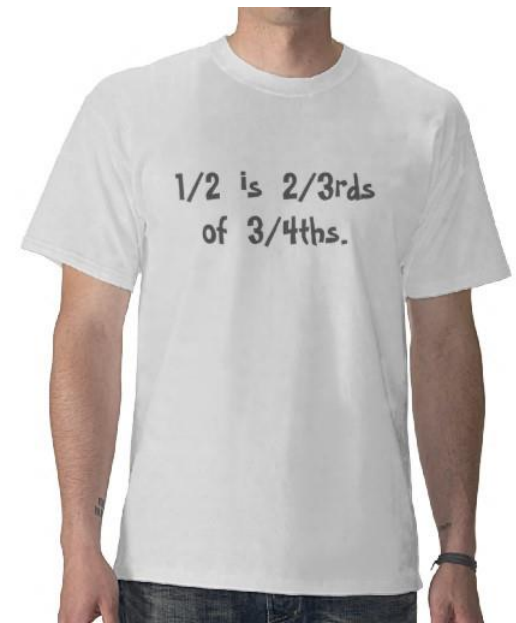# Building a fraction class

I LOVE
FRACTIONS

5 out of 4
people can't
do fractions

1/2 is 2/3rds
of 3/4ths.

http://www.zazzle.com/fraction+tshirts
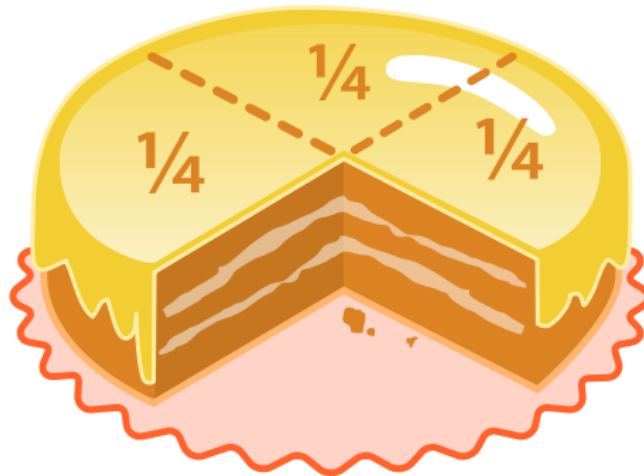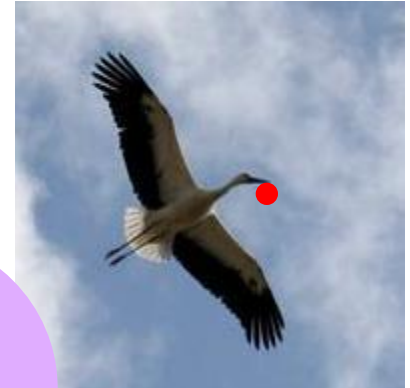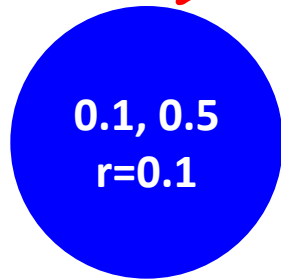
Overview

- Object oriented techniques
  - Constructors
  - Methods that take another object of same type
  - Private helper methods
- Fraction class
  - Create a class to represent fractions

Hey objects, where did you come from?

"Dude, don't you know where objects come from?!? The object stork totally dropped us off."

0.1, 0.5
r=0.1

0.5, 0.3
r=0.2

0.9, 0.6
r=0.15

```
public Ball(double x, double y, double r)
{
    posX   = x;
    posY   = y;
    radius = r;
}
```

Constructor = the object stork

Automatic default constructors

- Rule 1: If you do not create a constructor one will be automatically created for you
  - Default no-arg constructor
    - Doesn't do anything, no code runs
    - All instance variable are whatever you initialized them to (or their default value)

```java
public class Fraction
{
    private int num;        // numerator (upstairs)
    private int denom;      // denominator (downstairs)
}
```

