Lesson 50: Object References and Copy Constructors (W16D4)

Balboa High School

Michael Ferraro

December 3, 2015

Do Now

Predict the output of the snippet below.

```
double a = -2.353;
double b = a;

a *= -1;

System.out.println("a = " + a);
System.out.println("b = " + b);
```

Aim

Students will learn about object references, objects having multiple references, garbage collection, and copy constructors.

Object References

Consider the statement below.

```
String str;
```

`str` is a **reference** to an object of type `String`; it's merely a *pointer* to a memory location that will contain a `String` object's data.

[1] One way to *access* data is to print it.

Object References

Consider the statement below.

```
String str;
```

`str` is a **reference** to an object of type `String`; it's merely a *pointer* to a memory location that will contain a `String` object's data.

→ What happens when you try to access[1] the data pointed to by `str` immediately after the declaration above?

- Create class `ObjectRef` in project `Lesson50`
- In `ObjectRef`'s `main()`:
  - declare `String str`
  - print `str` (≈2min)

[1] One way to *access* data is to print it.

Uninitialized Variables

Is the output of each version the same?

```
public class UninitObjA {

    static String str;

    public static void main(String[] args) {
        System.out.println(str);
    }
}
```

```
public class UninitObjB {

    public static void main(String[] args) {
        String str;
        System.out.println(str);
    }
}
```

Uninitialized Variables

Is the output of each version the same? **NO!**

```
public class UninitObjA {

    static String str;  //field, default value = NULL

    public static void main(String[] args) {
        System.out.println(str);
    }
}
```

```
public class UninitObjB {

    public static void main(String[] args) {
        String str;  //local var gets no default value
        System.out.println(str);
    }
}
```

Objects with 2+ References

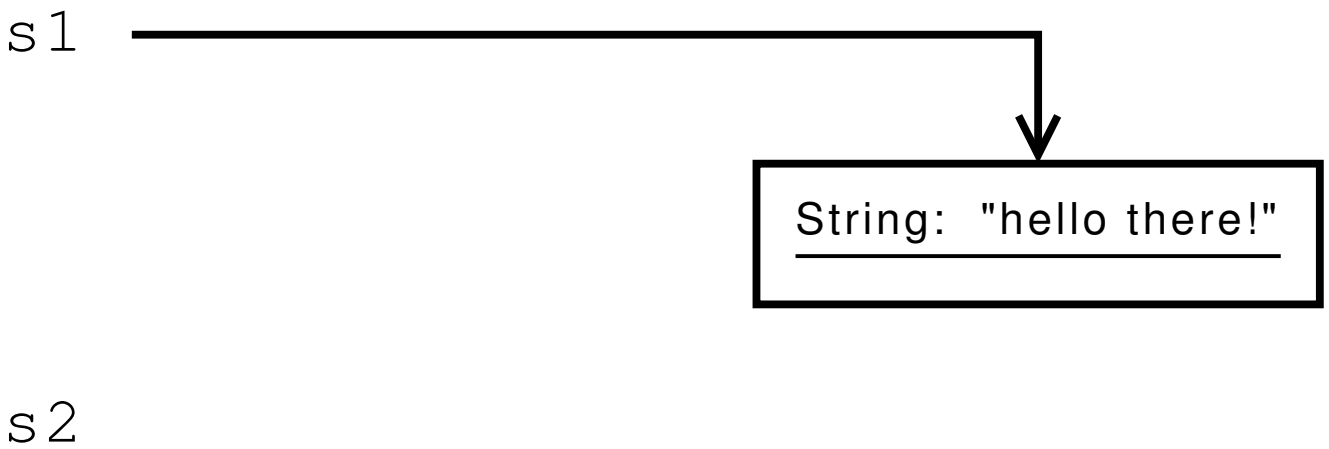
Create a `String` object having multiple references. Then, print the value of the `String` referred to by each reference.

```
public class MultiRef {
    public static void main(String[] args) {
        String s1 = "hello there!";
        String s2 = s1;

        System.out.println(s1 + "\n" + s2);
    }
}
```

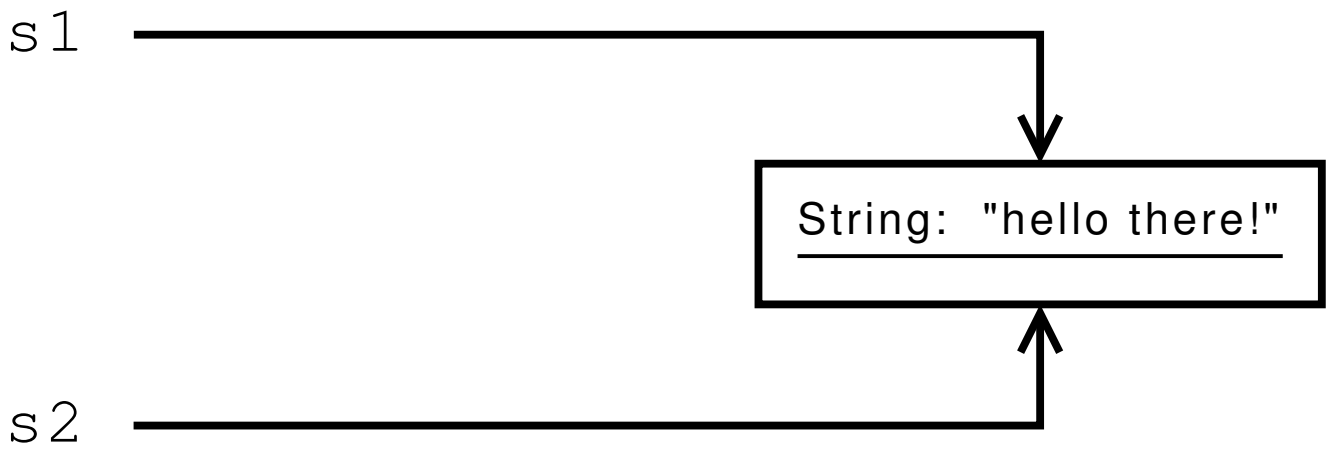
Objects with 2+ References

What just happened:



Objects with 2+ References

What just happened:



Objects with 2+ References

Now, modify the first reference, having it point to a different `String` object.

