

# Advanced Class Features

Making the Fraction Class Bigger and Better!

# Base Fraction Class

- Instance fields are private
- Constructor name matches class name
- Constructor doesn't have a return type

```
public class Fraction
{
    private int numer, denomer;

    public Fraction(int n, int d)
    {
        numer = n;
        denomer = d;

        reduce(); //where is this method?
    }

    public double decimalValue()
    {
        return (double)numer/denomer;
    }
}
```

# Fraction's Reduce

The reduce method and the instance fields are marked as **private** because they shouldn't be used outside of this class

```
private void reduce()
{
    if(numer == 0)
        denomer = 1;

    if(denomer < 0)
    {
        denomer *= -1;
        numer *= -1;
    }

    int min = Math.min(numer, denomer);
    for(int i = min; i >= 1; i--)
    {
        if(numer % i == 0 && denomer % i == 0)
        {
            numer /= i;
            denomer /= i;
            break;
        }
    }
}
```

# Fraction usage

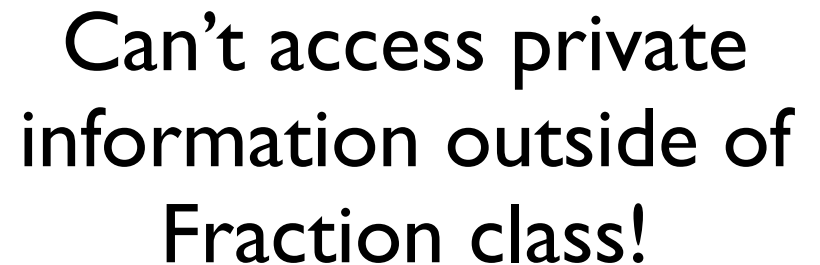
```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        System.out.println(f.decimalValue());

        f.reduce(); //illegal!
        f.numer = 5; //illegal!
    }
}
```

# Fraction usage

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        System.out.println(f.decimalValue());

        f.reduce(); //illegal!
        f.numer = 5; //illegal!
    }
}
```



Can't access private information outside of Fraction class!

# Simple Fraction Additions

```
public int getNumerator()  
{  
    return numer;  
}
```

```
public int getDenominator()  
{  
    return denomer;  
}
```

**Notice that I'm  
using the variables  
numer and  
denomer, not the  
variables n or d!**

# Simple Fraction Additions 2

```
public void setNumerator(int n)
{
    numer = n;
    reduce();
}
```

```
public void setDenominator(int d)
{
    denomer = d;
    reduce();
}
```

**Why are the reduce() calls important in these methods?**

# Additional Constructors

```
public Fraction(int n)
{
    numer = n;
    denomer = 1;
    reduce();
}
```

**What is the first constructor good for?**

```
public Fraction(Fraction other)
{
    numer = other.numer;
    denomer = other.denomer;
}
```

**The second constructor is called a COPY constructor**

# Constructor Use

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(3);
        Fraction f2 = new Fraction(f1);
    }
}
```

# Constructor Use

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(3);
        Fraction f2 = new Fraction(f1);
    }
}
```


f:



1/2

# Constructor Use


```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(3);
        Fraction f2 = new Fraction(f1);
    }
}
```

f: 

f1: 

# Constructor Use

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(3);
        Fraction f2 = new Fraction(f1);
    }
}
```

f: 

f1: 

f2: 

# Copy Constructor Mechanics

```
public Fraction(Fraction other)
{
    numer = other.numer;
    denomer = other.denomer;
}
```