Creating with default constructor

- To create object using no-arg constructor
  - Use empty ()'s after the new
  - Parameter list always sent when new'ing object
    - Java needs to know which constructor to run

```java
public class FractionClient
{
    public static void main(String [] args)
    {
        Fraction a = new Fraction();

        Fraction [] fracs = new Fraction[2];
        fracs[0] = new Fraction();
        fracs[1] = new Fraction();
    }
}
```
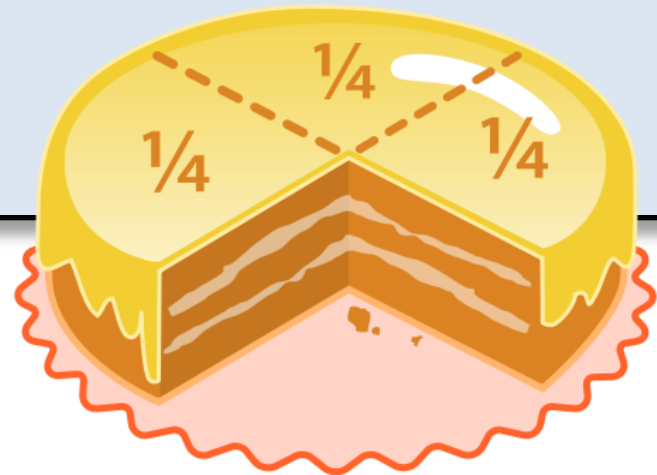
Declaring your own constructor

- Rule 2: If you declare any constructor, a default one will not be automatically created

```java
public class Fraction
{
    private int num;        // numerator (upstairs)
    private int denom;      // denominator (downstairs)

    public Fraction(int n, int d)
    {
        num   = n;
        denom = d;
    }
}
```
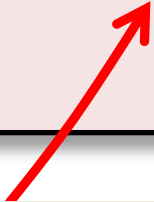
Creating with default constructor

```java
public class FractionClient
{

   public static void main(String [] args)
   {

      Fraction a = new Fraction();

      Fraction [] fracs = new Fraction[2];
      fracs[0] = new Fraction();
      fracs[1] = new Fraction();
   }
}
```

We broke all the calls to create a Fraction object since there no longer exists a no-arg version of the constructor.

```
% javac FractionClient.java
FractionClient.java:5: error: constructor
Fraction in class Fraction cannot be
applied to given types;
        Fraction a = new Fraction();
                         ^
  required: int,int
  found: no arguments
  reason: actual and formal argument lists
differ in length
```

Constructor overloading

- Rule 3: You can declare as many constructor versions as you need/want

Hooray! Now our code using a no-arg constructor will work again.

```java
public class Fraction
{
    private int num;        // numerator (upstairs)
    private int denom;      // denominator (downstairs)

    public Fraction()
    {
    }

    public Fraction(int n, int d)
    {
        num   = n;
        denom = d;
    }
}
```

Parameters of your own type

- Create a new object based on another instance of the same type
  - A copy constructor

```java
public class Fraction
{
    private int num;        // numerator (upstairs)
    private int denom;      // denominator (downstairs)

    // Create a new Fraction object that has the same
    // values as some other fraction object.
    public Fraction(Fraction other)
    {
        num   = other.num;
        denom = other.denom;
    }
}
```

You can access private instance variables of another object of the same type inside a method of that type.

Multiplying fractions

- Goal: Given two fraction objects, return a new fraction that is the multiplication of the two

```java
public class FractionClient
{
    public static void main(String [] args)
    {
        Fraction a = new Fraction(1, 2);
        Fraction b = new Fraction(2, 3);

        Fraction c = a.multiply(b);

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

```
% java FractionClient
1/2 * 2/3 = 1/3
```

Multiply method

```java
public class Fraction
{
    private int num;        // numerator (upstairs)
    private int denom;      // denominator (downstairs)

    ...

