Chapter 6

Constructing Objects Part 1

Class contains

1) Variables (also called **data fields**)

2) Constructors

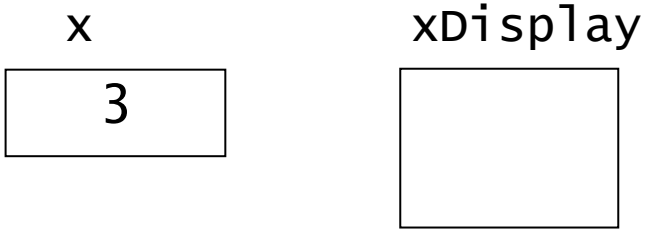
3) Methods

Illustrative class

```
 1 class OOP1
 2 {
 3     private static int x = 3;
 4     private int y;
 5     //-----------------------------------
 6     public OOP1(int yy)
 7     {
 8        y = yy;
 9     }
10     //-----------------------------------
11     public static void xDisplay()
12     {
13        System.out.println(x);
14     }
15     //-----------------------------------
16     public void yDisplay()
17     {
18        System.out.println(y);
19     }
20 }
```

```
22 class TestOOP1
23 {
24    public static void main(String[] args)
25    {
26       OOP1.xDisplay();
27
28       OOP1 n;
29       n = new OOP1(10);
30       OOP1 m = new OOP1(20);
31
32       n.yDisplay();
33       m.yDisplay();
34    }
35 }
```

Static members

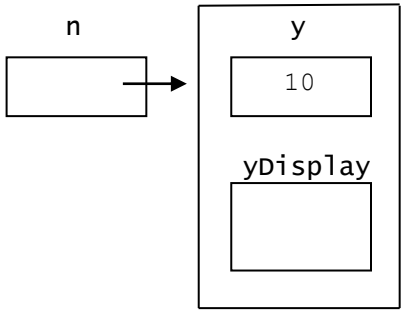


static variables and methods exist independently of objects

```
OOP1.xDisplay();
```

Instance variable and method within object

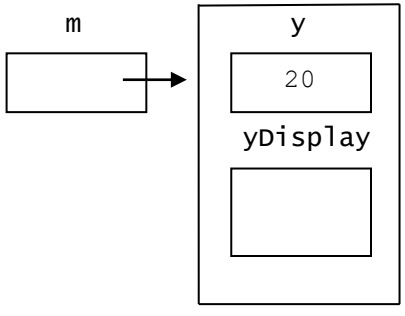
```
OOP1 n;                  // declare reference
n = new OOP1(10); // create object
```



instance variable and method contained in object

Do both in one statement

```
OOP1 m = new OOP1(20);
```



Constructor

1) The reserved word `public`
2) Its name, which must be the same as the class name
3) Its parameter list within parentheses

```
public void OOP1()
```

Summary

- Static variables and methods exist independently of objects.
- An external access to a static variable or an external call to a static method must be qualified with the class name.
- Nonstatic variables and methods are contained in objects.
- An external access of a nonstatic variable in an object or an external call of a nonstatic method in an object must be qualified with the reference to the object.
- External access is not permitted if the variable or method to be accessed is marked as private.
- A constructor always has the same name as its class.
- The header for a constructor does not include a return type.
- Static variables are also called class variables. Static method are also called class methods
- Nonstatic fields are also call instance variables. Nonstatic methods are also called instance methods.