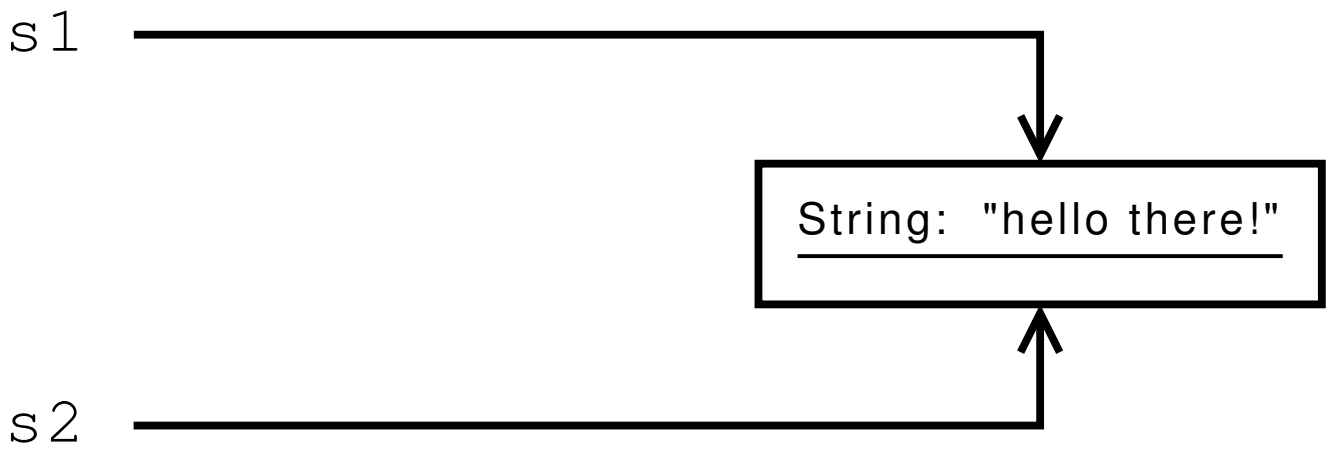
```
public class MultiRef {
    public static void main(String[] args) {
        String s1 = "hello there!";
        String s2 = s1;

        s1 = "I'm new here :)";

        System.out.println(s1 + "\n" + s2);
    }
}
```

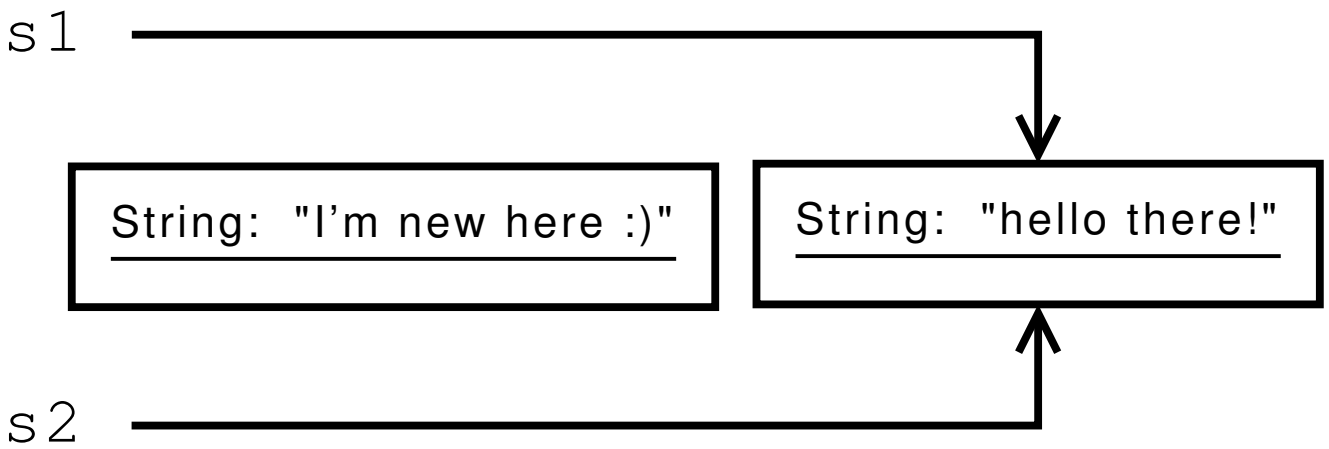
Objects with 2+ References

What just happened:



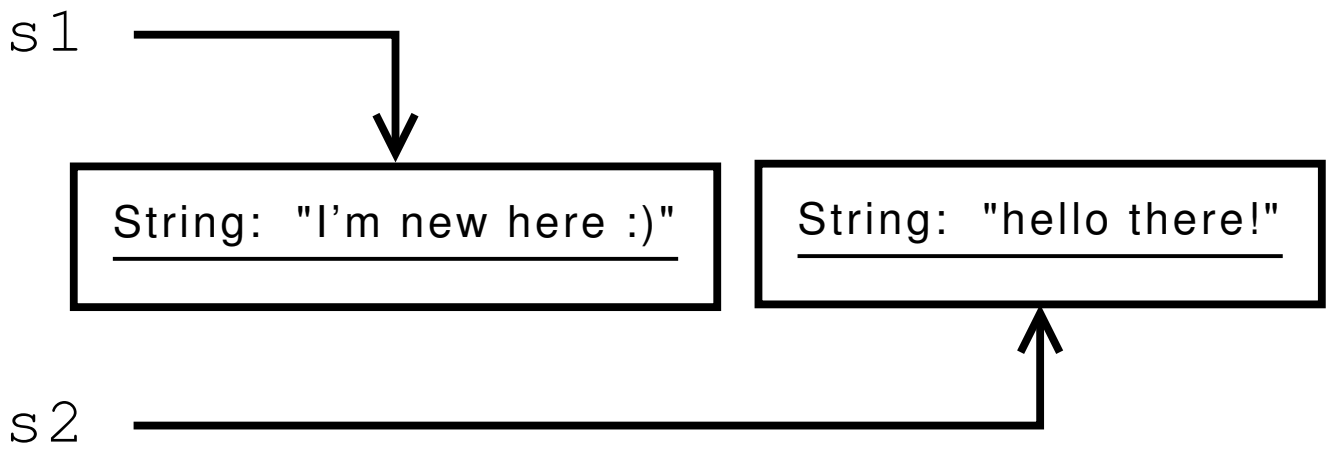
Objects with 2+ References

What just happened:



Objects with 2+ References

What just happened:



Objects with 2+ References

`Strings` are tricky in that their state can never truly be changed (more on that in the next chapter). Let's transition to `Fractions`.