    public Fraction multiply(Fraction other)
    {
        Fraction result = new Fraction(num    * other.num,
                                        denom * other.denom);
        return result;
    }
}
```

Denominator of the object that called multiply (before the dot).

Denominator of the object passed as a parameter to the multiply() method

```java
Fraction c = a.multiply(b);
```

Multiplying fractions

- Attempt 1: Hmmmm, we forgot something…

```java
public class FractionClient
{
   public static void main(String [] args)
   {
      Fraction a = new Fraction(1, 2);
      Fraction b = new Fraction(2, 3);

      Fraction c = a.multiply(b);

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

```
% java FractionClient
Fraction@164f1d0d * Fraction@23fc4bec = Fraction@8dc8569
```

Multiplying fractions

- Attempt 2: Close, but not in lowest terms…

```java
public class Fraction
{
    private int num;        // numerator (upstairs)
    private int denom;      // denominator (downstairs)

    ...

    public String toString()
    {
        return "" + num + "/" + denom;
    }
}
```

An object's `toString()` method is called automatically whenever you attempt to print an object

```
% java FractionClient
1/2 * 2/3 = 2/6
```

Lowest terms

- Attempt 3: Add code to reduce to lowest terms

```java
public class Fraction
{
    ...
    public Fraction multiply(Fraction other)
    {
        Fraction result = new Fraction(num   * other.num,
                                       denom * other.denom);
        int i = Math.min(Math.abs(result.num),
                         Math.abs(result.denom));
        if (i == 0)
            return result;
        while ((result.num % i != 0) || (result.denom % i != 0))
            i--;
        Fraction result2 = new Fraction(result.num   / i,
                                        result.denom / i);

        return result2;
    }
}
```

```
% java FractionClient
1/2 * 2/3 = 1/3
```

Divide method

- Very similar method for division:

```java
public class Fraction
{
    ...
    public Fraction divide(Fraction other)
    {
        Fraction result = new Fraction(num    * other.denom,
                                       denom * other.num);
        int i = Math.min(Math.abs(result.num),
                         Math.abs(result.denom));
        if (i == 0)
            return result;
        while ((result.num % i != 0) || (result.denom % i != 0))
            i--;
        Fraction result2 = new Fraction(result.num   / i,
                                        result.denom / i);
        return result2;
    }
}
```

Repeated code is evil™

```java
public Fraction multiply(Fraction other)
{
    Fraction result = new Fraction(num    * other.num,
                                    denom * other.denom);
    int i = Math.min(Math.abs(result.num),
                     Math.abs(result.denom));
    if (i == 0)
        return result;
    while ((result.num % i != 0) || (result.denom % i != 0))
        i--;
    Fraction result2 = new Fraction(result.num    / i,
                                     result.denom / i);
    return result2;
}
```

Where should this code really live? There are a number of choices, but not here for sure.

We'd have to repeat it in the `divide()`, `add()`, and `subtract()` methods as well.

Helper methods

- Add a private helper method, reduce()

```java
public class Fraction
{
    private void reduce()
    {
        int i = Math.min(Math.abs(num), Math.abs(denom));
        if (i == 0)
            return;
        while ((num % i != 0) || (denom % i != 0))
            i--;
        num   = num   / i;
        denom = denom / i;
    }
    public Fraction multiply(Fraction other)
    {
        Fraction result = new Fraction(num   * other.num,
                                       denom * other.denom);

        result.reduce();
        return result;
    }
}
```

Because it is a private method, can only be called inside other methods in the Fraction class

Fill in the missing code

```java
public class Fraction
{
    public Fraction multiply(Fraction other)
    {
        Fraction result = new Fraction(num   * other.num,
                                       denom * other.denom);

        result.reduce();
        return result;
    }

    public boolean equals(Fraction other)
    {
    }

    public Fraction reciprocal()
    {
    }

    public Fraction add(Fraction other)
    {
    }

    public Fraction subtract(Fraction other)
    {
    }
}
```

Summary

- Objects
  - No-arg default constructors
  - Passing objects of same type to method
  - Private helper methods
- Fraction object
  - Built an object to represent a fraction