Class with no constructor

```
 1 class OOP2   // No explicit constructor
 2 {
 3    private int q;
 4    //---------------------------------
 5    public void set(int qq)
 6    {
 7       q = qq;
 8    {
 9    //---------------------------------
10    public int get()
11    {
12       return q;
13    }
14 }
15 //================================================
16 class TestOOP2
17 {
18    public static void main(String[] args)
19    {
20       OOP2 r = new OOP2();
21       r.set(10);                        // sets q to 10
22       System.out.println(r.get());   // displays 10
23    }
24 }
```

Default Constructor

```
public OOP2()  // has no parameters
{
}


OOP2 r = new OOP2(5);   // illegal
```

this

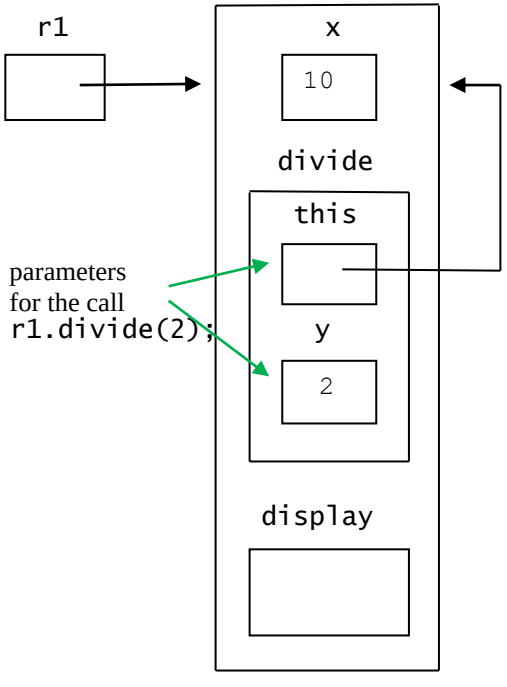
```
 1 class OOP3
 2 {
 3    private int x;
 4    //-----------------------------------
 5    public OOP3(int xx)
 6    {
 7       x = xx;
 8    }
 9    //-----------------------------------
10    public void divide(int y)
11    {
12       x = x/y;   // translated as this.x = this.x/y;
13       display(); // passes its this parameter
14    }
15    //-----------------------------------
16    private void display()
17    {
18       System.out.println(x); // translated as this.x
19    }
20 }
```

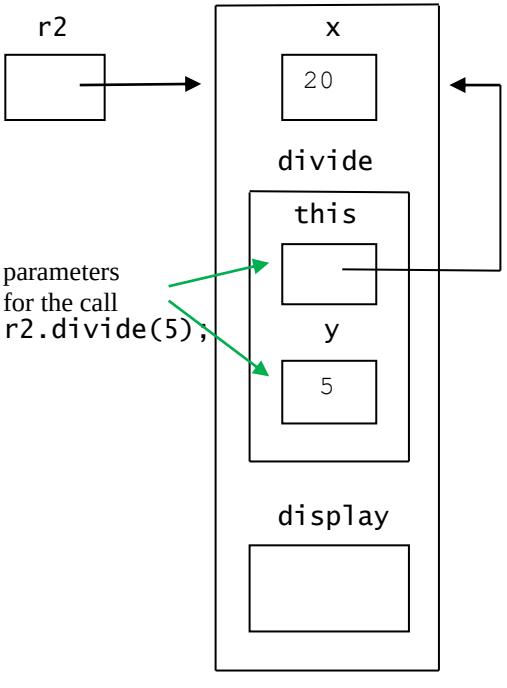
```
21 //=================================================
22 class TestOOP3
23 {
24    public static void main(String[] args)
25    {
26       OOP3 r1 = new OOP3(10);
27       OOP3 r2 = new OOP3(20);
28       r1.divide(2);            // r1 and 2 passed
29       r2.divide(5);            // r2 and 5 passed
30    }
31 }
```

