Advanced Programming Techniques in Java

Object Oriented Programming

Lecture 9

Class Objectives

- Constructor methods (Section 8.3)
- Add more behavior to Point (Section 8.2)

- equals()
  - `this` keyword (Section 8.3)

Review: Object Behavior: Method

- **Definition**
- An **instance method** (or **object method**) is a meth
class and gives behavior to each object
- **Syntax** `public <type> <name>(<type> <name>,`

```
 }
```

- **Example**

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

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

Review: The `toString` Meth

- **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

Review: The `toString` Method

- It is recommended to write a `toString()` method
- Do not place `println` statements in the `toStrin`
- `toString()` simply return a String that the client can us
- Keep in mind that well formed classes of objects do all

Review: Constructor

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

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

- **Example**

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

Review: Constructor

- The constructor run when the client uses the `new`
- No return type is specified, it implicitly "returns" the
- If a class has no constructor, Java supplies a defau
- The default constructor initialize all fields to zero-equival

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

```
public class Point{
    int x;
    int y;                                  same as the class's n

    // constructs a new point with the
    public Point(int initialX, int init

            y = initialY;
    }

    // shifts points location by the gi
     public void translate (int dx, int
     dx; y += dy;
     }
```

Once
Java

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

`Point` Class (ver. 4) with Const

PointMain.java (ver. 4)

```
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 locati
          p1.translate(11, 6);
          p2.translate(1, 7);
          //Print the points again
          System.out.println("p1 is "+ p1);
          System.out.println("p2 is "+ p2);
   }
```

```
}
```

PointMain.java (ver. 4)

```
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 locati
          p1.translate(11, 6);
          p2.translate(1, 7);
          //Print the points again
          System.out.println("p1 is "+ p1);
          System.out.println("p2 is "+ p2);
   }
```

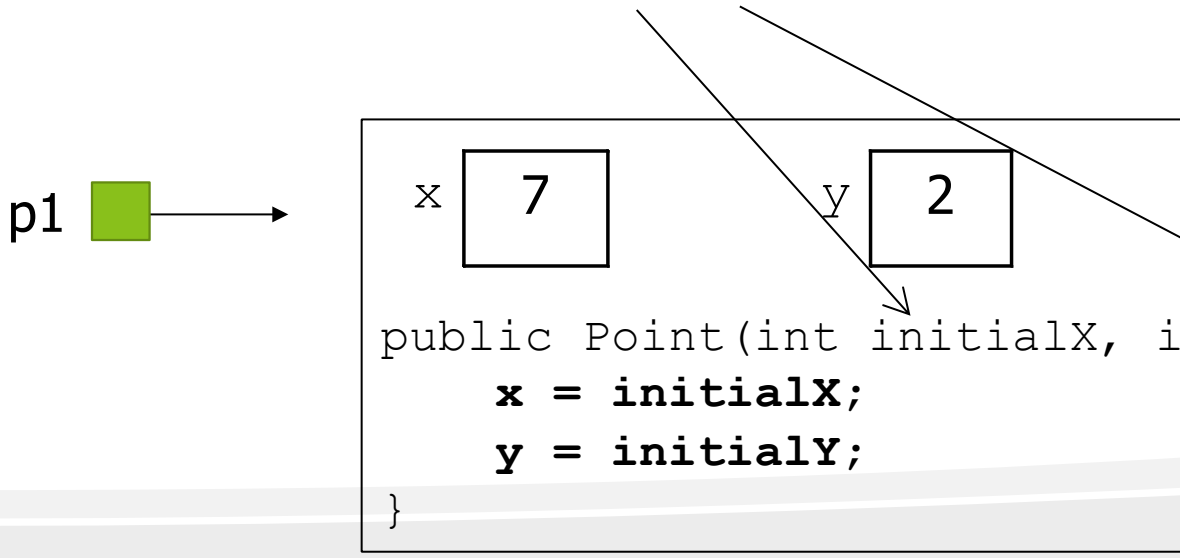
```
}
```

- Note: `Point p1 = new Point();` **//will not**

Tracing a Constructor Call

- What happens when the following call is made?

```
Point p1 = new Point(7, 2);
```



Multiple Constructors

- A class can have multiple constructors to provide m
objects of that class
- Each constructor must accept a unique set of parameters
- Write a `Point` constructor with no parameters that

```
//Construct a Point at (0,0) loca
public Point() {
        x = 0;
        y = 0;
}
```

```
//Create two Point objects
```

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

`Point` Class (ver. 4)

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

   // constructs a new point with the given (x,
   public Point(int initialX, int initialY){
           x = initialX;
           y = initialY;
   }
   public Point(){
           x = 0;
           y = 0;
   }
    // shifts points location by the given amoun
    public void translate (int dx, int dy){
           x += dx;
           y += dy;
    }
```

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

Common Programming Bugs

- Using `void` with a constructor

```
//Construct a Point at the given x and y
public void Point(int initialX, int init
        x = initialX;
        y = initialY;
}
```

- Constructors aren't supposed to have return types
- Tough to catch, because the `Point.java` file still
successfully
- Re-declaring fields in a constructor

Common Programming Bugs