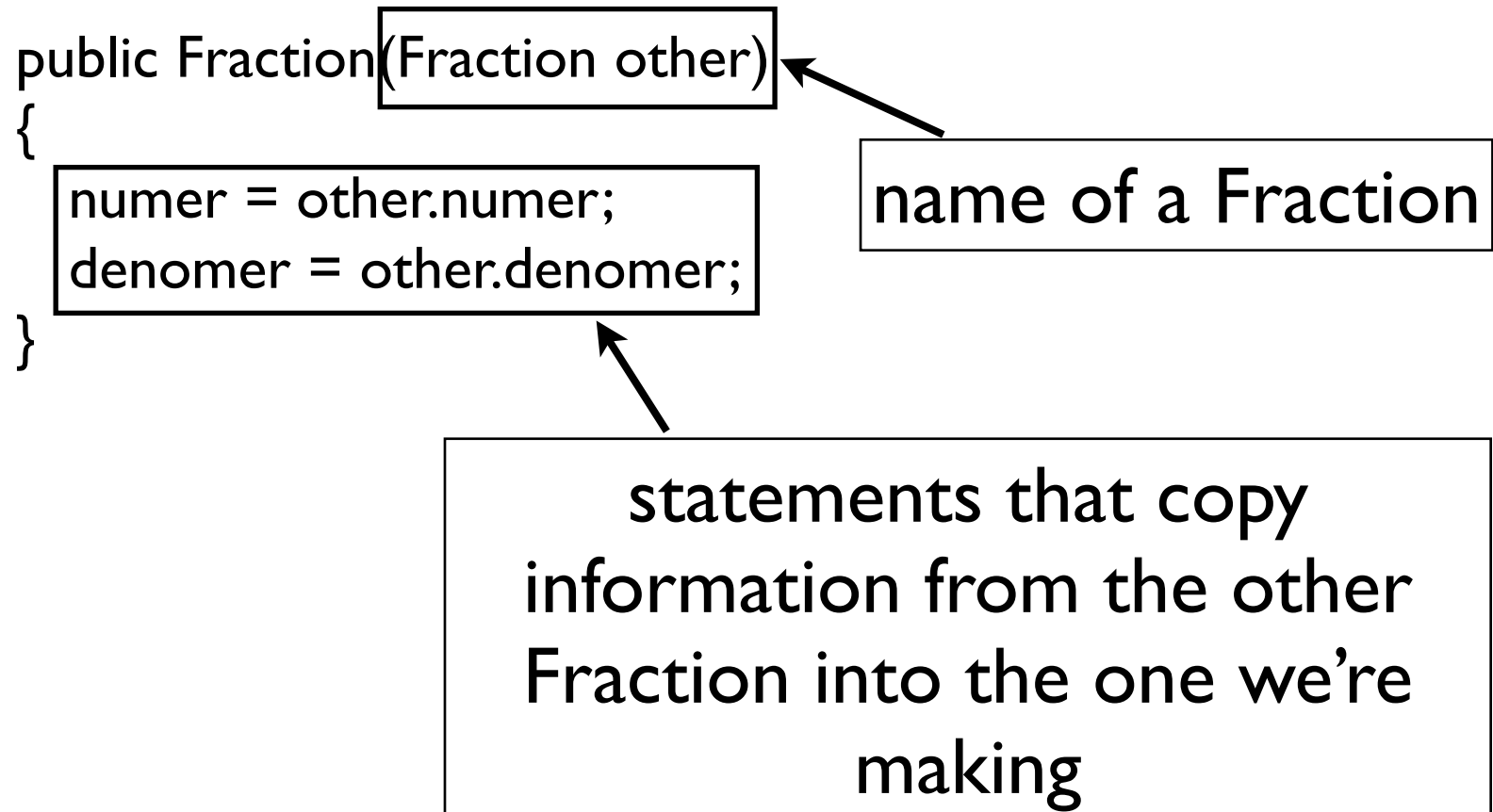
# Copy Constructor Mechanics

```
public Fraction(Fraction other)
{
    numer = other.numer;
    denomer = other.denomer;
}
```



name of a Fraction

# Copy Constructor Mechanics



# Copy Constructor Mechanics

```
public Fraction(Fraction other)
{
    numer = other.numer;
    denomer = other.denomer;
}
```

name of a Fraction

Why is **other.numer** legal? I thought numer was private...

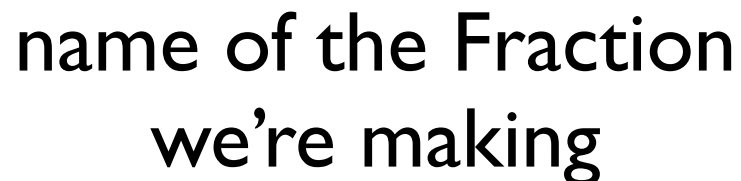
statements that copy information from the other Fraction into the one we're making

# Copy Constructor Usage

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(f);
    }
}
```

# Copy Constructor Usage

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(f);
    }
}
```



name of the Fraction  
we're making

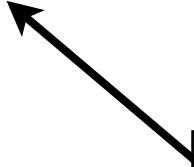
# Copy Constructor Usage

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(f);
    }
}
```

name of the  
“other” Fraction



name of the Fraction  
we're making



# You Try

- Write the Copy Constructor for the Die class

