Creating objects

The class Fraction represents a fraction.

Question

How do you create a Fraction object with numerator 1 and denominator 2?

Creating objects

The class Fraction represents a fraction.

Question

How do you create a Fraction object with numerator 1 and denominator 2?

Answer

```
Fraction half = new Fraction(1, 2);
```

Creating objects

The class Fraction represents a fraction.

Question

How do you create a Fraction object with numerator 1 and denominator 2?

Answer

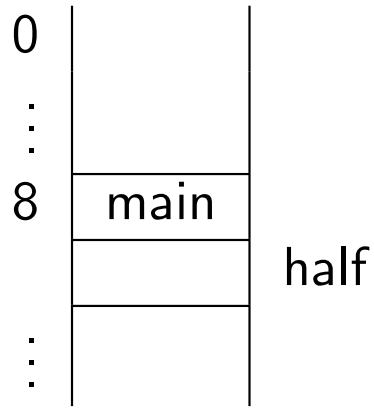
```
Fraction half = new Fraction(1, 2);
```

Question

Draw the memory diagram.

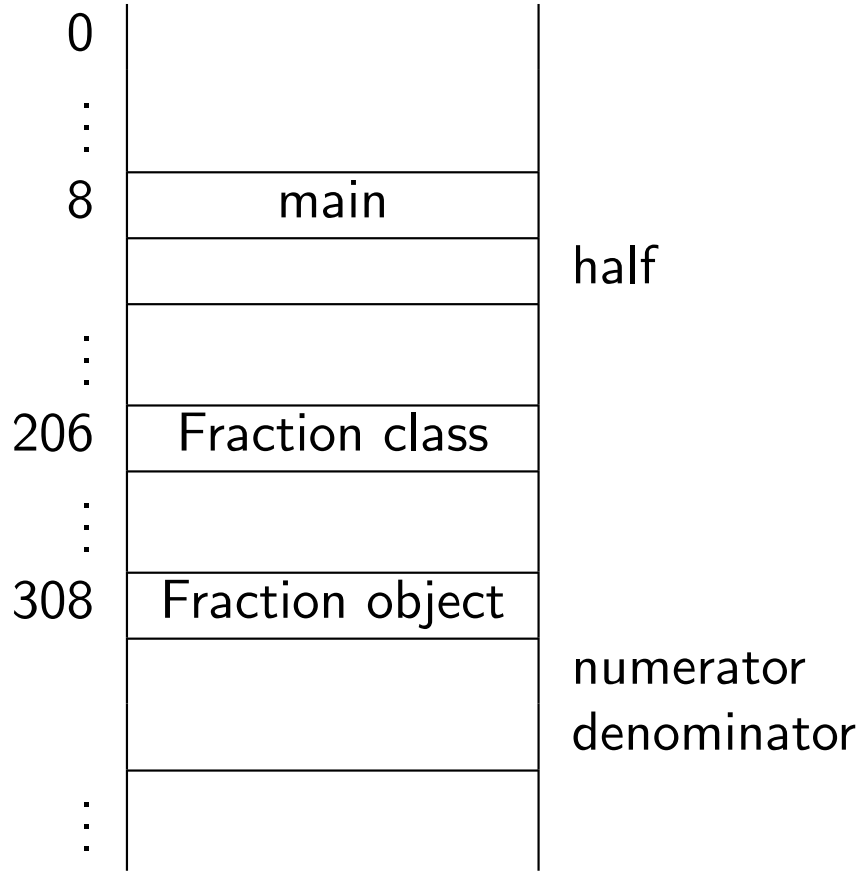
Memory diagram

```
Fraction half = new Fraction(1, 2);
```



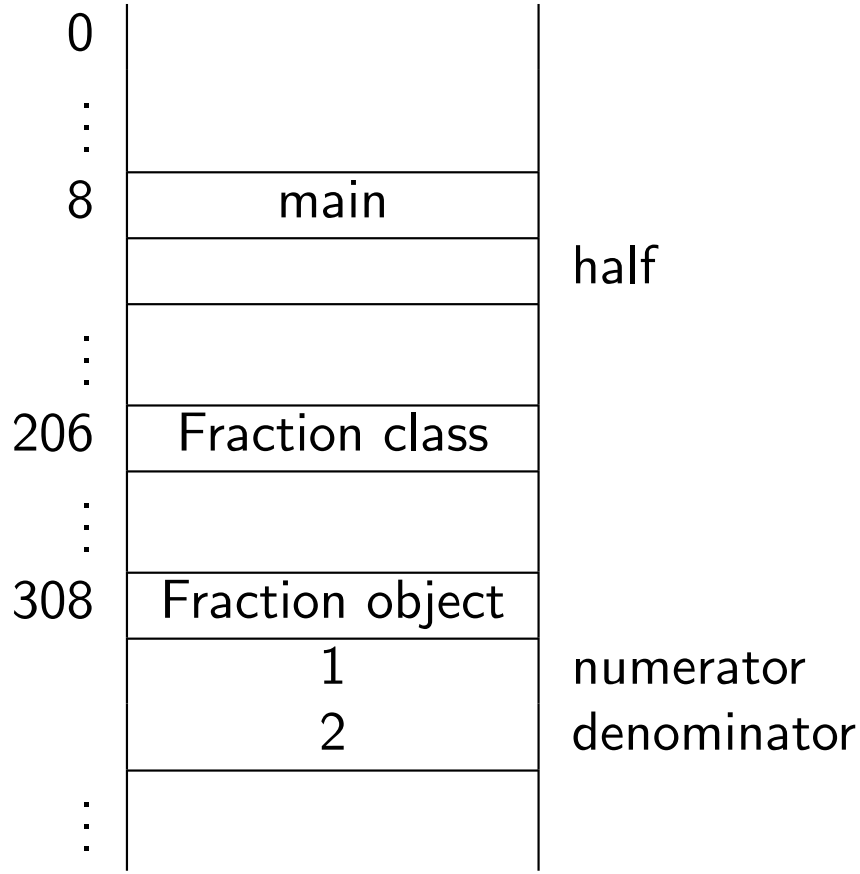
Memory diagram

```
Fraction half = new Fraction(1, 2);
```



