Advanced Programming Techniques in Java

COSI 12B

Review: 2D Arrays

- **Declaring** and **creating** a 2D array:

```
<type> [][]arrayName = new <type> [<row>][<column>]
```

```
int[][] score = new int[5][8];
```

Number of rows

Number of columns

- To **access** an element: `arrayName[<row>][<column>]`

`score[3][4]` will give you the value at row 3, column 4

score

	0	1	2	3	4	5	6	7
0	15	8	3	16	12	7	9	5
1	6	11	9	4	1	5	8	13
2	17	3	5	18	10	6	7	21
3	8	14	13	6	13	12	8	4
4	1	9	5	16	20	2	3	9

Review: Wrapper classes

Primitive Type	Wrapper Type
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`float`	`Float`
`boolean`	`Boolean`

- A wrapper is an object whose sole purpose is to hold a primitive value
- Once you construct the list, use it with primitives as normal

Review: Bubble Sort

```
public static void bubbleSort(int[] arr){
    int didswap = 1, tmp = 0;
    while (didswap == 1) {
        didswap = 0;
        for (int i = 1; i < arr.length; i++) {
                if (arr[i - 1] > arr[i]) {
                        tmp = arr[i - 1];
                        arr[i - 1] = arr[i];
                        arr[i] = tmp;
                        didswap = 1;
                }
        }
    }
}
```

Class objectives

- Intro to Object Oriented Design (Section 8.1?)

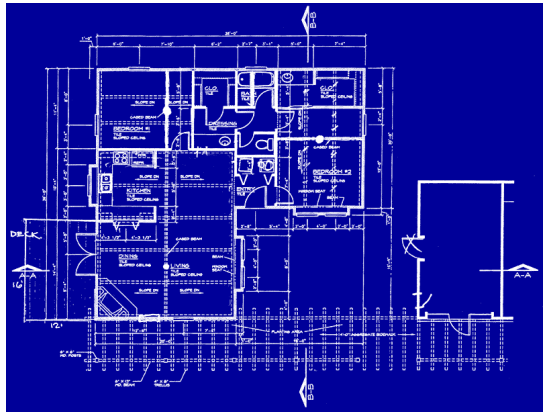
Classes and Objects

Review: State and Behavior

- **Definition**
  - A **state** is a set of values (internal data) stored in an object
- **Definition**
  - A **behavior** is a set of actions an object can perform, often reporting or modifying its internal state

Review: What is a Class?

- **Definition**
  - A **class** is like a blueprint (defined by the user) for which objects are created
    - It is a definition of a new type of objects



- The objects of a given class are built according to its blueprint
- Objects of a class are referred to as instance of the class

Review: Object State: Fields

- **Definition**
  - A field is a variable inside an object that makes up part of its state

- **Syntax**

```
<type> <name>;
```

- **Example**

```
public class Student{
      String name;
      double gpa;
}
```

Each `Student` object has a `name` and `gpa` field

Review: Constructing objects

- **Construct**: To create a new object
  - Objects are constructed with the **`new`** keyword
  - Most objects must be constructed before they can be used

- **Syntax**

```
<type> <name> = new <type> ( <parameters> );
```

- **Example**:

```
Point p = new Point();
```

- Strings are also objects, but can be constructed without new

```
String name = "Amanda Ann Camp";
```

Review: `Point` Class (ver. 1)

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

Point.java

- The `Point` class isn't itself an executable program
- Objects themselves are not complete programs
  - They can only be used as part of larger programs to solve problems
- The program that creates and uses objects is known as **client code**

Client Code

- **Definition**
  - A **client code** is the code that interacts with a class or objects of that class
- The way a client code interacts with the objects is by sending messages to them and asking them to perform behavior

- Remember the objects from the built-in classes `String`, `Scanner`, `File`, `Random`, etc.
  - You and your programs have been clients of these objects

A Class and its Client

- `Point.java` is not, by itself, a runnable program
  - A class can be used by client programs