```
//Construct a Point at the given x a
public Point(int initialX, int initi
         int x = initialX;
int y = initialY; }
```

- Behaves in an odd way
- It compiles successfully, but when the client code c
object its initial coordinates are always (0, 0)

Why?

- Re-declaring fields in a constructor

Common Programming Bugs

```
//Construct a Point at the given x a
public Point(int initialX, int initi
         int x = initialX;
int y = initialY; }
```

- Behaves in an odd way
- It compiles successfully, but when the client code c
object its initial coordinates are always (0, 0)

Why?

- We say that these local x and y variables **shadow** c

Add more methods

- Write a method `setLocation` that changes a Poi
  passed

```
public void setLocation(int newX,
          x = newX;
y = newY; }
```

- Write an alternative method `translate` that uses

```
public void translate (int dx, int
dy){ setLocation(x + dx, y + dy);
```

Add more methods (cont.)

- Write a method `distance` that computes the distar
  `Point` parameter

```
public double distance(Point o
        int dx = x - other.x;
        int dy = y - other.y;
        return Math.sqrt(dx *
}
```

- Write a method `distanceFromOrigin` that return
  the origin, (0, 0)

```
public double distanceFromOrig
return Math.sqrt(x * x + y * y
```

Mutators and Accessors

- **Definition** A **mutator** is an instance method that
- Examples: `setLocation`, `translate`
- Has a void return type

- **Definition** An **accessor** is an instance method th of an object without modifying it
- Examples: `distance`, `distanceFromOrigin`
- Often has a non-void return type

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

  // constructor
  public Point(int initialX, int initialY){
    x = initialX; y = initialY;
  }

  // constructor
  public Point(){
  x = 0; y = 0;
  }

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

  // computes the distance between two points
  public double distance(Point other){ int dx
  = x - other.x; int dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
} …
```

```
…
  /
  pu
  Po
  }

  pu
  "
}
```

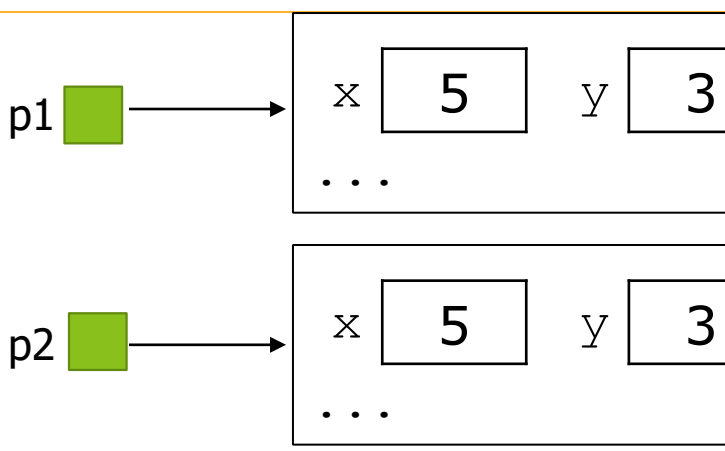
Poi

`equals()` M

Comparing objects

- The `==` operator does not work well with objects
- `==` compares references to objects and only evaluates t
object (it doesn't tell us whether two objects have the sar
- **Example**:

```
Point p1 = new Point(5, 3);
Point p2 = new Point(5, 3);
if (p1 == p2) {  // false
    System.out.println("equal")
}
```



The `equals` method

- The `equals` method compares the state of objects

- When we write our own classes of objects, Java doesn't kn
- The default `equals` behavior acts just like the `==` o

```
if (p1.equals(p2)) {  // still
    System.out.println("equal")
}
```

- We can replace this default behavior by writing an `eq`
- The method will compare the state of the two objects and

Initial `equals` method

- This is one implementation of the equals method fo

```
public boolean equals(Point other) { if
     (x == other.x && y == other.y) {
     return true;
     } else { return
           false;
     }
}
```

- Do we like this method?

Initial `equals` method

```
public boolean equals(Point other) { if
     (x == other.x && y == other.y) {
     return true;
     } else { return
           false;
     }
}
```

- This is one implementation of the equals method fo

- Do we like this method?

Initial flawed `equals` method

- You might think that the following is a valid implem

```
public boolean equals(Point other) { if
     (x == other.x && y == other.y) {
     return true;
     } else { return
           false;
     }
}
```

- However, it has several flaws that we should corre
- One initial improvement: the body can be shortene

```
other.y;
```

```
public boolean equals(Point other) {
return x == other.x && y == other.y; }
```

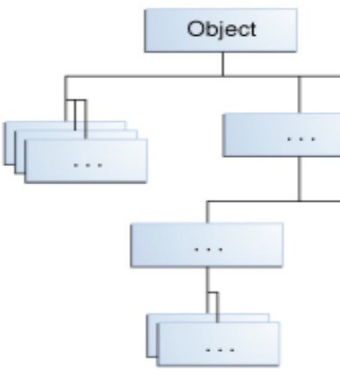
equals and the Object class

- The `equals` method should not accept a paramet

  should be legal to compare `Points` to any other ol

```
Point p = new Point(7, 2); if
(p.equals("hello")) {   // false
...
}
```