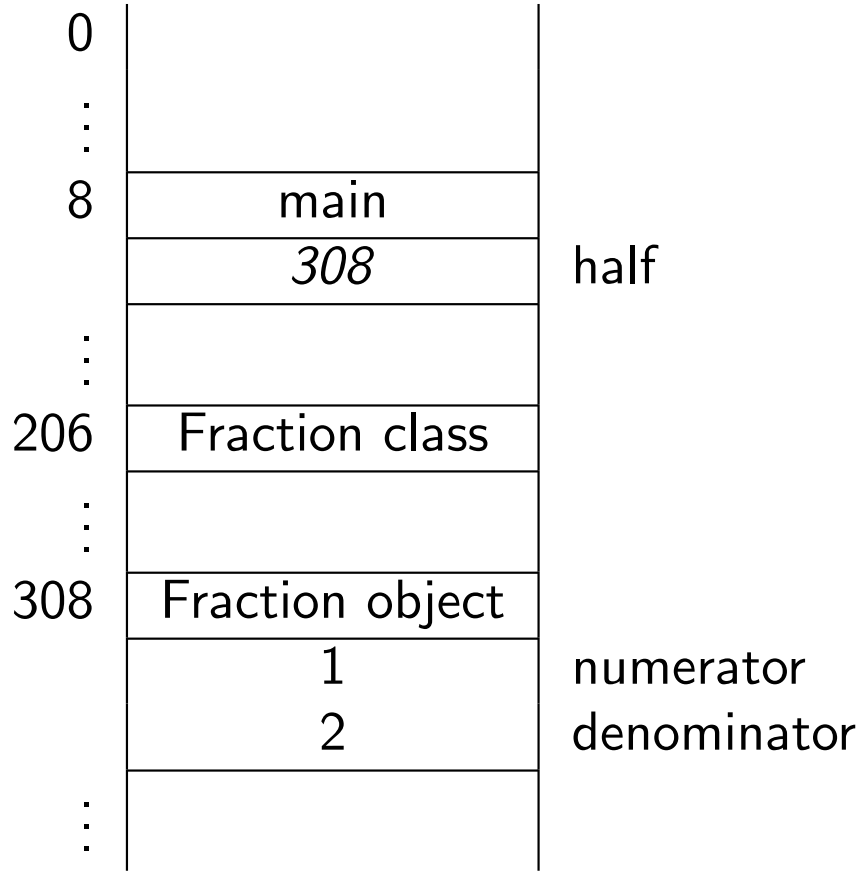
Memory diagram

```
Fraction half = new Fraction(1, 2);
```



Memory diagram

```
Fraction half = new Fraction(1, 2);
```



Creating objects

The class Fraction represents a fraction.

Question

How do you create a random Fraction object?

Creating objects

The class Fraction represents a fraction.

Question

How do you create a random Fraction object?

Answer

```
Fraction random = Fraction.createFraction();
```

Creating objects

The class Fraction represents a fraction.

Question

How do you create a random Fraction object?

Answer

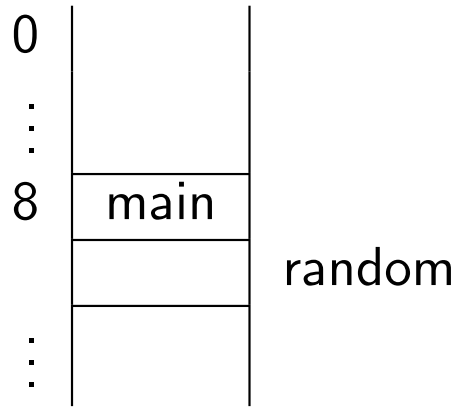
```
Fraction random = Fraction.createFraction();
```

Question

Draw the memory diagram.