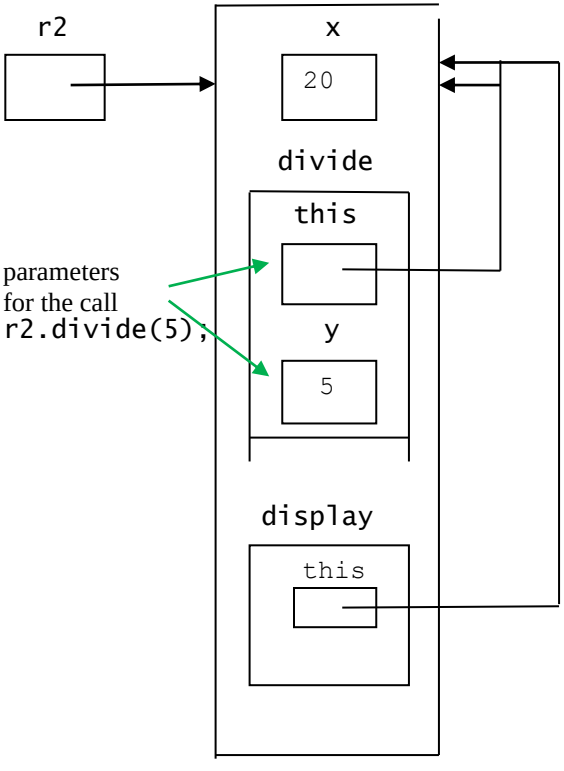
Reference passed in an instance method call

```
r1.divide(2);
```

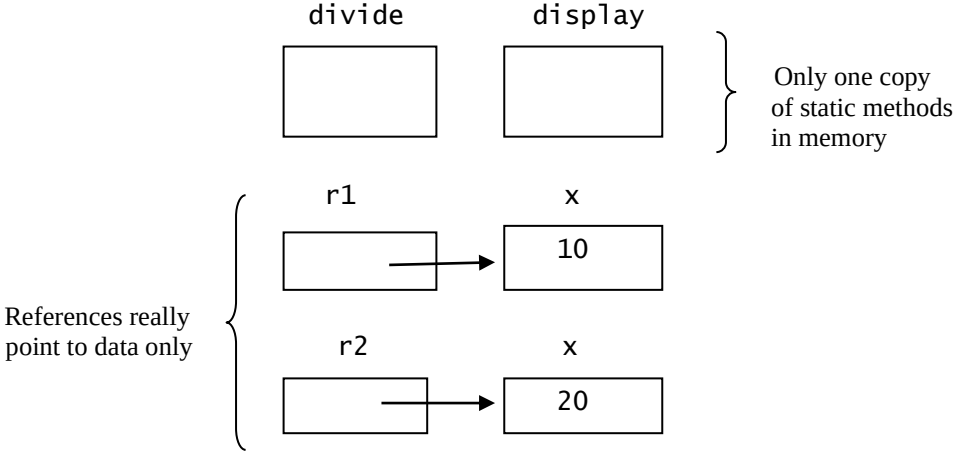
Value in `r1` and `2` are both passed to divide


r1
x
10
divide
this
parameters
for the call
r1.divide(2);
y
2
display


r2
x
20
divide
this
parameters
for the call
r2.divide(5);
y
5
display


r2
x
20
divide
this
parameters
for the call
r2.divide(5);
y
5
display
this

Conceptual View Vs Actual View



Summary

- An external call of an instance method must be qualified with a reference.
- An internal call of an instance method is translated as if it were qualified with `this`. Thus, every call of an instance method is qualified, either explicitly or implicitly.
- The qualifying reference in a call of an instance method is passed to the `this` parameter of the called method.
- Internal accesses of instance variables are translated as if they were qualified with `this`. Thus, they access the data the `this` parameter points to.

