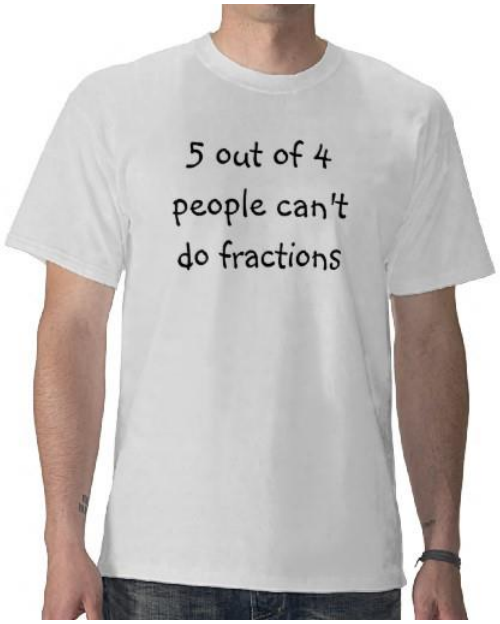
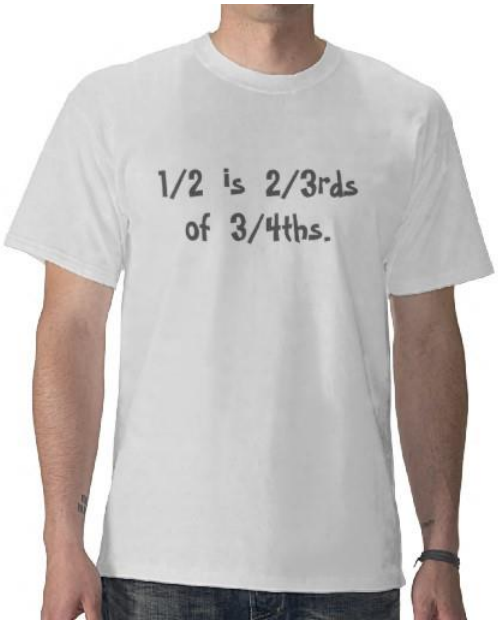
Building a fraction class

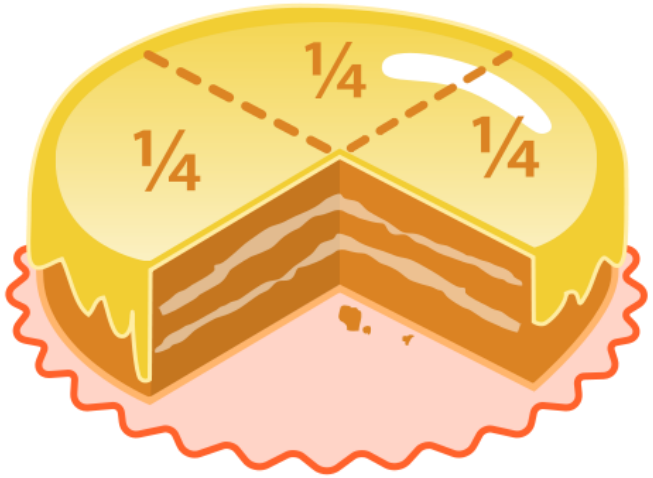




http://www.zazzle.com/fraction+tshirts

Overview

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



Wumpus World – And What are Objects Anyway?

- **The Game.** The Wumpus World game takes place in a cave with different rooms in it. You can think of the cave as an NxM rectangular grid. The player always starts in position 0,0 (in the lower left hand corner of the cave), which is guaranteed to be safe (but it may still be smelly or breezy or glittery).

  The objective of the game is to find the gold. The player will know when he/she is in a room with the gold because there will be a "glitter" in that room. If the player detects a glitter, he/she can pick up the gold and the game is won.

  Bottomless pits are present in some of the rooms. There is a 20% chance that any given room will have a pit. All rooms adjacent to a pit are breezy, that is, a player entering a room adjacent to a pit will detect a breeze. If the player moves into the room with a pit, he/she falls in and dies a horrible death. "Adjacent" to means up down, left or right - you don't have to worry about diagonals.

  There is only one wumpus in the cave, and he is also placed at random. Rooms adjacent to the wumpus are smelly, that is, a player will detect a stench in a room adjacent to a wumpus. The wumpus cannot move. If the player enters a room with the wumpus, he/she will be eaten, and, once again, die a horrible death.

  There is also only one room in the cave that contains the gold. Unlike the other objects, the player has to be in the same room as the gold in order to detect a glitter. Like the wumpus, the gold is placed at random.