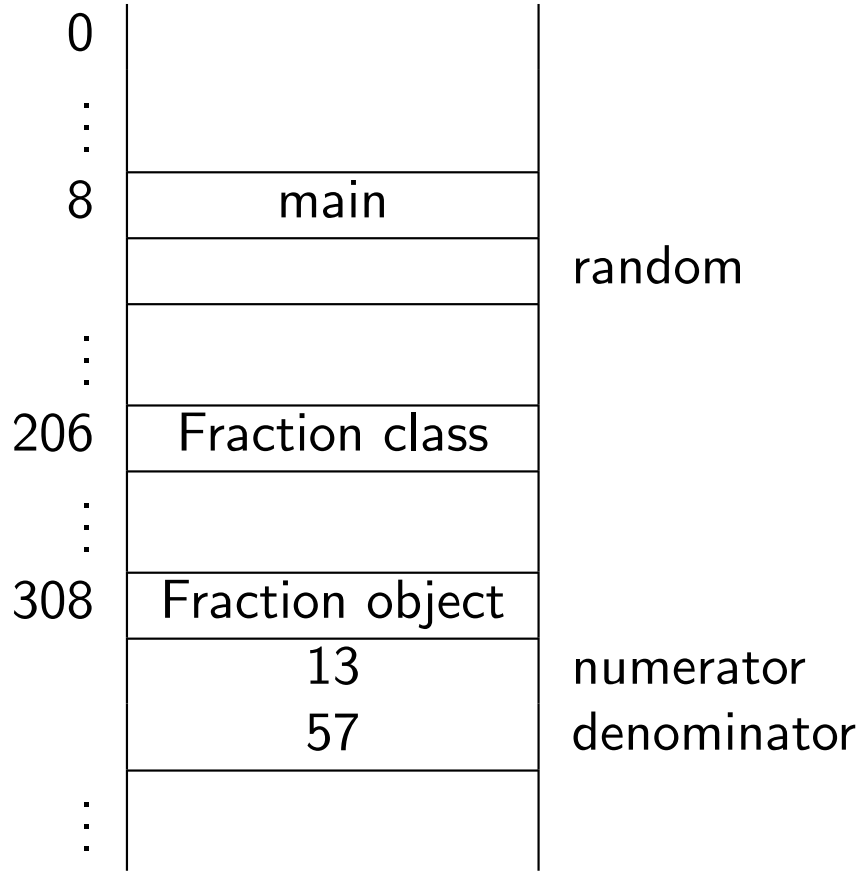
Memory diagram

```
Fraction random = Fraction.createFraction ();
```



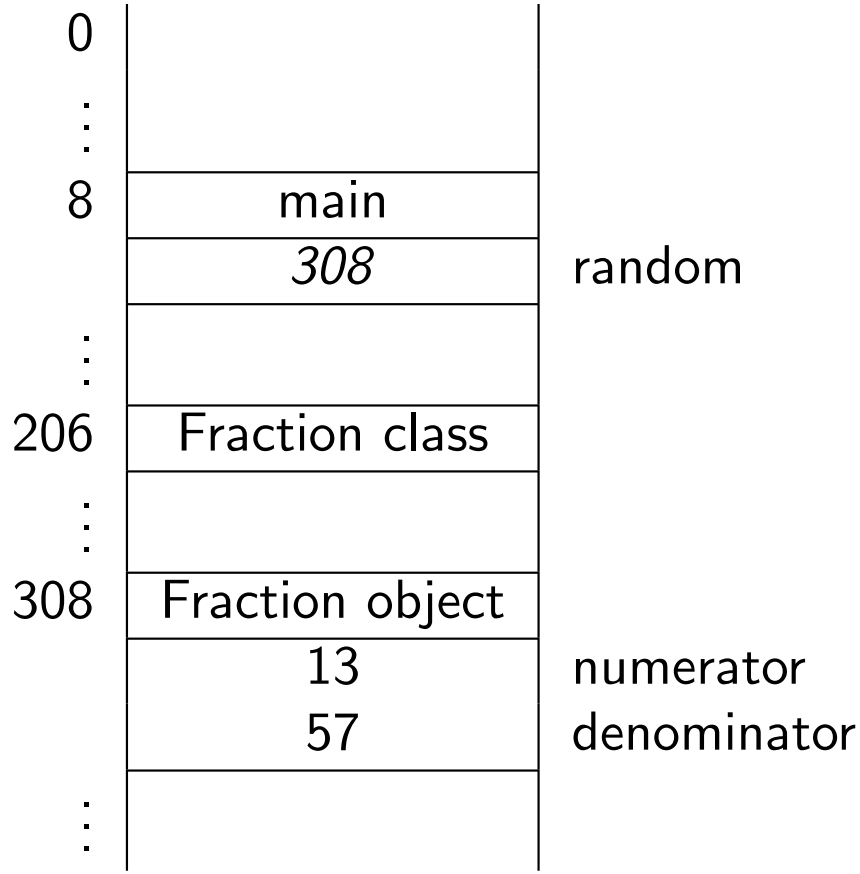
Memory diagram

```
Fraction random = Fraction.createFraction();
```



Memory diagram

```
Fraction random = Fraction.createFraction();
```



Compute $\frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7}$

```
long numerator = 1;
long denominator = 7;
Fraction seventh = new Fraction(numerator, denominator);
Fraction sum = new Fraction();
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
String result = sum.toString();
output.println(result);
```

Check whether $\frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7}$ is 1

To check whether $\frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7}$ is equal to 1, let us first contrast ...