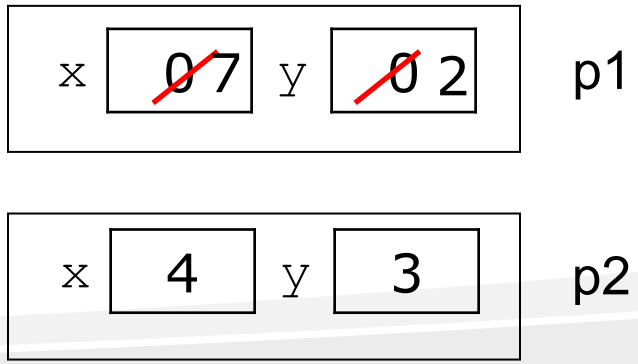
`PointMain.java` **(client program)**

```
public class PointMain {
  ... main(String[] args) {
    Point p1 = new Point();
    p1.x = 7;
    p1.y = 2;

    Point p2 = new Point();
    p2.x = 4;
    p2.y = 3;
    ...
  }
}
```

`Point.java` **(class)**

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



Client Program for the Point Class (ver. 1)

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

        // print p1
        System.out.println(p1.x + "," + p1.y);

        // move p2 and then print it
        p2.x += 2;
        p2.y++;
        System.out.println(p2.x + "," + p2.y);
    }
}
```

PointMain.java

Client Program for the `Point` Class (ver. 2)

PointMain.java

```
public class PointMain {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 7;
        p1.y = 2;

        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;

         // print the points
        System.out.println("p1 is (" + p1.x + "," + p1.y + ")");
        System.out.println("p2 is (" + p2.x + "," + p2.y + ")");

        // translate each point to a new location
        p1.x += 11;
        p1.y += 6;
        p2.x += 1;
        p2.y += 7;

        // print the points
        System.out.println("p1 is (" + p1.x + "," + p1.y + ")");
        System.out.println("p2 is (" + p2.x + "," + p2.y + ")");
    }
}
```

Client Program for the `Point` Class (ver. 2)

PointMain.java

```
public class PointMain {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 7;
        p1.y = 2;

        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;

           // print the points
        System.out.println("p1 is (" + p1.x + "," + p1.y + ")");
        System.out.println("p2 is (" + p2.x + "," + p2.y + ")");

        // translate each point to a new location
        p1.x += 11;
        p1.y += 6;
        p2.x += 1;
        p2.y += 7;

        // print the points
        System.out.println("p1 is (" + p1.x + "," + p1.y + ")");
        System.out.println("p2 is (" + p2.x + "," + p2.y + ")");
    }
}
```

Not the best way to implement this

Client Program for the `Point` Class (ver. 2)

- The client program has some redundancy
- Translating a point is a common operation, we should represent it as a method

```
//A static method to translate a Point
public static void translate(Point p, int dx, int dy){
        p.x += dx;
        p.y += dy;
}
```

```
//Method call
translate(p1, 11, 6)
```

Client Program for the `Point` Class (ver. 2)

```
public class PointMain {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 7;
        p1.y = 2;

        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;

        // print the points
        System.out.println("p1 is (" + p1.x + "," + p1.y + ")");
        System.out.println("p2 is (" + p2.x + "," + p2.y + ")");

        // translate each point to a new location
        translate (p1, 11, 6);
        translate (p2, 1, 7);

        // print the points
        System.out.println("p1 is (" + p1.x + "," + p1.y + ")");
        System.out.println("p2 is (" + p2.x + "," + p2.y + ")");
    }

    // static method to translate a Point
    public static void translate(Point p, int dx, int dy){
        p.x += dx;
        p.y += dy;
    }

}
```

PointMain.java

Problem with Static Methods

- We are missing a major benefit of objects: code reuse
  - Every program that uses `Point` objects would need a translate method
- So far, our `Point` class contains state, but no behavior
- The reason of classes is to combine state and behavior
  - The `translate` method belong inside each `Point` object

Instance Methods

Object Behavior: Methods

- **Definition**
  - An **instance method** (or **object method**) is a method that exists inside each object of a class and gives behavior to each object
- **Syntax**

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

- **Example**

```
public void shout() {
     System.out.println("HELLO THERE!");
}
```

- Same syntax as static methods, but without `static` keyword

Point Class (ver. 2)

Point.java

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

    // shifts this point;s location by the given amount
    public void translate(int dx, int dy){
        x += dx;
        y += dy;
   }
}
```

- The translate method no longer has a `Point p` parameter
  - How does the method know which point to translate?
  - How does the method access the point's `x` and `y` data?