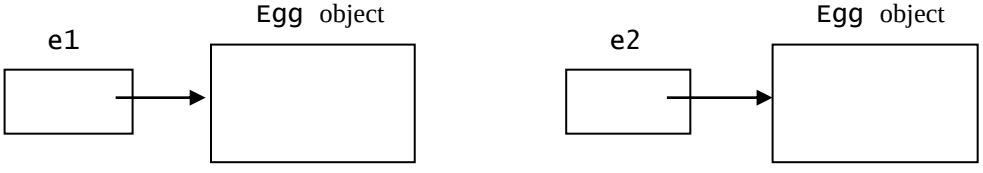
Case Study

```
 1 class Number
 2 {
 3    private int x;
 4    //-----------------------------------
 5    public Number(int xx)
 6    {
 7       x = xx;
 8    }
 9    //-----------------------------------
10    public Number add(Number r)
11    {
12       return new Number(x + r.x);
13    }
14    //-----------------------------------
15    public String toString()
16    {
17       return "" + x;  // return value in x as String
18    }
19 }
```

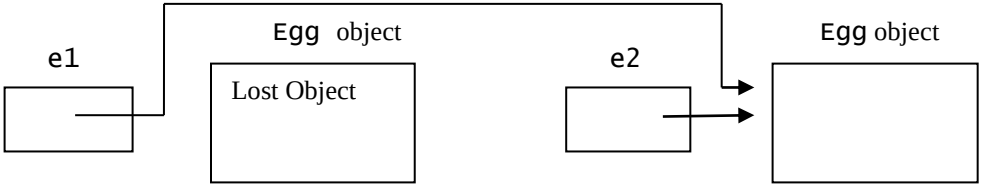
```
20 //================================================
21 class TestNumber
22 {
23    public static void main(String[] args)
24    {
25       Number n1 = new Number(10);
26       Number n2 = new Number(20);
27       Number n3 = n1.add(n2);
28       System.out.println("n3 = " + n3.toString());
29    }
30 }
```

Garbage Collection

```
Egg e1 = new Egg();
Egg e2 = new Egg();
```



```
e1 = e2;
```



Cannot access instance variables and methods from a static method

```
 1 class OOP4
 2 {
 3    private static int x = 1;     // static variable
 4    public int y = 2;             // instance variable
 5    //-----------------------------------
 6    public static void addOne()   // static method
 7    {
 8       x++;                       // legal
 9       y++;                       // illegal
10       display();                 // illegal
11    }
12    //-----------------------------------
13    public void display()         // instance method
14    {
15       System.out.println(x);     // legal
16       System.out.println(y);     // legal
17    }
18 }
```

```
19 //=================================================
20 class TestOOP4
21 {
22    public static void main(String[] args)
23    {
24       OOP4.addOne();
25       OOP4 p = new OOP4();
26       p.display();                // legal
27       System.out.println(p.y);  // legal
28    }
29 }
```

Named Constants versus Literal Constants

```
averageSalary = sumOfSalaries/100;

int employeeNumber = 1;
while (employeeNumber <= 100)
{
    .
    .
    .
    employeeNumber++;
}
```

Use variable to hold constant

Instead of 100, declare and initialize variable:

```
public static int numberOfEmployees = 100;
```

Then use variable in place of literal constant:

```
averageSalary = sumOfSalaries/numberOfEmployees;
```

Advantages

1) The variable name conveys a meaning that the literal constant does not. .

2) Using a variable makes it easy to modify the program if the constant should change.

3) If the constant requires many keystrokes, it is easier to enter the variable name than the constant. For example, entering `PI` is easier than entering `3.141592654`.

Better: Use Named Constant

```
public static final int NUMBEROFEMPLOYEES = 100;
```

Summary of Constants and Variables

1) Literal constants like `123`, `3.5`, `3.5f`, `'x'`, `"hello"`, `true`, and `false`.

2) Named constants. For example, the following statement creates the named constant `PI`:

```
public static final double PI = 3.141592654;
```

1) Local variables

```
class C1
{
    ...
    public void f()
    {
        int w;
        ...
    }
}
```

2) Parameters

```
class C2
 {
     ...
     public void g(int x)
     {
         ...
     }
 }
```

3) static variables (also called class variables)

```
class C3
{
    private static int y;
        ...
}
```

4) Nonstatic variables (also called instance variables)

```
class C4
{
   private int z;
      ...
}
```