. . . objects versus object references

Question

```
Fraction f = new Fraction();
Fraction g = new Fraction();
Fraction h = new Fraction(1, 2);
Fraction i = new Fraction(0, 2);
Fraction j = g;
Fraction k = j;
```

At the end of the execution of the above snippet, how many objects are there and how many objects references are there?

. . . objects versus object references

Question

```
Fraction f = new Fraction();
Fraction g = new Fraction();
Fraction h = new Fraction(1, 2);
Fraction i = new Fraction(0, 2);
Fraction j = g;
Fraction k = j;
```

At the end of the execution of the above snippet, how many objects are there and how many objects references are there?

Answer

Four objects and six object references.

. . . objects versus object references

Question

```
Fraction f = new Fraction();
Fraction g = new Fraction();
Fraction h = new Fraction(1, 2);
Fraction i = new Fraction(0, 2);
Fraction j = g;
Fraction k = j;
```

At the end of the execution of the above snippet, how many objects are there and how many objects references are there?

Answer

Four objects and six object references.

Exercise

Draw the diagram representing the memory once the execution has reached the end of the above snippet.

Solution to exercise

100	main	
	300	f
	400	g
	500	h
	600	i
	400	j
	400	k
200	Fraction class	
300	Fraction object	
	0	numerator
	1	denominator
400	Fraction object	
	0	numerator
	1	denominator
500	Fraction object	
	1	numerator
	2	denominator
600	Fraction object	
	0	numerator
	2	denominator

When are two objects references the same?

What do we mean by the same?

- Do they refer to the same object, that is, do they have the same identity?
- Do they refer to objects with the same state, that is, do their attributes have the same values?

When are two objects references the same?

What do we mean by the same?

- Do they refer to the same object, that is, do they have the same identity?
- Do they refer to objects with the same state, that is, do their attributes have the same values?

```
Fraction sum = ...
Fraction one = new Fraction(1, 1);
boolean identical = (sum == one);
boolean same = sum.equals(one);
```

When are two objects references the same?

Question

```
Fraction f = new Fraction();
Fraction g = new Fraction();
Fraction h = new Fraction(1, 2);
Fraction i = new Fraction(0, 2);
Fraction j = g;
Fraction k = j;
```

Fill the following table with true (T) and false (F).

==	f	g	h	i	j	k
f						
g						
h						
i						
j						
k						

When are two objects references the same?

Answer

```
Fraction f = new Fraction();
Fraction g = new Fraction();
Fraction h = new Fraction(1, 2);
Fraction i = new Fraction(0, 2);
Fraction j = g;
Fraction k = j;
```

==	f	g	h	i	j	k
f	T	F	F	F	F	F
g	F	T	F	F	T	T
h	F	F	T	F	F	F
i	F	F	F	T	F	F
j	F	T	F	F	T	T
k	F	T	F	F	T	T

When are two objects references the same?

Question

```
Fraction f = new Fraction();
Fraction g = new Fraction();
Fraction h = new Fraction(1, 2);
Fraction i = new Fraction(0, 2);
Fraction j = g;
Fraction k = j;
```

Fill the following table with true (T) and false (F).

equals	f	g	h	i	j	k
f						
g						
h						
i						
j						
k						

When are two objects references the same?

Answer

```
Fraction f = new Fraction();
Fraction g = new Fraction();
Fraction h = new Fraction(1, 2);
Fraction i = new Fraction(0, 2);
Fraction j = g;
Fraction k = j;
```

equals	f	g	h	i	j	k
f	T	T	F	T	T	T
g	T	T	F	T	T	T
h	F	F	T	F	F	F
i	T	T	F	T	T	T
j	T	T	F	T	T	T
k	T	T	F	T	T	T

Check whether $\frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7}$ is 1

```
long numerator = 1;
long denominator = 7;
Fraction seventh = new Fraction(numerator, denominator);
Fraction sum = new Fraction();
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
sum.add(seventh);
Fraction one = new Fraction(1, 1);
boolean equal = sum.equals(one);
output.println(equal);
```

More memory diagrams

```
Fraction f = new Fraction();
Fraction g = new Fraction(1, 2);
Fraction h = new Fraction();
f = g;
```

Draw the diagram representing the memory once the execution has reached the end of the snippet.

More memory diagrams

100	main	
	400	f
	400	g
	500	h
200	Fraction class	
300	Fraction object	
	0	numerator
	1	denominator
400	Fraction object	
	1	numerator
	2	denominator
500	Fraction object	
	0	numerator
	1	denominator

Garbage collection

Question

How many object references refer to the object at address 300?

Garbage collection

Question

How many object references refer to the object at address 300?

Answer

Zero.

The object at address 300 has become an orphan.

Every now and then, the garbage collector removes all orphans from memory.

Garbage collection

```
HugeObject elephant = new HugeObject();
...
/* at this point in the code we do not
   need the elephant any more */
```

Question

How can we make the `HugeObject` an orphan so that it can be garbage collected?

