# Classes and Objects

Problem: Perform operations on Fraction

**What are classes and objects?**

- A class is a model, blueprint, or template of what an object should look like. The definition of a class does not actually create any object.
- An object is an instance of the class. When it is created, it has the same properties and rules defined in the class.
- Here is an example:

A fraction is an expression of the form $\frac{a}{b}$ together with rules on how operations such as addition and multiplication are perform – This is a **class**

Examples of fractions are $\frac{3}{4}$, $\frac{7}{-3}$ or $\frac{-2}{5}$ – These are **objects** (particular instances of the class)

**Creating the class**

To begin implementing fraction using Java, the operation will first be ignored. Consider a fraction is composed of two parts: a numerator and a denominator.

```
class Fraction {
     int num;
     int den;
}
```

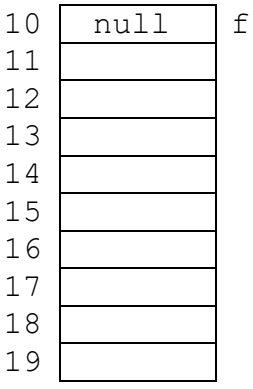
- This class does not contain a `main` method (no method at all).
- This class called `Fraction` contains two `int` parts called `num` and `den`.
- These two `int` parts are called *fields* of the class.
- A ***field*** is defined outside a method while a local variable is defined inside a method

**Creating the object**

- To create an actual fraction, using the template (class), first, make the declaration:

```
Fraction f;
```

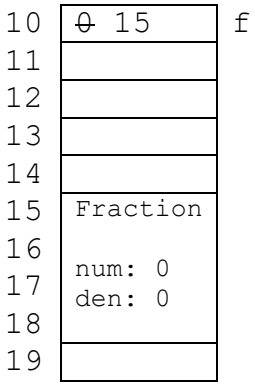
This creates a variable f that is a reference to a fraction. It does not actually refer to a fraction until you create an object and assign to the variable



- To create a new object, the `new` statement is used:

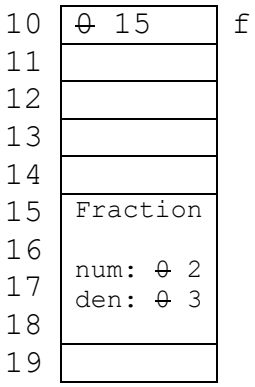
```
f = new Fraction();
```

When an instance of `Fraction` is created, it has its own `num` and `den` fields, and they are the ***instance fields***.




- Once the object is created, its instance fields can be modified. For example, to represent the fraction $\frac{2}{3}$, the following assignment statements can be made:

```
f.num = 2;
f.den = 3;
```




**Creating instance methods**

- Functionalities associated with individual object are implemented through instance methods (non-static).
- For example, the following instance method will return the magnitude of a `Fraction` object:

```
public double size() {
     return Math.abs((double)num/den);
}
```

- The above is only a declaration of the method. It needs to be called (e.g. from the main method) to be executed. For example,

```
double s = f.size();
```

Several things to note about the above method

- The method does not have the modifier `static`, thus is an instance method.
- The general form of call to an instance method that returns a value is <object identifier>.<method identifier>(<parameter list>);
- Within the method `size`, the identifier `num` and `den` are used, not `f.num` and `f.den`. No object is mentioned explicitly. Instead, it is implicit that `num` and `den` are those of the object currently associated with the method. For example, when the statement

```
t = g.size();
```

is called, g would be the implicit parameter and java would use the num and den fields of g in its calculations.

- The current implicit object in an instance method can be referred to in Java as `this`.

Consider the following method with a non-empty parameter list:

```
public Fraction larger (Fraction other) {
     if (this.size() >= other.size()) {
          return this;
     } else {
          return other;
     }
}
```

- Assume `f`, `g`, and `h` are all of type Fraction, and if the statement

  ```
  h = f.larger(g);
  ```

  is called, the implicit object (`this`) would refer to f and the explicit parameter, other, would prefer to `g`.