Classes and Objects in Java – Creating Objects

In mathematics, a fraction can be defined as the ratio between two, non-zero integers. We typically express a fraction in the form $\frac{a}{b}$, and there are a number of rules that describe how to perform operations such as addition and multiplication on fractions.

Examples of fractions are expressions such as $\frac{3}{4}$, $\frac{7}{-3}$, or $\frac{-2}{5}$.

The definition of a fraction would be an example of a *class* in Java – it is the definition, or *blueprint*, for all such expressions without specifying actual values for *a* and *b*.

The examples of fractions listed above would be *instances* of the fraction class. These instances form the objects that we use and manipulate in our programs.

Creating a Fraction Class

To clarify, we will start by creating a Fraction class in Java. A fraction is made up of two parts, a numerator and denominator. Both the numerator and denominator are integer values.

```
class Fraction
{
      int num;   // numerator
      int den;   // denominator
}
```

You may notice that this class is very different that those we have previously studied in Java. There is no `main` method. In fact, there are no methods at all. This class doesn't do anything, It simply defines something called a `Fraction`, which has an integer numerator and an integer denominator.

These two parts, with the labels `num` and `den`, are the *fields* of the class. Fields are similar to *local variables*, except they are not declared inside any method. This class declaration is very similar to the concept of *data type*, which you may recognize from previous experience with other programming languages.

Creating a Fraction Object

The class definition does not, in itself, create any fractions. It simply shows us what a fraction will look like once we create one.

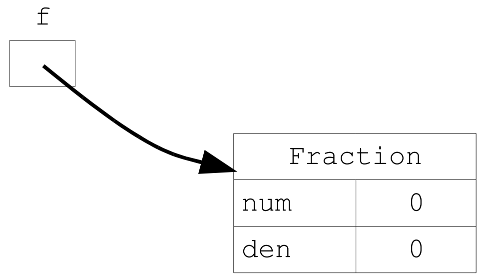
Our first attempt to create a fraction might be a simple declaration statement, similar to those used for `char` or `int`.

```
Fraction f;           // declare a Fraction variable
```

This statement does create a variable `f` that can act as a *reference* to a fraction (i.e., it can *point to* a fraction), but there is still no *instance* of a fraction yet. To achieve this, we now add the line:

```
f = new Fraction();   // instantiate a Fraction object
```

This second line causes Java to create an *object*, an instance of the `Fraction` class, somewhere in the computer's memory. In addition, the variable f is set to act as a *reference* to that memory location. We often illustrate references to memory locations using a diagram:



Notice that the `num` and `den` fields are both shown with the value zero. When variables are declared, Java does not initialize them. When we create, or *instantiate*, an object, all numeric fields are initialized to zero.

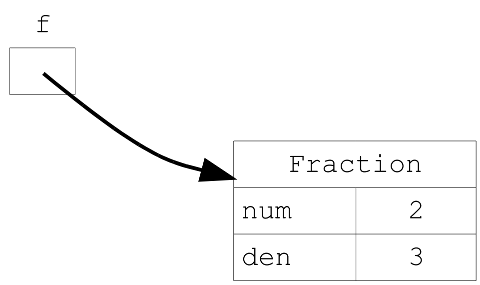
Declaring and creating an instance of a class is commonly combined into a single line:

Fraction f = new Fraction();

Assigning Values to Object Fields

If we wanted to modify our object to represent $\frac{2}{3}$ instead of $\frac{0}{0}$, we could now make the following assignment statements:

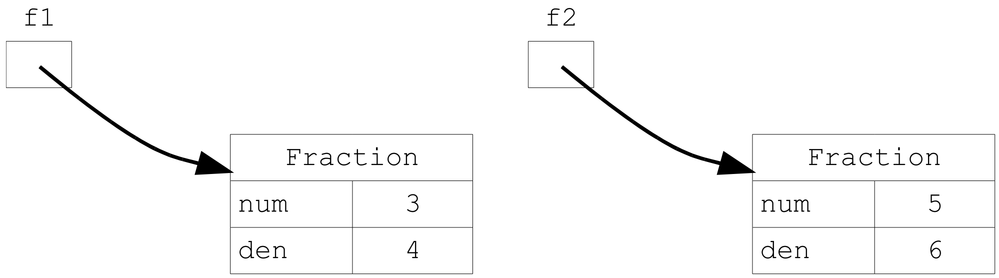
```
f.num = 2;      // set the numerator of f to 2
f.den = 3;      // set the denominator of f to 3
```



To create two fractions, f1 and f2, representing the fractions $\frac{3}{4}$ and $\frac{5}{6}$, we could write:

```
Fraction f1 = new Fraction();
f1.num = 3;
f1.den = 4;
Fraction f2 = new Fraction();
f2.num = 5;
f2.den = 6;
```

Pictorially, the result would be:



Using these objects, we can now manipulate their fields in the same ways we do with other variables.

```
f1.num--;
f1.den = f2.den + 3;
```

would change f1 to represent the fraction $\frac{3-1}{6+3}=\frac{2}{9}$, while f2 would be unchanged.

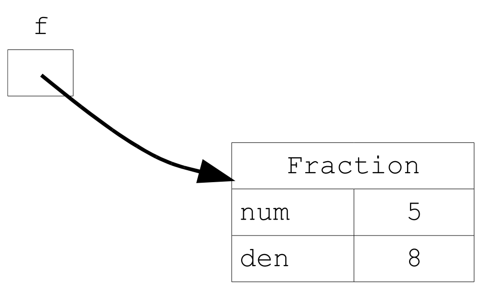
Multiple References to the Same Object

Objects are very different from our primitive data types (e.g., integers). With primitive data types, the variable refers directly to a *value*, and all operations are performed on the value. With objects (including Strings), the variable is a reference to a *memory location*.

For example, suppose we start with

```
Fraction f = new Fraction();
f.num = 5;
f.den = 8;
```

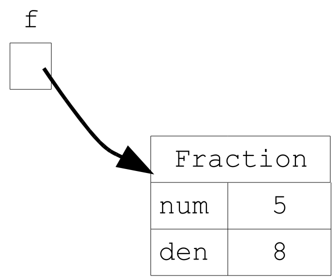
This produces the object shown below.



If we add the following:

```
Fraction g = f;
```

the value of `g` is set to be equal to the value of `f`. But in the case of objects, the value of the variable is actually a reference to a memory location. As a result, the variable `g` now points to the same memory location as `f`, as illustrated in the following diagram.



If we were to write

```
g.num = 1;
```

this would change the `num` field of the object shown. Since `f` also refers to the same object, both `f` and `g` would now represent the fraction $\frac{1}{8}$.

Questions & Exercises

1. What is the essential difference between a local variable and a field?

2. Write statements that could be used to create an object of type `Fraction` representing the fraction $\frac{2}{7}$.

3. Assuming the class Fraction is defined as it was in the lesson, state the error in the following fragment.

```
Fraction p;
p.num = 7;
p.den = 8;
```

4. Draw diagrams like those shown in the lesson to illustrate the results of the given fragment.

```
Fraction p, q, r;
p = new Fraction();
q = new Fraction();
r = q;
p.num = p.den = 2;
q.num = 2*p.den;
p.den++;
p.num--;
r.den = p.num + 2;
```

5. Assuming that two objects f1 and f2 of type Fraction have been created and assigned values, write statements to perform each task.
   a) Double the value of f1.
   b) Invert f2.
   c) Make f1 equal to the (unsimplified) product of f1 and f2.
   d) Make f2 equal to the (unsimplified) sum of f1 and f2.
   e) Make f1 equal to $|f1|$ (the absolute value of f1).