Garbage collection

```
HugeObject elephant = new HugeObject();
...
/* at this point in the code we do not
   need the elephant any more */
```

Question

How can we make the `HugeObject` an orphan so that it can be garbage collected?

Answer

```
elephant = null;
```

What is `null`?

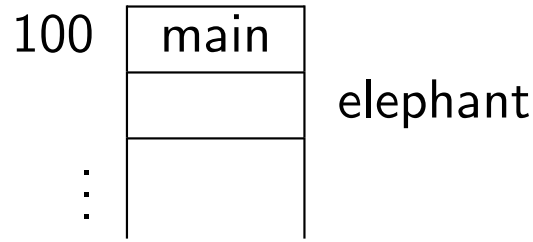
According to the Collins English dictionary

null . . . **4.** nonexistent; amounting to nothing.

In Java, `null` is a reserved word and it is compatible with any reference type.

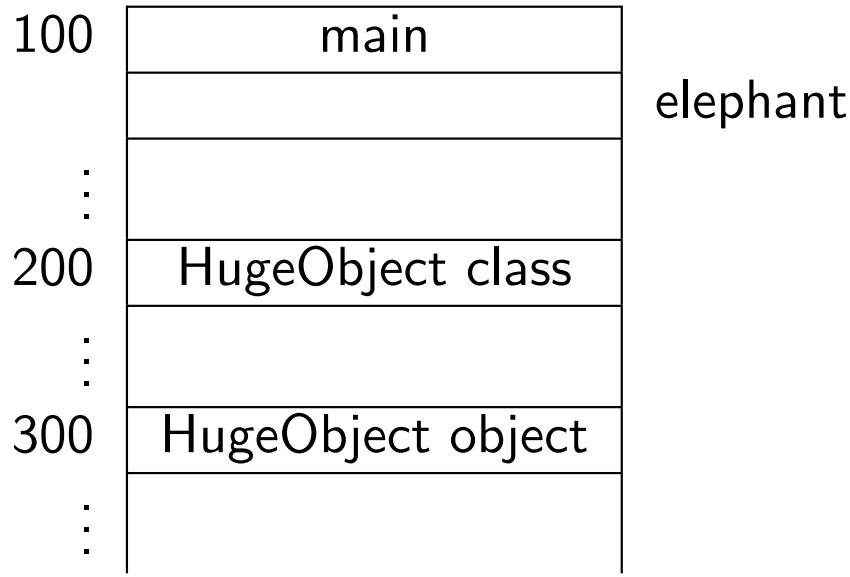
Null

```
HugeObject elephant = new HugeObject();
 ...
/* at this point in the code we do not
   need the elephant any more */
elephant = null;
```



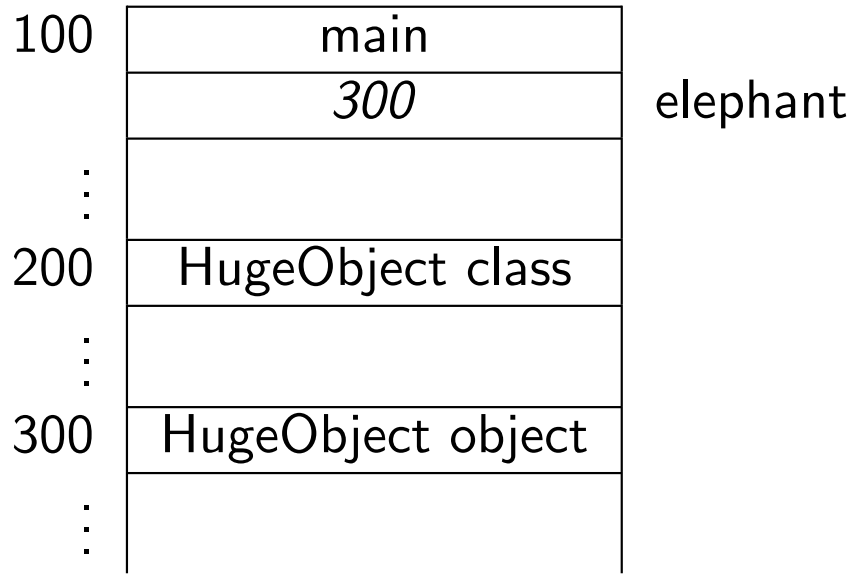
Null

```
HugeObject elephant = new HugeObject();
 ...
/* at this point in the code we do not
   need the elephant any more */
elephant = null;
```