1. Import `Fraction.java` from here.
2. Add this `main()`:

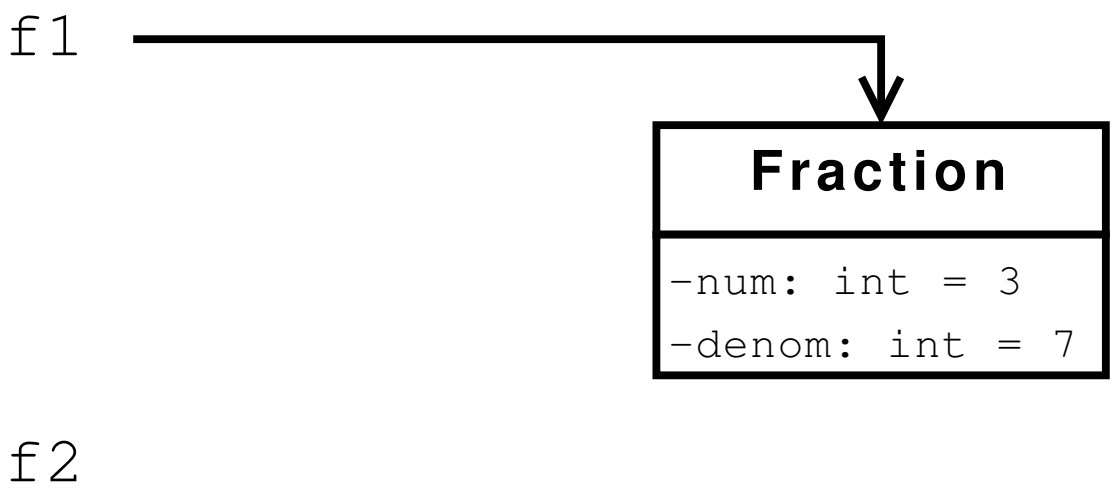
```
public static void main(String[] args) {
    Fraction f1 = new Fraction(3, 7);
    Fraction f2 = f1;
    f1.denom = 6;

    System.out.println(f1 + "\n" + f2);
}
```

3. What was printed? Why?

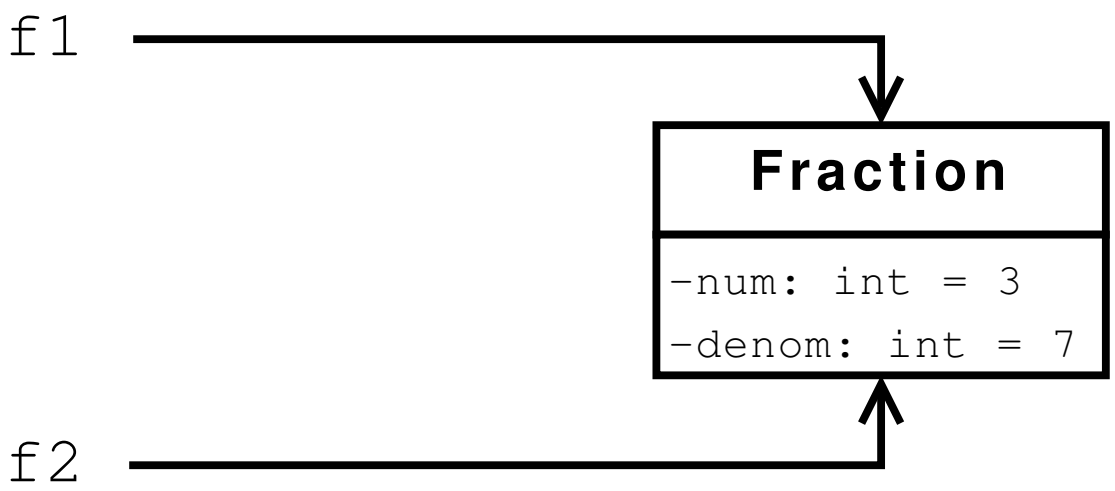
Objects with 2+ References

What just happened:



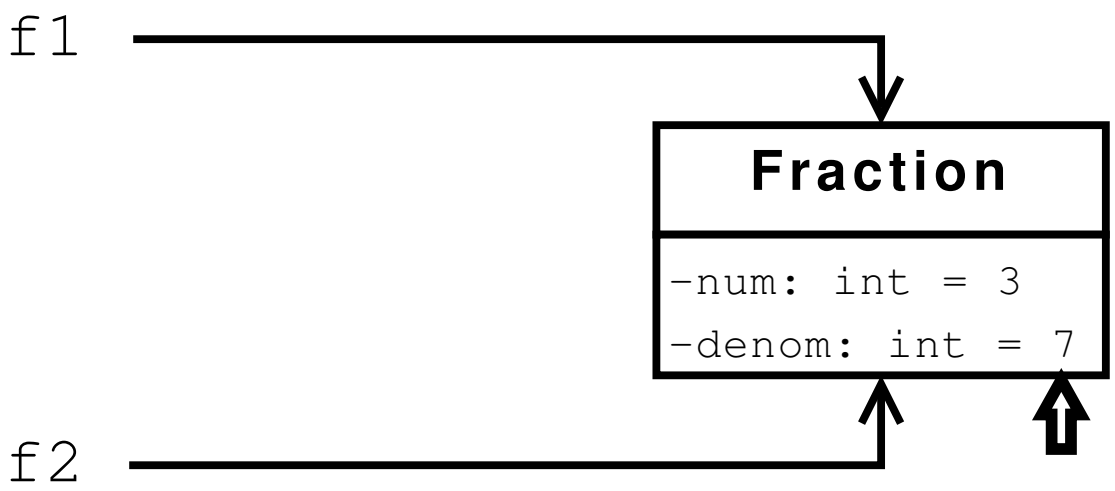
Objects with 2+ References

What just happened:



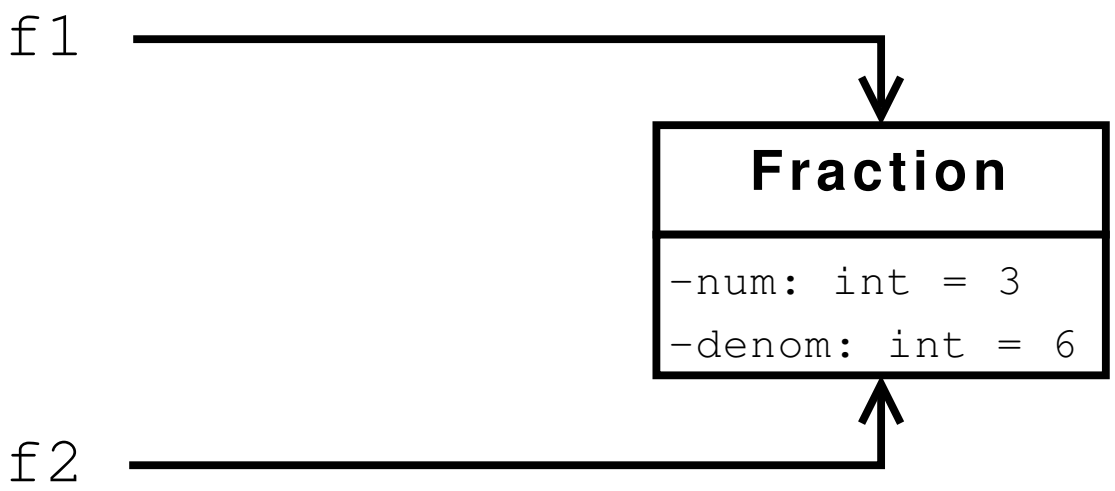
Objects with 2+ References

What just happened:



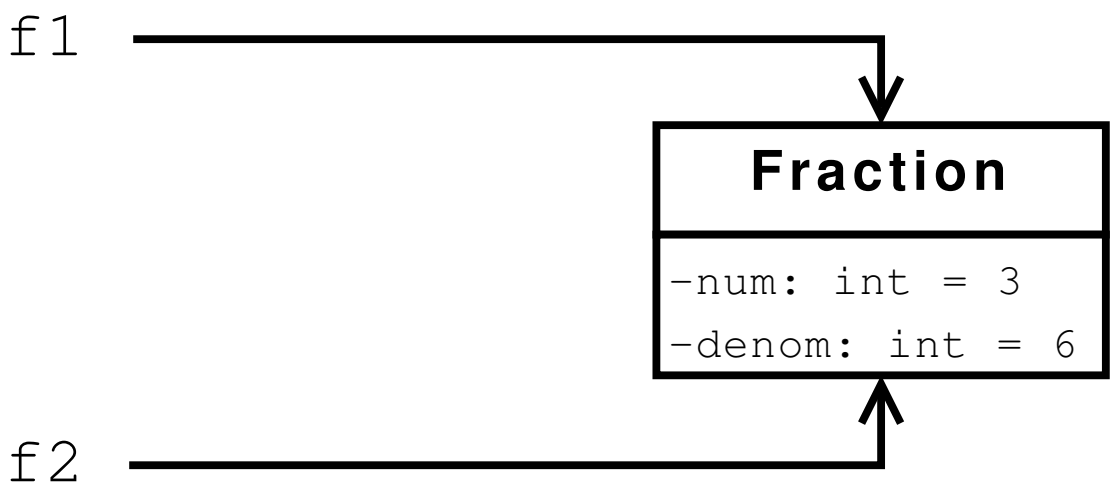
Objects with 2+ References

What just happened:



Objects with 2+ References

What just happened:



→ *What would happen if you called* `f2.reduce()`*?*

Copy Constructors

When you *don't* want 2+ vars to point to the **same** object in memory, use a *copy constructor* to make an indentical copy of the first object in memory.

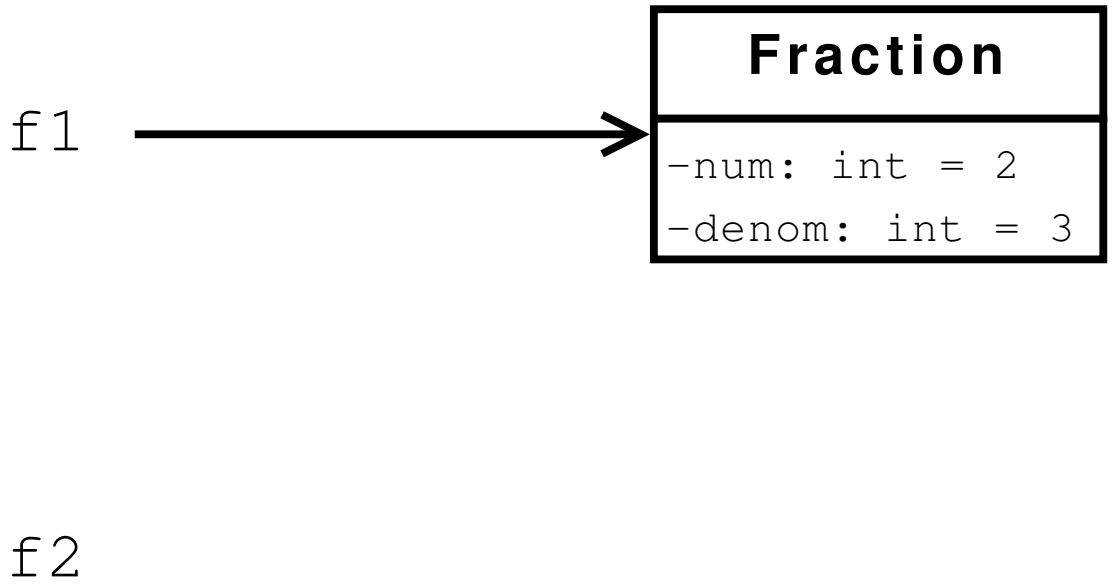
Try this:

```
Fraction f1 = new Fraction(2, 3);
Fraction f2 = new Fraction(f1);
f1.num = 1;

System.out.println(f1 + "\n" + f2);
```