- The parameter to a proper equals method must be
that an object of any type can be passed)



The Object class

- The Object class sits at the top of every class in the
- It defines the basic state and behavior that all object
compare oneself to another object, to convert to a st

* We will talk more about the Object class later

`equals` and the `Object` class

- **Syntax**:

```
public boolean equals(Object <name>) {
        <statement(s) that return a boolean value> ;
}
```

```
public boolean equals(Object o) {
return x == o.x && y == o.y;
}
```

Another flawed version

- You might think that the following is a valid implem

or

```
public boolean equals(Object o) {
      if (x == o.x && y == o.y) {
      return true;
      } else { return
             false;
      }
}
```

```
pub
ret
```

- However, it does not compile

```
Point.java:36: cannot find symbol
symbol  : variable x location:
class java.lang.Object if (x ==
o.x && y == o.y) {
          ^
```

Type-casting objects

- The object that is passed to the `equals` method c
class's type
- **<u>Example</u>**:

```
public boolean equals(Object o) {
    Point other = (Point) o;
    return x == other.x && y == oth
}
```

- Type-casting with objects behaves differently than
- We are really casting a reference of type `Object` into a
- We're promising the compiler that `o` refers to a `Point`

Comparing different types

- Currently when we compare `Point` objects to any

```
Point p = new Point(7, 2); if
(p.equals("hello")) {    // false
...
```

```
}
```

- The code crashes with the following exception:

```
Exception in thread "main"
java.lang.ClassCastException: java.lang.String
at Point.equals(Point.java:25) at
PointMain.main(PointMain.java:25)
```

- The culprit is the following line that contains the type

```
equals(Object o) {
            Point other = (Point) o;
```

The `instanceof` keyword

- We can use a keyword called `instanceof` to ask
  of a given type
- **Syntax**: `<variable> instanceof <type>`
- The above is a `boolean` expression that can be used a
- **Example**:

`s insta`
`s insta`

```
String s = "hello";

Point p = new Point();
```

p insta
p insta
null in

Final version of `equals` method

- This version of the `equals` method allows us to c
against any other type of object:

```
// Returns whether o refers to a P
// the same (x, y) coordinates as
object public boolean equals(Objec
instanceof Point) { Point other =
```

```
        return x == other.x && y =
    } else { return
        false;
    }
}
```

Template for your `equals()` m

```
public boolean equals (Object o){
   if (o instanceof <type>){
         <type> other = (<type>) o;
         //compare the state and re
   } else { return
   false;
   }
}
```

`this` keyw

Remember … Common Programm

- Re-declaring fields in a constructor

```
//Construct a Point at the given x and y l
public Point(int initialX, int initialY) {
          int x = initialX;
int y = initialY; }
```

- Behaves in an odd way
- It compiles successfully, but when the client code co
coordinates are always (0, 0)

Why?

- We say that these local x and y variables **shadow** o

Variable shadowing

- **Definition** **Shadowing** indicates two variables with
- Normally illegal, except when one variable is a field

```
public class Point
    { int x; int y;
    ...
    // this is legal
    public void setLocation(int x, int y) {
    }
```

- In most of the class, `x` and `y` refer to the fields

- In `setLocation`, `x` and `y` refer to the method's p

Fixing shadowing

- Use the keyword **this**

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

- Inside `setLocation`,
- To refer to the data field `x`, say `this.x`
- To refer to the parameter `x`,say `x`

The `this` keyword

- **Definition** The **`this`** keyword refers to the curren
- The `this` keyword is used to eliminate confusion parameters with the same name
  - Refer to a field: `this`.**field**
  - Call a method: `this`.**method**(**paramet**

- One constructor can call another: `this(parameters);`
- So far, the compiler was converting expressions au
- `x → this.x`
- `setLocation(10,12) → this.setLocation(10,`

Programming style: shadowing is

```
public void setLocation(int newX, int new
          x = newX;
y = newY;    }
```

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

- Clearer style
- Matches client code that call methods as `objec`
- You don't have to invent new variable names

The `this` keyword

- Using `this` with a constructor

- From within a constructor, you can also use the `this` key
class

```
public class Point {
  int x; int y; public Point() { this(0, 0);
  // calls (x, y) constructor
  }
  public Point(int x, int y) {
        setLocation(x,y);
  }
  ...
}
```

- Avoids redundancy between constructors
- Only a constructor (not a method) can call another c

- You cannot call `Point(0,0)`, it is illegal

Exercise

- Write a constructor that accepts a Point as a para
have the same (x,y) values

Exercise

- Write a constructor that accepts a Point as a parar
  have the same (x,y) values
- **Option 1**

```
public Point(Point p){
    //you have access to x, y directly
    this.x = p.x; this.y = p.y;
}
```

- **Option 2, preferable**

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