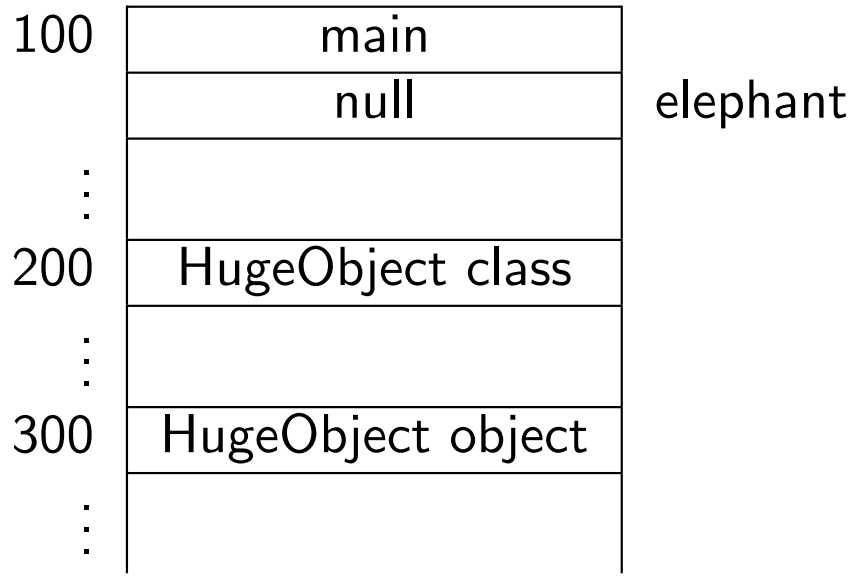
Null

```
HugeObject elephant = new HugeObject();
 ...
/* at this point in the code we do not
   need the elephant any more */
elephant = null;
```



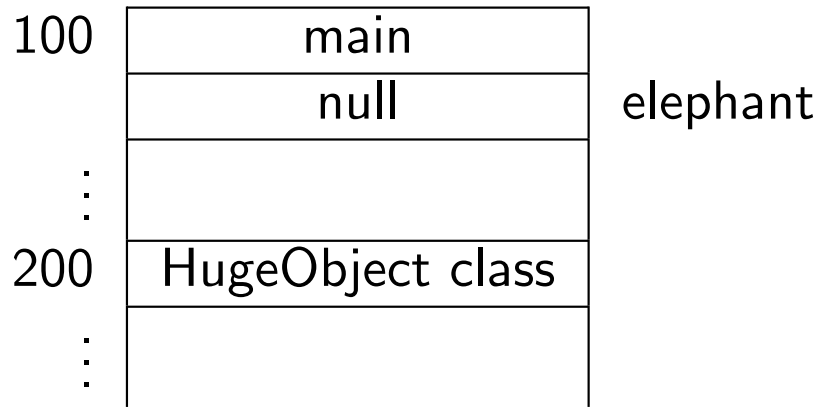
Null

```
HugeObject elephant = new HugeObject();
...
/* at this point in the code we do not
 need the elephant any more */
elephant = null;
```



Null

```
HugeObject elephant = new HugeObject();
...
/* at this point in the code we do not
 need the elephant any more */
elephant = null;
```



null

Question

What happens when you invoke a method on an object reference whose value is `null`?

null

Question

What happens when you invoke a method on an object reference whose value is `null`?

Answer

Let's try it!

null

Question

What happens when you invoke a method on an object reference whose value is `null`?

Answer

Let's try it!

Answer

The app crashes with a `NullPointerException`.

Question

Let `f` be an object reference whose value is not `null`. What are the values of

- `null == null`,
- `f == null`,
- `null == f`,
- `null.equals(null)`,
- `f.equals(null)` and
- `null.equals(f)`?

null

Question

Let `f` be an object reference whose value is not `null`. What are the values of

- `null == null`,
- `f == null`,
- `null == f`,
- `null.equals(null)`,
- `f.equals(null)` and
- `null.equals(f)`?

Answer

true, false, false, crash, false, crash.

Observe the state of an object

Question

What is the state of an object?

Observe the state of an object

Question

What is the state of an object?

Answer

Its attributes and their values.

Observe the state of an object

Question

What is the state of an object?

Answer

Its attributes and their values.

To observe the state of an object, it suffices to answer the

Question

How do you determine the value of an attribute?

Observe the state of an object

Question

What is the state of an object?

Answer

Its attributes and their values.

To observe the state of an object, it suffices to answer the

Question

How do you determine the value of an attribute?

Answer

By means of a method. These methods are known as accessors and by convention have the name get*N* where *N* is the name of the attribute.

Change the state of an object

To change the state of an object, it suffices to answer the

Question

How do you change the value of an attribute?

Change the state of an object

To change the state of an object, it suffices to answer the

Question

How do you change the value of an attribute?

Answer

By means of a method. These methods are known as mutators and by convention have the name set$N$ where N is the name of the attribute.

Why do we have accessors and mutators?

Question

Rather than introducing an accessor and mutator for a private attribute, why not simply make the attribute public?

Why do we have accessors and mutators?

Question

Rather than introducing an accessor and mutator for a private attribute, why not simply make the attribute public?

Answer

An accessor and mutator allow us to ensure that the attribute always has a particular property. For example, we can ensure that the `quantity` attribute of an `Investment` object is never negative.

How to ensure that the quantity is never negative?