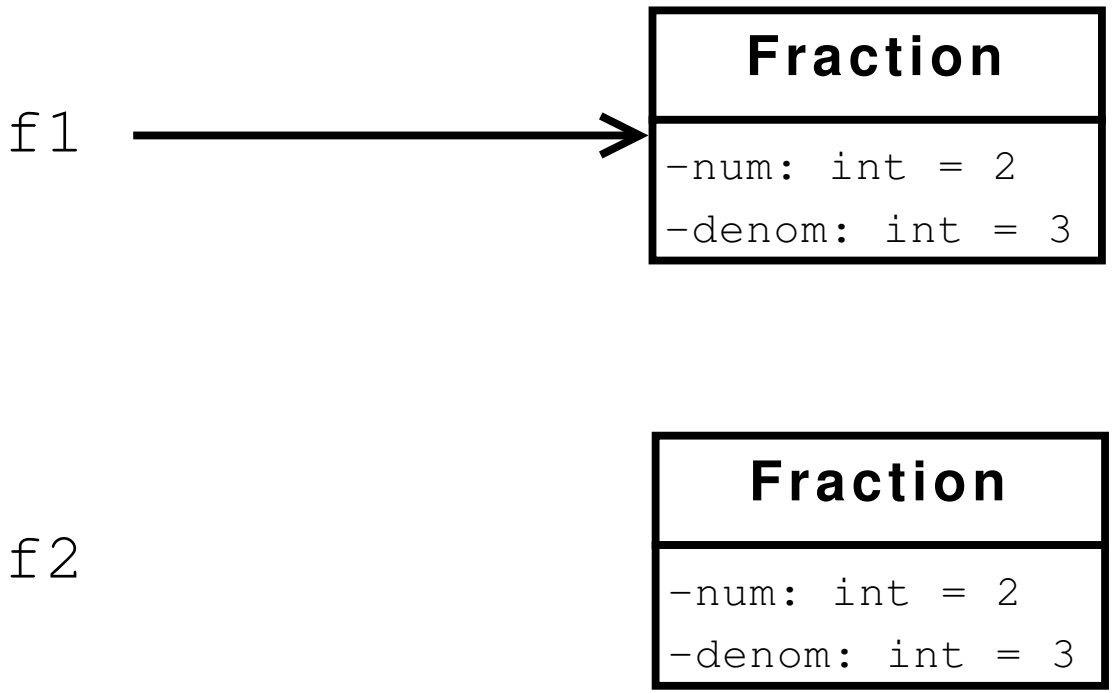
Copy Constructors

What just happened:



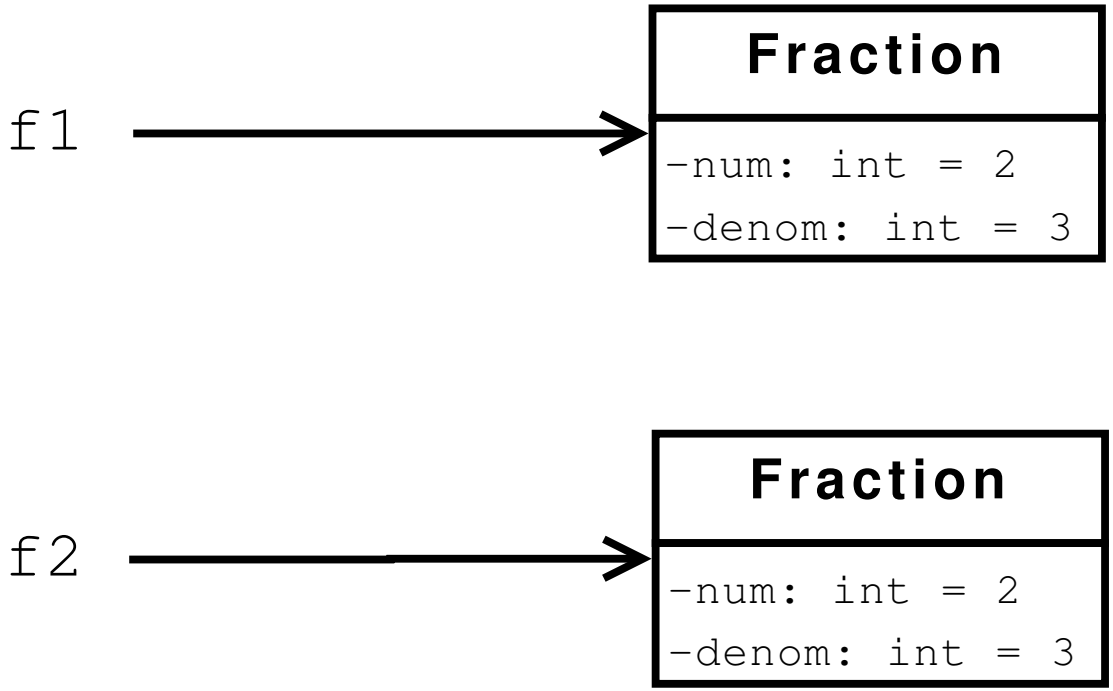
Copy Constructors

What just happened:



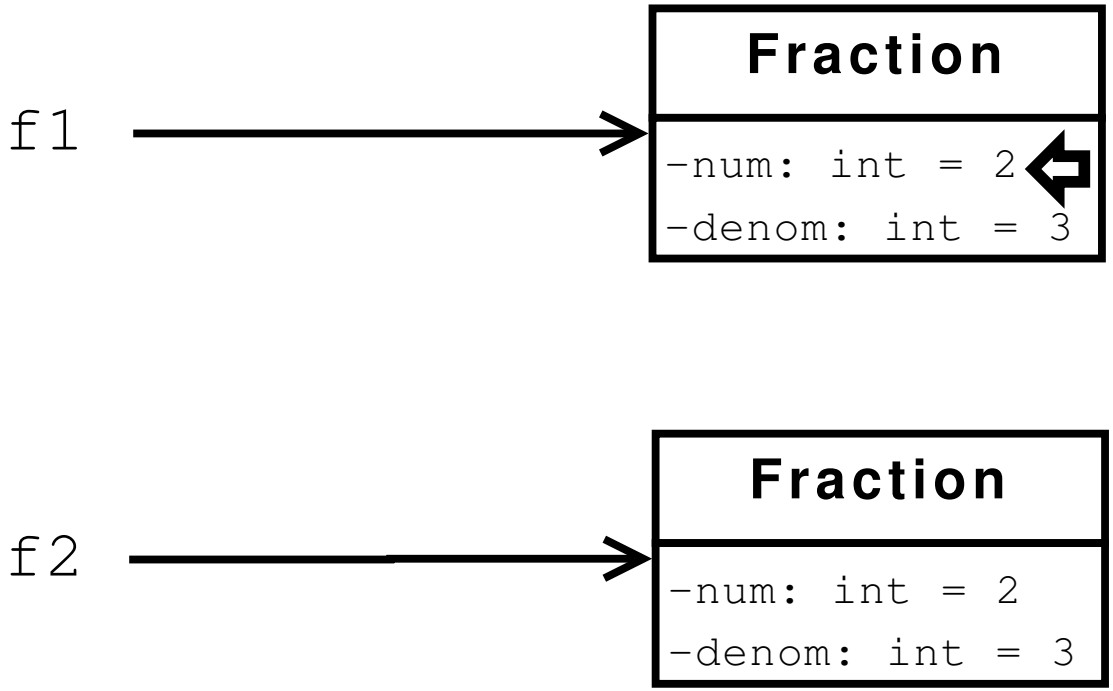
Copy Constructors

What just happened:



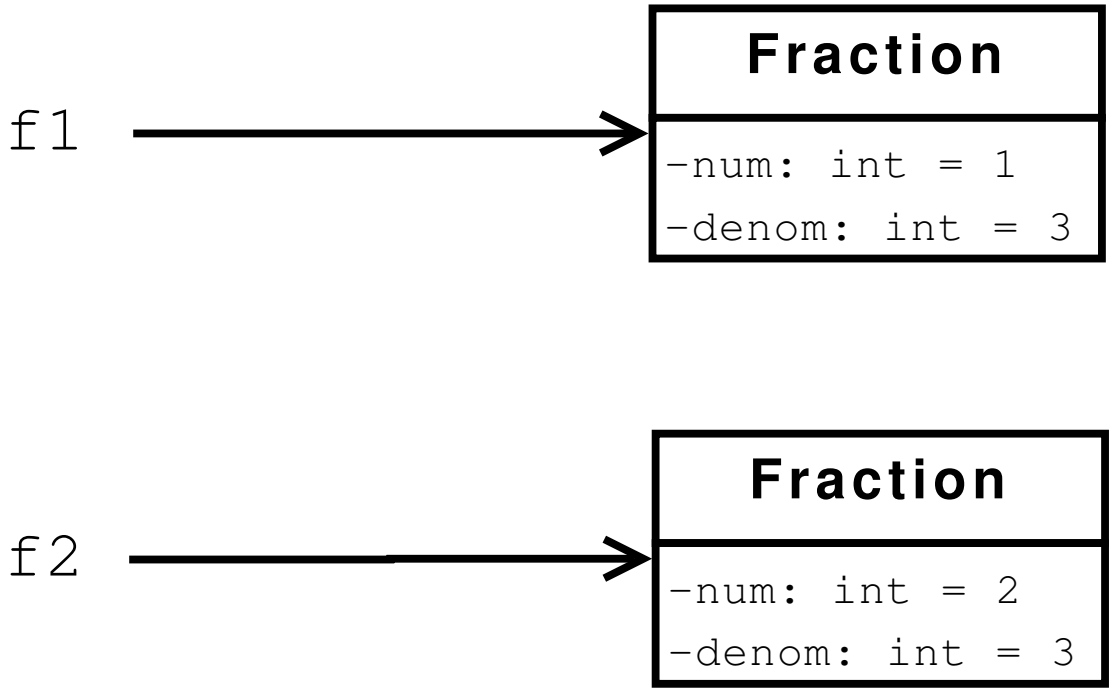
Copy Constructors

What just happened:



Copy Constructors

What just happened:



Dereferencing Objects

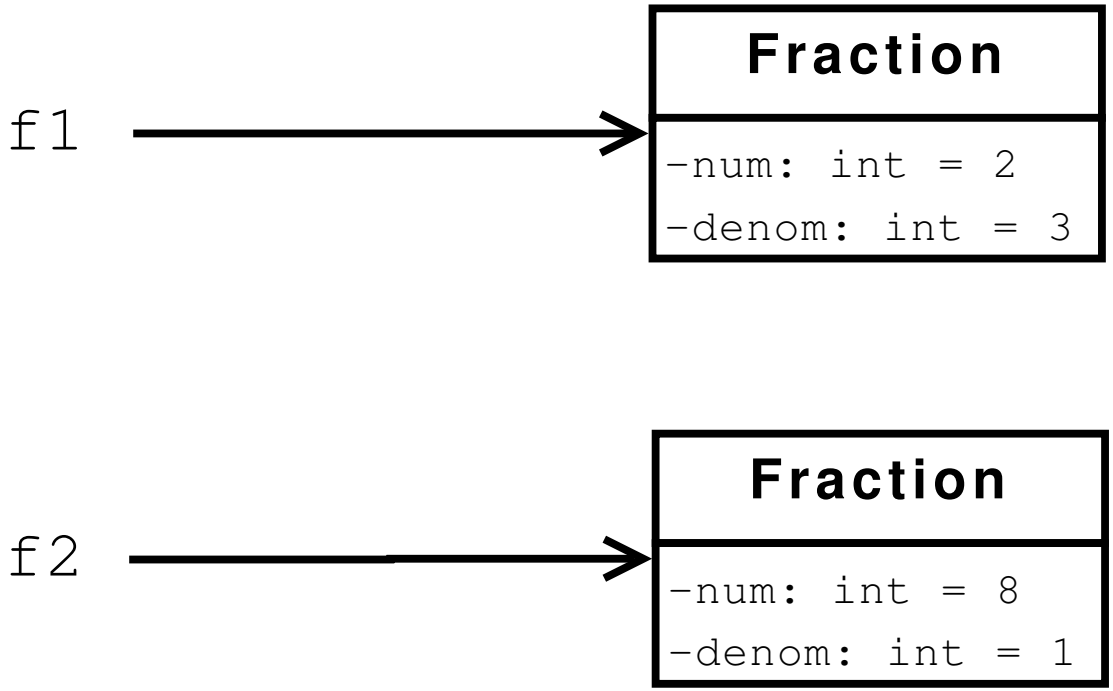
Let's see what happens when an object no longer has any references to it:

```
Fraction f1 = new Fraction(2, 3);
Fraction f2 = new Fraction(8);
f1 = f2;
```

→ *What do you think becomes of the* `Fraction` *whose value is* $\frac{2}{3}$*?*

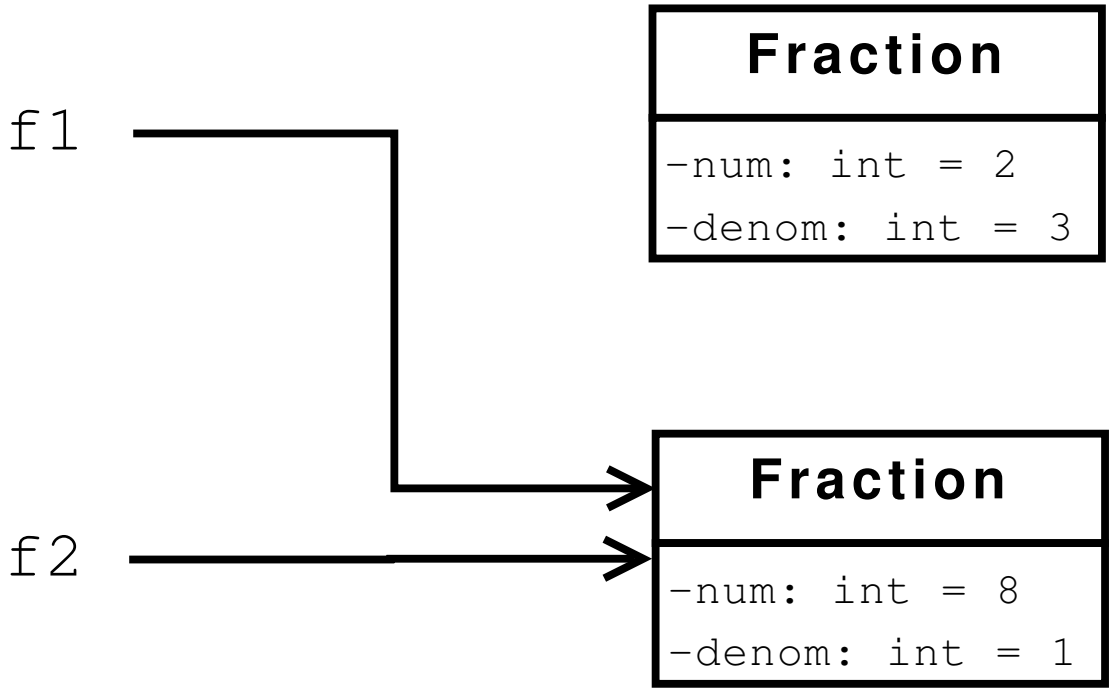
Dereferencing Objects

What just happened:



Dereferencing Objects

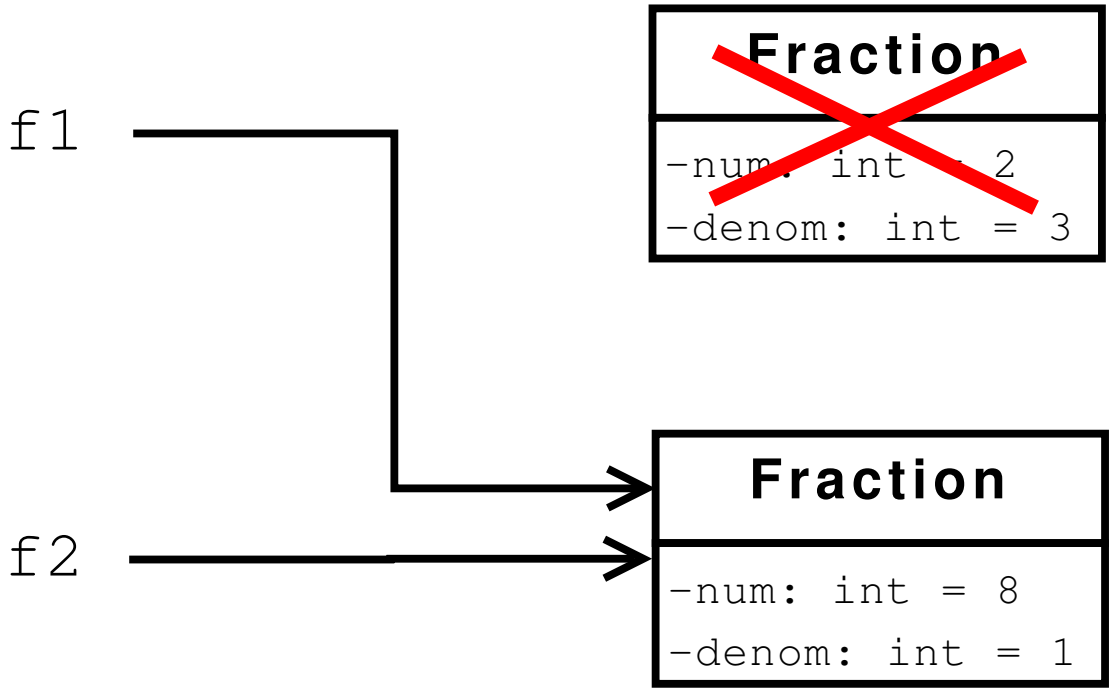
What just happened:



First `Fraction` is now **dereferenced**, meaning it no longer has any references.

Dereferencing Objects

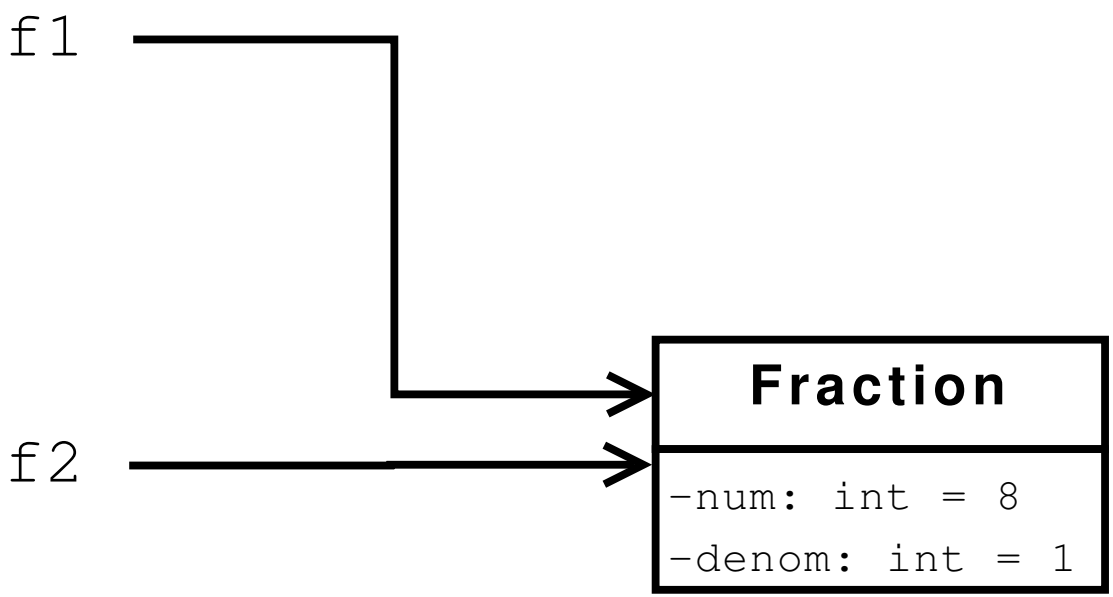
What just happened:



It is now subject to **garbage collection** by the Java virtual machine (JVM). In other words, the memory space that `Fraction` occupies will be freed and reclaimed for later use.

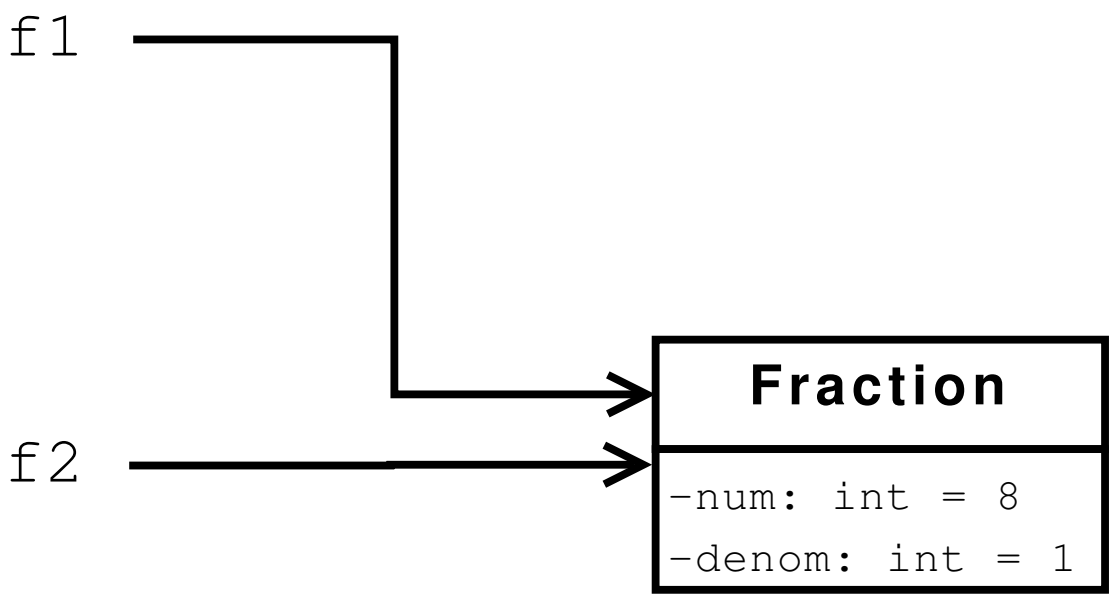
Dereferencing Objects

What just happened:



Dereferencing Objects

What just happened:



→ *What do you think the term "memory leak" means in the context of an application?*

Next. . .

- Work on PS #9, §3.3 — Copy Constructors and `private` Fields
- Do you still need a sign-off for §2.2?

HW

- PS #9, §4: Constructors
- PS #9, §5: Object References