

A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar. The wall is partially visible, extending from the left edge towards the center of the frame.

Building Java Programs

Chapter 8: Classes and Objects

Lecture outline

- the keyword `this`
 - multiple constructors
- static fields and methods in a class

A brick wall is visible on the left side of the slide, extending from the bottom to the top. The bricks are reddish-brown with white mortar. The background is a solid blue color.

The keyword *this*

reading: 8.7

Using the keyword `this`

- **`this`** : A reference to the implicit parameter.
 - *implicit parameter*: object on which a method/constructor is called
- `this` keyword, general syntax:
 - To refer to a field:
`this.<field name>`
 - To call a method:
`this.<method name>(<parameters>) ;`
 - To call a constructor from another constructor:
`this(<parameters>) ;`

Variable names and scope

- Usually it is illegal to have two variables in the same scope with the same name.
- Recall: Point class's setLocation method:
 - Params named `newX` and `newY` to be distinct from fields `x` and `y`

```
public class Point {
    int x;
    int y;
    ...
    public void setLocation(int newX, int newY) {
        if (newX < 0 || newY < 0) {
            throw new IllegalArgumentException();
        }
        x = newX;
        y = newY;
    }
}
```

Variable shadowing

- However, a class's method can have a parameter whose name is the same as one of the class's fields.

- Example:

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

- Fields `x` and `y` are *shadowed* by parameters with same names.
- Any `setLocation` code that refers to `x` or `y` will use the parameter, not the field.

- **shadowed variable:** A field that is "covered up" by a parameter or local variable with the same name.

Avoiding shadowing with `this`

- The keyword `this` prevents shadowing:

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

    ...

    public void setLocation(int x, int y) {
        if (x < 0 || y < 0) {
            throw new IllegalArgumentException();
        }
        this.x = x;
        this.y = y;
    }
}
```

Inside the `setLocation` method:

- When `this.x` is seen, the *field* `x` is used.
- When `x` is seen, the *parameter* `x` is used.

Multiple constructors

- It is legal to have more than one constructor in a class.
 - The constructors must accept different parameters.

```
public class Point {  
    private int x;  
    private int y;  
  
    public Point() {  
        x = 0;  
        y = 0;  
    }  
  
    public Point(int initialX, int initialY) {  
        x = initialX;  
        y = initialY;  
    }  
  
    ...  
}
```


Multiple constructors w/ this

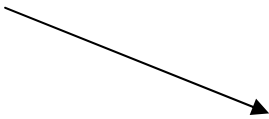
- One constructor can call another using `this`
 - We can also rename the parameters and use `this.` field syntax.

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

    public Point() {
        this(0, 0);    // calls the (x, y) constructor
    }

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

    ...
}
```



Static fields / methods

Static fields vs. fields

- **static:** Part of a class, rather than part of an object.
 - Classes can have static fields.
 - Unlike fields, static fields are not replicated into each object; instead a single field is shared by all objects of that class.
- static field, general syntax:

```
private static <type> <name>;
```

or,

```
private static <type> <name> = <value>;
```

- Example:

```
private static int count = 0;
```

Static field example

- Count the number of Husky objects created:

```
public class Husky implements Critter {  
    // count of Huskies created so far  
    private static int objectCount = 0;  
  
    private int number;    // each Husky has a number  
  
    public Husky() {  
        objectCount++;  
        number = objectCount;  
    }  
  
    ...  
  
    public String toString() {  
        return "I am Husky #" + number +  
            "out of " + objectCount;  
    }  
}
```

Static methods

- **static method:** One that's part of a class, not part of an object.
 - good places to put code related to a class, but not directly related to each object's state
 - shared by all objects of that class
 - does not understand the *implicit parameter*; therefore, cannot access fields directly
 - if `public`, can be called from inside or outside the class
- Declaration syntax: *(same as we have seen before)*

```
public static <return type> <name>( <params> ) {  
    <statements> ;  
}
```

Static method example 1

- Java's built-in `Math` class has code that looks like this:

```
public class Math {  
    ...  
    public static int abs(int a) {  
        if (a >= 0) {  
            return a;  
        } else {  
            return -a;  
        }  
    }  
  
    public static int max(int a, int b) {  
        if (a >= b) {  
            return a;  
        } else {  
            return b;  
        }  
    }  
}
```

Static method example 2

- Adding a static method to our Point class:

```
public class Point {
    ...
    // Converts a String such as "(5, -2)" to a Point.
    // Pre: s must be in valid format.
    public static Point parse(String s) {
        s = s.substring(1, s.length() - 1); // "5, -2"
        s = s.replaceAll(",", ""); // "5 -2"
        // break apart the tokens, convert to ints
        Scanner scan = new Scanner(s);
        int x = scan.nextInt(); // 5
        int y = scan.nextInt(); // 2
        Point p = new Point(x, y);
        return p;
    }
}
```

Calling static methods, outside

- Static method call syntax (*outside* the class):

<class name> . <method name> (<values>) ;

- This is the syntax client code uses to call a static method.

- Examples:

```
int absVal = Math.max(5, 7);
```

```
Point p3 = Point.parse("(-17, 52)");
```


Calling static methods, inside

- Static method call syntax (*inside* the class):

`<method name> (<values>) ;`

- This is the syntax the class uses to call its own static method.
- Example:

```
public class Math {  
    // other methods such as ceil, floor, abs, etc.  
    // ...  
    public static int round(double d) {  
        if (d - (int) d >= 0.5) {  
            return ceil(d);  
        } else {  
            return floor(d);  
        }  
    }  
}
```