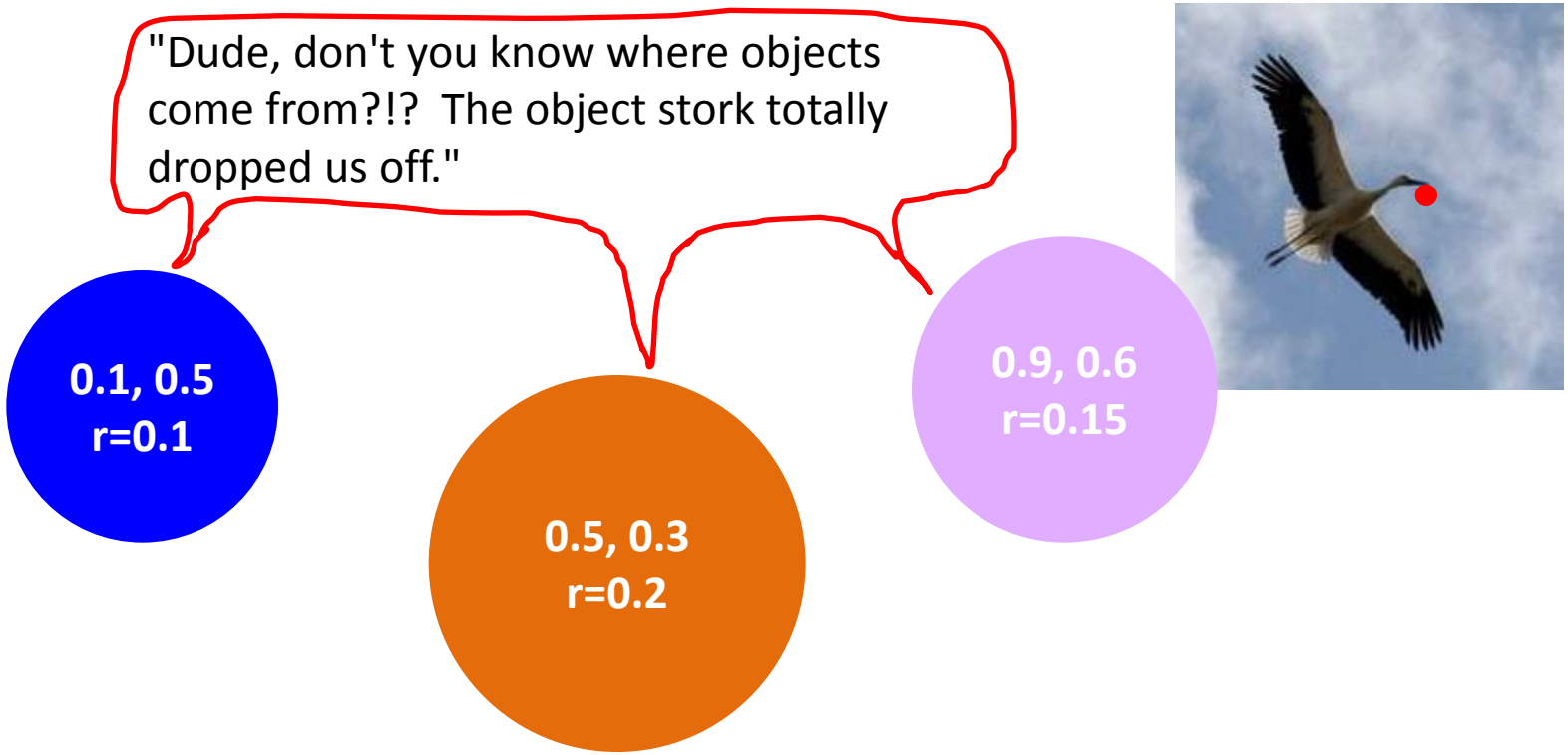
  The player can move up, down, left, or right. The player also has one arrow. Once it's used up, it's gone. It can be used to shoot a wumpus, and can be shot in any direction the player can move in. The wumpus does not need to be in the next room in a given direction - an arrow shot in some direction should travel in that direction until it either hits a wall or the wumpus. If the player is successful in shooting the wumpus, the wumpus will emit a blood-curdling scream, and will no longer be a threat. The only other action the player can perform is to "grab gold".

  When the player first starts the game, he/she does not know (and cannot see) where the location of pits, gold and the wumpus are. The only clues are whether the current room is breezy, smelly, or glittery.

What is a Fraction?

- Well… it has a numerator and a denominator. You can perform mathematical operations on it, like addition, subtraction, multiplication and division. You can reduce it to its lowest terms. If you do any operations on it, you probably want to print the results out.

Hey objects, where did you come from?




```
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)

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

Creating with default constructor

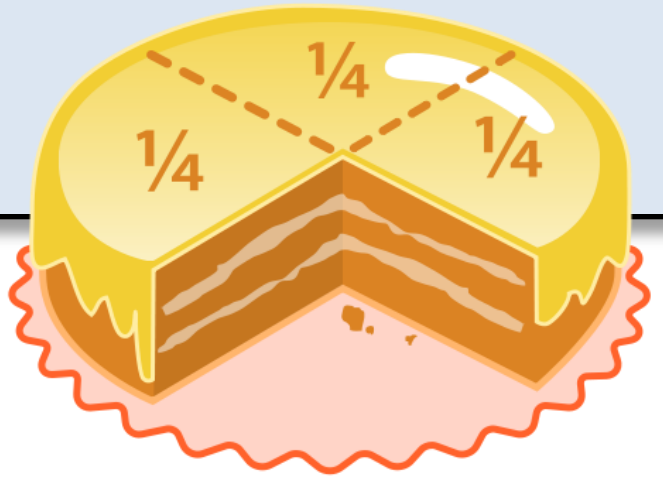
- To create object using no-arg constructor
  - Use empty ()'s after the new
  - Parameter list always sent when new'ing object
    - Java needs to know which constructor to run

```
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

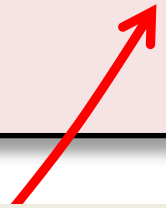
- Rule 2: If you declare any constructor, a default one will not be automatically created

```
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

```
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.

```
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

```
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

```
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

```
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

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

Multiplying fractions

- Attempt 1: Hmmmm, we forgot something…

```
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…

```
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

```
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:

```
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™

```
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()`

```
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

```
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