- `public void setQuantity(int quantity)`
  Sets the quantity of this investment to the given quantity.
  Parameters: `quantity` - the new quantity of this investment
  Precondition: `quantity >= 0`
- `public boolean setQuantity(int quantity)`
  Sets the quantity of this investment to the given quantity if it is nonnegative.
  Parameters: `quantity` - the new quantity of this investment
  Returns: true if `quantity >= 0`, false otherwise
- `public void setQuantity(int quantity) throws Exception`
  Sets the quantity of this investment to the given quantity.
  Parameters: `quantity` - the new quantity of this investment
  Throws: `Exception` - if `quantity < 0`

Accessors and mutators

- The attribute has both an accessor and a mutator.
  Example: `numerator` of `Fraction`
- The attribute has an accessor but no mutator.
  Example: `blue` of `Color`
- The attribute has a mutator but no accessor.
  Example: ?
- The attribute has neither an accessor nor a mutator.
  Example: `value` of `Integer`

Fractions

Question

How many different fractions can be represented by `Fraction` objects?

Fractions

Question

How many different fractions can be represented by `Fraction` objects?

Answer

Less than 2^{128}. Note that $\frac{1}{2}$ and $\frac{2}{4}$ represent the same fraction. Hence, computing the exact number is tricky.

Not all fractions can be represented by a `Fraction` object.

Fractions

Question

Consider

```
Fraction f = new Fraction(..., ...);
Fraction g = new Fraction(..., ...);
f.operation(g);
```

For which values for `...` and for which `operation` do we get an incorrect result?

Fractions

Question

Consider

```
Fraction f = new Fraction(..., ...);
Fraction g = new Fraction(..., ...);
f.operation(g);
```

For which values for `...` and for which `operation` do we get an incorrect result?

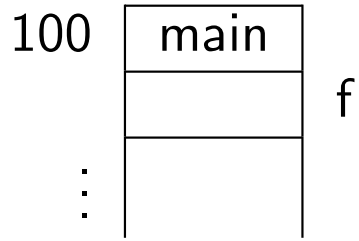
Question

There are many correct answers, including

```
Fraction f = new Fraction(1, Long.MAX_VALUE);
Fraction g = new Fraction(1, 2);
f.multiply(g);
```

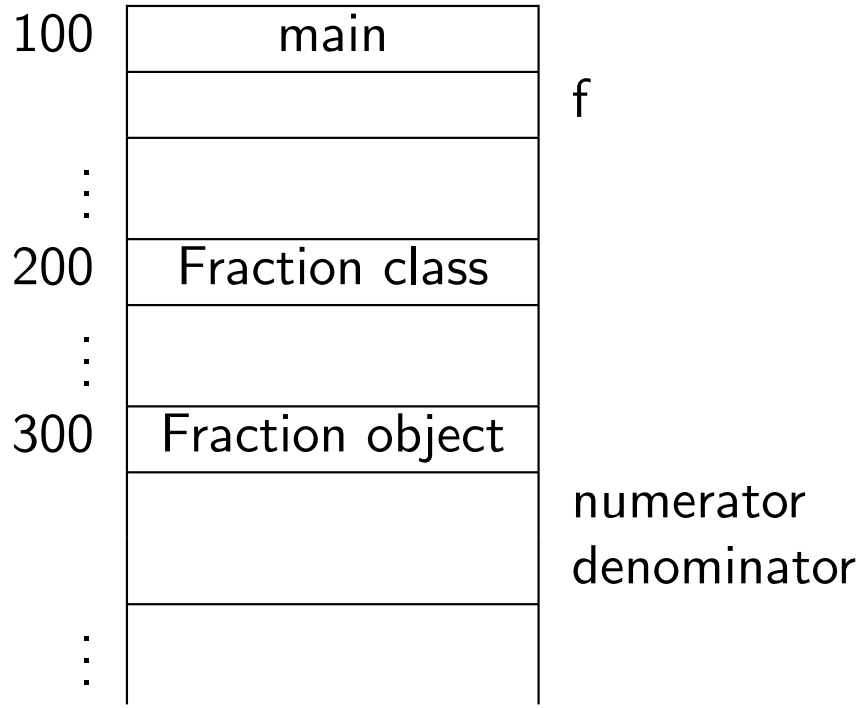
Yet more memory diagrams

```
Fraction f = new Fraction(1, Long.MAX_VALUE);
```