```
public class Die
{
    private int numSides;

    //YOUR CODE FOR COPY CONSTRUCTOR
    //
    //
    //
}
```

# Answer

- Write the Copy Constructor for the Die class

```
public class Die
{
    private int numSides;

    public Die(Die other)
    {
        numSides = other.numSides;
    }
}
```

# Returning Class Types

- What we'd like to do... but can't

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(3);

        Fraction f2 = f + f1; //illegal!
    }
}
```

# Returning Class Types

- What we can do with a slight change

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(3);

        Fraction f2 = f.add(f1);
    }
}
```


# Returning Class Types

- What we can do with a slight change

```
public class Driver
{
    public static void main(String[] args)
    {
        Fraction f = new Fraction(2, 4);
        Fraction f1 = new Fraction(3);

        Fraction f2 = f.add(f1);
    }
}
```

Looking at this code, what do we know about the add method?



# Add Method

```
Fraction f2 = f.add(f1);
```

- It **returns** a Fraction
- It has a Fraction as a parameter
- It's a method of the Fraction class

# Add Method

```
Fraction f2 = f.add(f1);
```

- It **returns** a Fraction
- It has a Fraction as a parameter
- It's a method of the Fraction class

How do you add fractions?

# Add Method

$$\frac{4}{3} + \frac{2}{5} = \frac{4 \cdot 5 + 2 \cdot 3}{3 \cdot 5}$$

```
public Fraction add(Fraction other)
{
    int newDenomer = denomer * other.denomer;
    int newNumer = numer * other.denomer + other.numer * denomer;

    return new Fraction(newNumer, newDenomer);
}
```

# Add Method

$$\frac{4}{3} + \frac{2}{5} = \frac{4 \cdot 5 + 2 \cdot 3}{3 \cdot 5}$$

```
public Fraction add(Fraction other)
{
    int newDenomer = denomer * other.denomer;
    int newNumer = numer * other.denomer + other.numer * denomer;

    return new Fraction(newNumer, newDenomer);
}
```

Why is no reduce() call needed?

# Add Method

$$\frac{4}{3} + \frac{2}{5} = \frac{4 \cdot 5 + 2 \cdot 3}{3 \cdot 5}$$

```
public Fraction add(Fraction other)
{
    int newDenomer = denomer * other.denomer;
    int newNumer = numer * other.denomer + other.numer * denomer;

    return new Fraction(newNumer, newDenomer);
}
```

Why is no reduce() call needed?

Notice that this method returns a class type!

# You Try

- Write the multiply method for the Fraction class

```
public class Fraction
{
    private int numer, denomer;

    //YOUR CODE
    //
    //
    //
    //
}
```

# Answer

- Write the multiply method for the Fraction class

```
public class Fraction
{
    private int numer, denomer;

    public Fraction multiply(Fraction other)
    {
        int newNumer = other.numer * numer;
        int newDenomer = other.denomer * denomer;

        return new Fraction(newNumer, newDenomer);
    }
}
```

# Summary

- A copy constructor takes a parameter that is the same type as the class it is within!
- A method/constructor within class A may access the private information of other **instances** (A type variables) of class A
- You may return a class type

# One last question

- Identify each of the parts of this constructor

```
public Die(Die other) {  
    faceValue = other.faceValue;  
}
```

# One last question

- Identify each of the parts of this constructor

```
public Die(Die other) {  
    faceValue = other.faceValue;  
}
```

public: access permissions

# One last question

- Identify each of the parts of this constructor

```
public Die(Die other) {  
    faceValue = other.faceValue;  
}
```

public: access permissions

First Die: constructor name

# One last question

- Identify each of the parts of this constructor

```
public Die(Die other) {  
    faceValue = other.faceValue;  
}
```

public: access permissions

First Die: constructor name

Die other: parameter

# One last question

- Identify each of the parts of this constructor

```
public Die(Die other) {  
    faceValue = other.faceValue;  
}
```

public: access permissions

faceValue:

First Die:

Die other:

# One last question

- Identify each of the parts of this constructor

```
public Die(Die other) {  
    faceValue = other.faceValue;  
}
```

public: access permissions

faceValue: instance field

Die: constructor name

Die other: parameter

other.faceValue:

other's instance field