Calling Instance Methods

- Objects contain methods that can be called by your program
  - When we call an object's method, we are sending a message to it
  - We must specify which object we are talking to, and then write the method's name
- **Syntax**

  <object name>.<method name>(<parameters)>

  The result will be different from one object to another
- **Example**

```
String s1 = ”Iraklis";
String s2 = "Antonella";
System.out.println(s1.length());     // 7
System.out.println(s2.length());     // 9
```

`toString()` Method

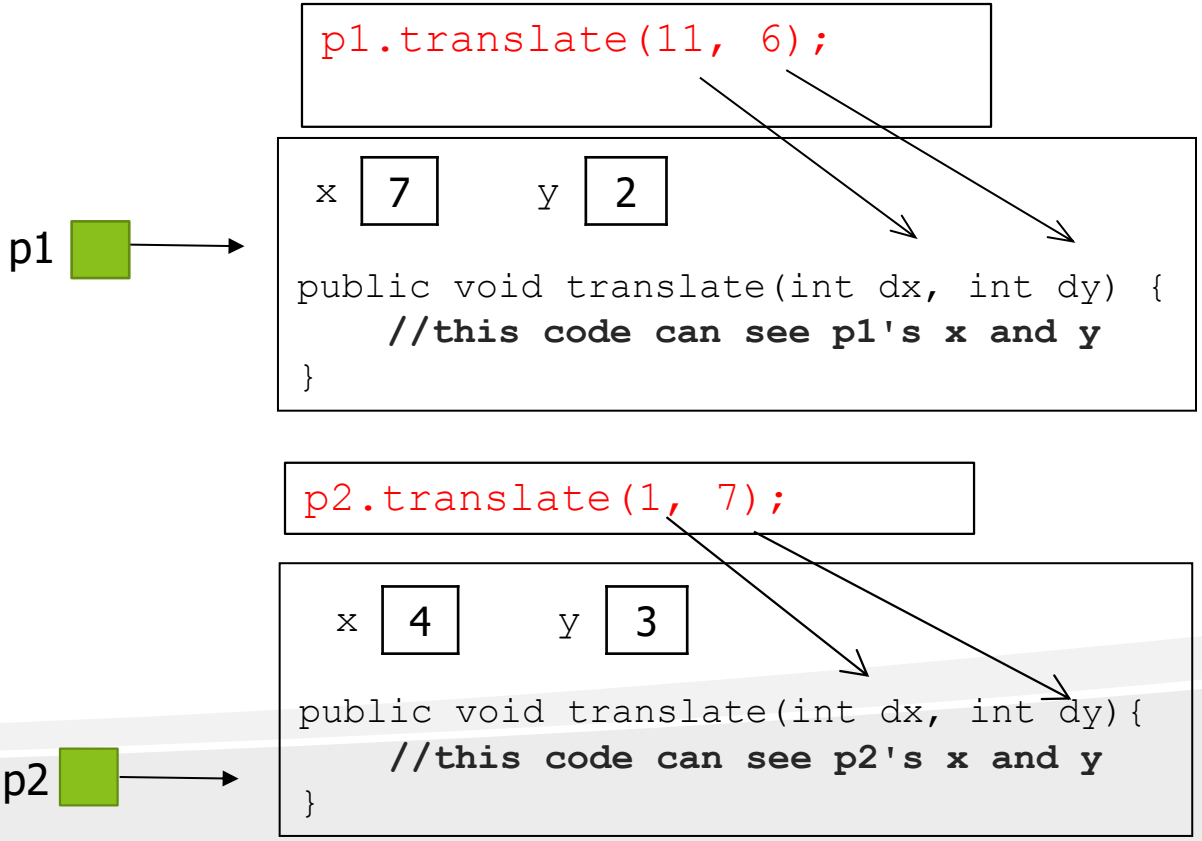
`Point` Objects with Method

- Each `Point` object has its own copy of the translate method, which operates in that object's state

```
Point p1 = new Point();
p1.x = 7;
p1.y = 2

Point p2 = new Point();
p2.x = 4;
p2.y = 3;

p1.translate(11, 6);
p2.translate(1, 7);
```



Implicit Parameter

- **Definition**
  - An **implicit parameter** is the object on which an instance method is called
- During the call `p1.translate(11, 6)`
  - The object referred to by `p1` is the implicit parameter
- During the call `p2.translate(1, 7)`
  - The object refereed to by `p2` is the implicit parameter

