y coordinates by the given parameter values.

```
public class Point {
    int x;
    int y;

    public void translate(int dx, int dy) {
        this.x += dx;
        this.y += dy;
    }
}
```

  - Note the use of the keyword `this` which allows the Point object to refer to itself.

Alternate syntax

- Instead of using `this.` we could just refer to x.
- What is the advantage of using `this.` ?

```
public class Point {
    int x;
    int y;

    public void translate(int dx, int dy) {
        x += dx;
        y += dy;
    }
   // also works in this case
}
```

Object 'context' and `this`

- The code for a method of a type of objects executes in the 'context' or perspective of a particular object.
  - A special keyword named `this` exists, which lets you refer to the object on which the method is running.
  - Using the `this` keyword lets you examine, print, or modify the values of that object's data fields.
  - The `this` is sometimes called the 'implicit parameter.'

```
// I'm a Point, and I'm being asked to adjust my
// x and y values by the given amounts.
public void translate(int dx, int dy) {
    this.x += dx;  // change 'my' x value
    this.y += dy;  // change 'my' y value
}
```

The meaning of `static`

- It is illegal to use the `this` keyword in a static method, because static code doesn't operate in the context of an object.
- You'll get a "cannot use keyword `this` from a static context" error.
- So what does `static` mean?
  - Look at methods from the `String` class and the `Math` class.

Objects' methods

- Methods of objects (methods without the `static` keyword) define the behavior for each object.
  - The object can use the `this` keyword to refer to its own fields or methods as necessary.

- **mutator**: A method that modifies the state of the object in some way.
  - Sometimes the modification is based on parameters that are passed to the mutator method, such as a setX method with an int x parameter.
  - The translate method is an example of a mutator.

Object method syntax

- Declaring an object's method, general syntax:

```
public <type> <name> ( <parameter(s)> ) {
    <statement(s)> ;
}
```

  - Example:

```
public void setLocation(int x, int y) {
    this.x = x;
    this.y = y;
}
```

  - Notice again how the object uses the keyword `this` when referring to its own data field variables.

Logic error with no this

- Example:

```
public void setLocation(int x, int y) {
    x = x;
    y = y;
}
```

- I want to set the Point object's x equal to the value of the setLocation method's parameter named x and likewise for y.
- But I have an identifier with the same name as a field of the object.
- This "shadows" the objects field and the this. is required to refer to the object's x.

Client code (2)

- The following client code (stored in `UsePoint2.java`) uses our modified Point class:

```
public class UsePoint2 {
    public static void main(String[] args) {
        Point p = new Point();
        p.x = 3;
        p.y = 8;
        p.translate(2, -1);

        System.out.println("(" + p.x + ", " + p.y + ")");
    }
}
```

OUTPUT:

```
(5, 7)
```

Constructors

- It is tedious to have to construct an object and assign values to all of its data fields manually.

```
Point p = new Point();
p.x = 3;
p.y = 8;                          // tedious
```

- We'd rather be able to pass in the data fields' values as parameters, as was possible with Java's built-in Point type.

```
Point p = new Point(3, 8);  // better!
```

- To do this, we need to learn about a special type of method called a *constructor*.

Point class w/ constructor

- **constructor**: A method that specifies how to initialize the state of a new object.
  - Constructors may have parameters to initialize the object.

```
public class Point {
    int x;
    int y;

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

    public void translate(int dx, int dy) {
        this.x += dx;
        this.y += dy;
    }
}
```

Constructor syntax

- Constructor, general syntax:

```
public <type> ( <parameter(s)> ) {
    <statement(s)> ;
}
```

  - Example:

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

- Note that the parameters to the constructor can have the same name as the object's data fields.
  - Java doesn't get confused by this, if we refer to the data fields with the `this.` notation.
  - A constructor doesn't need to specify a return type (not even `void`) because it implicitly returns a new Point object.

Client code (3)

- The following client code (stored in `UsePoint3.java`) uses our new Point class with constructor:

```
public class UsePoint3 {
    public static void main(String[] args) {
        Point p = new Point(3, 8);
        p.translate(2, -1);

        System.out.println("(" + p.x + ", " + p.y + ")");
    }
}
```

OUTPUT:

```
(5, 7)
```