Chapter 11

Inheritance Part 2

Overriding vs overloading

```
 1 class A
 2 {
 3    public void f()
 4    {
 5       System.out.println("f in A");
 6    }
 9    //-----------------------------------
 7    public void g()
 8    {
 9       System.out.println("g in A");
10    }
11 }
```

```
13 class B extends A
14 {
15    public void f()
16    {
17       System.out.println("f in B");
18    }
19    //-----------------------------------
20    public void g(int x)
21    {
22       System.out.println("g in B");
23    }
24 }
```

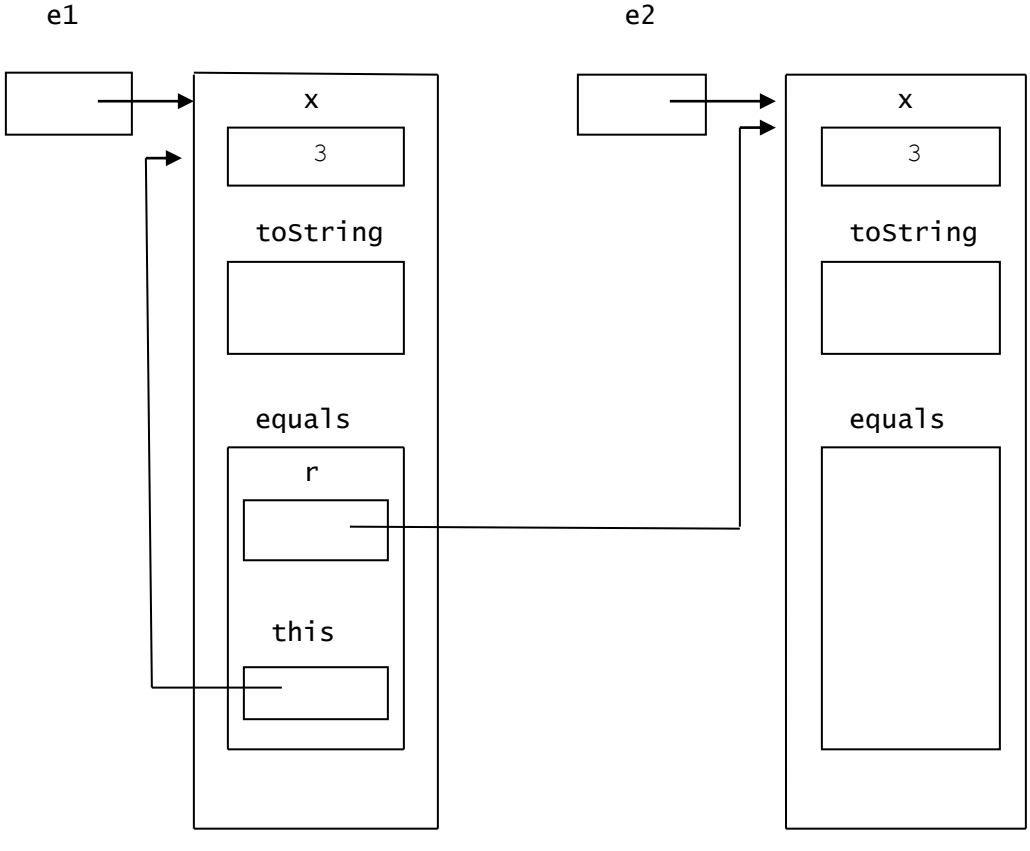
```
26 class OverrideOverload
27 {
28    public static void main(String[] args)
29    {
20       B b;
31       b = new B();
32       b.f();          // displays f in B
33       b.g();          // displays g in A
34       b.g(5);         // displays g in B
35    }
36 }
```

Calling a constructor with `this`

```
 1 class ThisExample
 2 {
 3    private int x, y, z;
 4    //-----------------------------------
 5    public ThisExample()
 6    {
 7       this(2, 3, 4);
 8    }
 9    //-----------------------------------
10    public ThisExample(int x, int y, int z)
11    {
12       this.x = x;
13       this.y = y;
14       this.z = z;
15    }
16 }
```

Writing an `equals` method

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


e1
e2
x
3
toString
equals
r
this
x
3
toString
equals

Problems with our equals method

```
Object obj = e2;
if (e1.equals(obj))  // invokes inherited equals
    ...


e1.equals(null)      // run-time error
```

A correct `equals` method

```
1 public boolean equals(Object r)
2 {
3    if (r == null) return false;
4    if (getClass() != r.getClass()) return false;
5    return x == ((EqualsExample)r).x;
6 }
```

Binding

```
 1 class Binding1
 2 {
 3    public static void f()
 4    {
 5       System.out.println("hello");
 6    }
 7    //-----------------------------------
 8    public static void main(String[] args)
 9    {
10       f();     // compile-time binding
11    }
12 }
```

```
 1 class A
 2 {
 3    public void f()
 4    {
 5       System.out.println("f in A");
 6    }
 7 }
 8 //================================================
 9 class B extends A
10 {
11    public void f()
12    {
13       System.out.println("f in B");
14    }
15 }
16 //================================================
17 class Binding2
18 {
19    public static void main(String[] args)
20    {
21       A a;
22       a = new A();
23       a.f();      // displays f in A
24
25       B b;
26       b  = new B();
27       a = b;     // a points "down" to B object
28       a.f();     // displays f in B
29    }
30 }
```

Why late binding is necessary

```
 1 A a1, a2;
 2 a1 = new A ();
 3 B b;
 4 b = new B();
 5
 6 for (int i = 0; i < 100; i++)
 7 {
 8       if (i % 2 == 0)
 9          a2 = a1;    // execute if i is even
10       else
11          a2 = b;     // execute if i is odd
12       a2.f();
13 }
```

Final methods and classes

```
 1 final class A
 2 {
 3    public void f()
 4    {
 5       System.out.println("hello");
 6    }
 7 }
 8 //================================================
 9 class B extends A // illegal--class A is final
10 {
11    ...
12 }
```

```
14 class C
15 {
16    public final void f()
17    {
18       System.out.println("hello");
19    }
20 }
21 //=================================================
22 class D extends C// legal--class C is not final
23 {
24    public void g()
25    {
26       System.out.println("bye");
27    }
28    public void f()// illegal--f is final in class C
29    {
30       System.out.println("bye");
31    }
32 }
```