Printing Objects

- By default, Java doesn't know how to print objects:

```
Point p = new Point();
p.x = 10;
p.y = 7;
System.out.println("p is " + p);  //p is Point@9e8c34

//better, but cumbersome     p is (10, 7)
System.out.println("p is (" + p.x + ", " + p.y + ")");

//desired behavior
System.out.println("p is " + p);  //p is (10, 7)
```

The `toString` Method

- When a Java program is printing an object or concatenating an object to a `String`, it calls a special method called **`toString()`**
- The `toString()` method tells Java how to convert an object into a `String`

```
Point p1 = new Point(7, 2);
System.out.println("p1: " + p1);

//the above code is really calling the following:
System.out.println("p1: " + p1.toString());
```

- Every class has a `toString` method, even if it's not in your code
  - Default: class's name @ object's memory address (base 16) `Point@9e8c34`

The `toString` Method (cont.)

- **Syntax**

```
public String toString() {
        code that returns a String representing this object;
}
```

- **Example**

```
//Returns a String representing this Point
public String toString() {
     return "(" + x + ", " + y + ")";
}
```

- Method name, return, and parameters must match exactly

The `toString` Method (cont.)

```
public class Point{
    int x;
    int y;
    …
    …
    public String toString(){
       return "(" + x + ", " + y + ")";
    }
```

Point class

```
…
int i = 42;
String s = "hello";
Point p = new Point();

System.out.println("i is " + i);
SysSystem.out.println("s is " + s);
System.out.println("p is " + p);
…
```

Client code

The `toString` Method Facts

- It is recommended to write a `toString()` method in every class you write
- Do not place `println` statements in the `toString()` method
  - `toString()` simply return a String that the client can use in a `println` statement
- Keep in mind that well formed classes of objects do not contain any `println` statement at all

Point Class (ver. 3)

Point.java

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

    // shifts points location by the given amount
    public void translate (int dx, int dy){
        x += dx;
        y += dy;
    }

    // toString method
    public String toString(){
        return "(" + x + " , " + y + ")";
   }
}
```

PointMain.java(ver. 3)

PointMain.java

```
public class PointMain {
    public static void main(String[] args){

        // Create two Point objects
        Point p1 = new Point();
        p1.x = 7;
        p1.y = 2;

        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;


        // Translate p1
        p1.translate(11, 6);
        System.out.println("p1 is " + p1);
    }
}
```

Constructor

Object Initialization

- To use a variable (of either primitive or reference type) you need to declare its data type and name

- **Example**

```
int x;
Point p;
```

- Before you use a variable (of either primitive or reference type) you must initialize it

- Currently it takes 3 lines to create a Point and initialize it

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

Object Initialization (cont.)

- We'd rather specify the fields' initial values at the start:

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

- Such statement is not legal for our `Point` class, because we don't have any code that specifies how to create a point with initial (x, y) location

Constructor

- **Definition**
  - A **constructor** initialize the state of a new object
- **Syntax**

```
public <class name>(<type> <name>, …, <type> <name>) {
            statement(s);
}
```

- **Example**

```
//Constructs a new point with given location
public Point(int initialX, int initialY) {
          x = initialX;
          y = initialY;
}
```

Constructor

- The constructor run when the client uses the `new` keyword
- No return type is specified, it implicitly "returns" the new object being created
- If a class has no constructor, Java supplies a default constructor with no parameter
  - The default constructor initialize all fields to zero-equivalent values

```
public <class name>(<type> <name>, …, <type> <name>) {
            statement(s);
}
```

Point Class (ver. 4) with Constructor

Point.java

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

    // constructs a new point with the given (x, y) location
    public Point(int initialX, int initialY){
        x = initialX;
        y = initialY;
    }


    // shifts points location by the given amount
     public void translate (int dx, int dy){
        x += dx;
        y += dy;
     }

     // toString method
     public String toString(){
        return "(" + x + " , " + y + ")";
    }
}
```

same as the class's name

Constructors could also call class methods

Once you write your own constructor
Java will NOT supply the default one

PointMain.java (ver. 4)

PointMain.java

```
public class PointMain {
   public static void main(String[] args){
        //Create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);
        //Print each point
        System.out.println("p1 is "+ p1);
        System.out.println("p2 is "+ p2);
        //Translate each point to a new location
        p1.translate(11, 6);
        p2.translate(1, 7);
        //Print the points again
        System.out.println("p1 is "+ p1);
        System.out.println("p2 is "+ p2);
   }
}
```