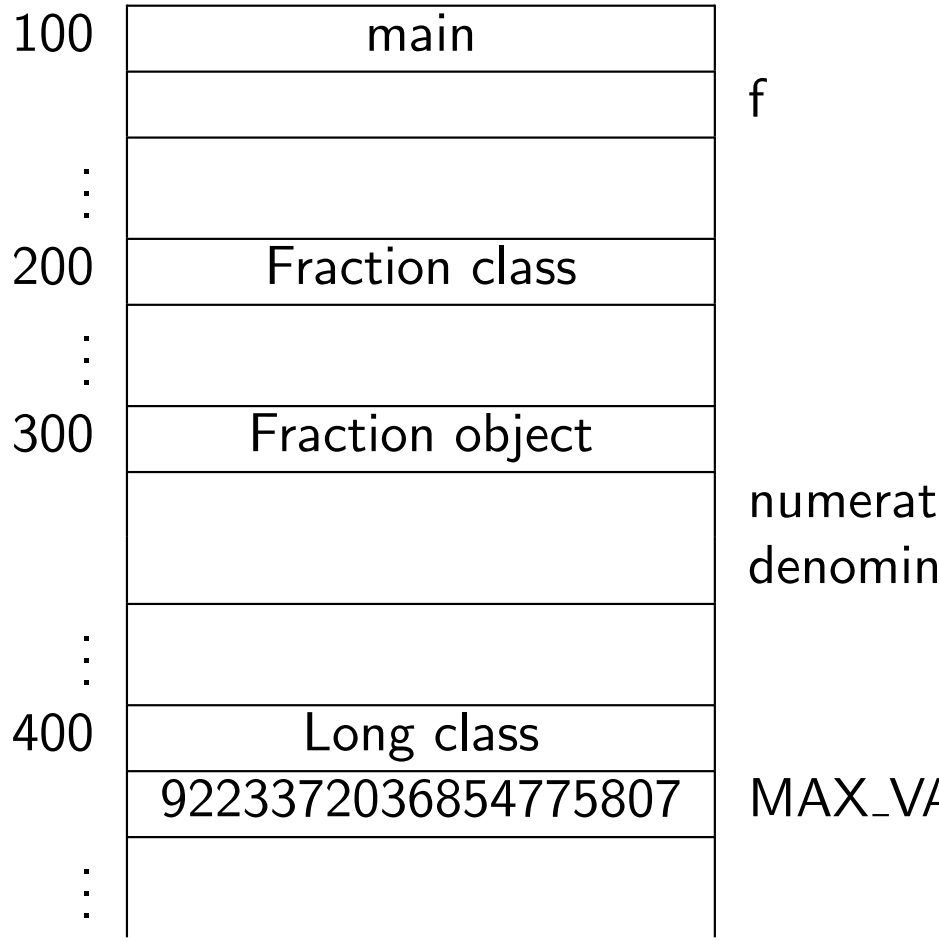
Yet more memory diagrams

```
Fraction f = new Fraction(1, Long.MAX_VALUE);
```



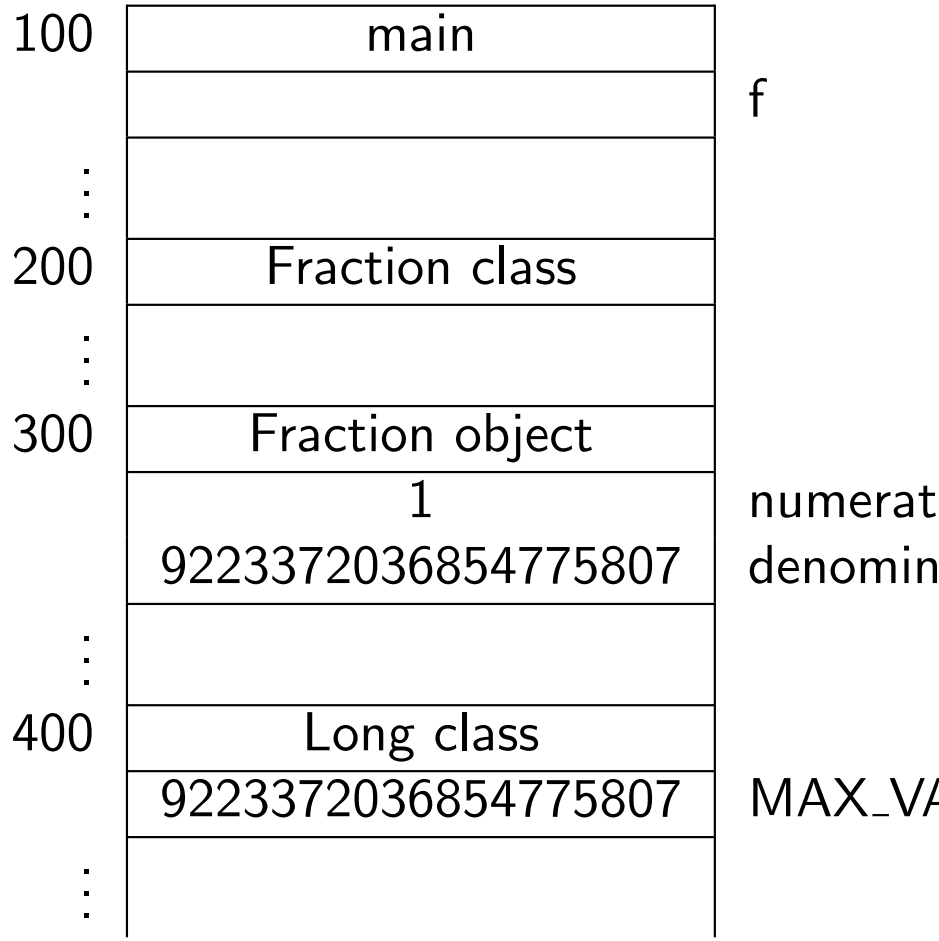
Yet more memory diagrams

```
Fraction f = new Fraction(1, Long.MAX_VALUE);
```



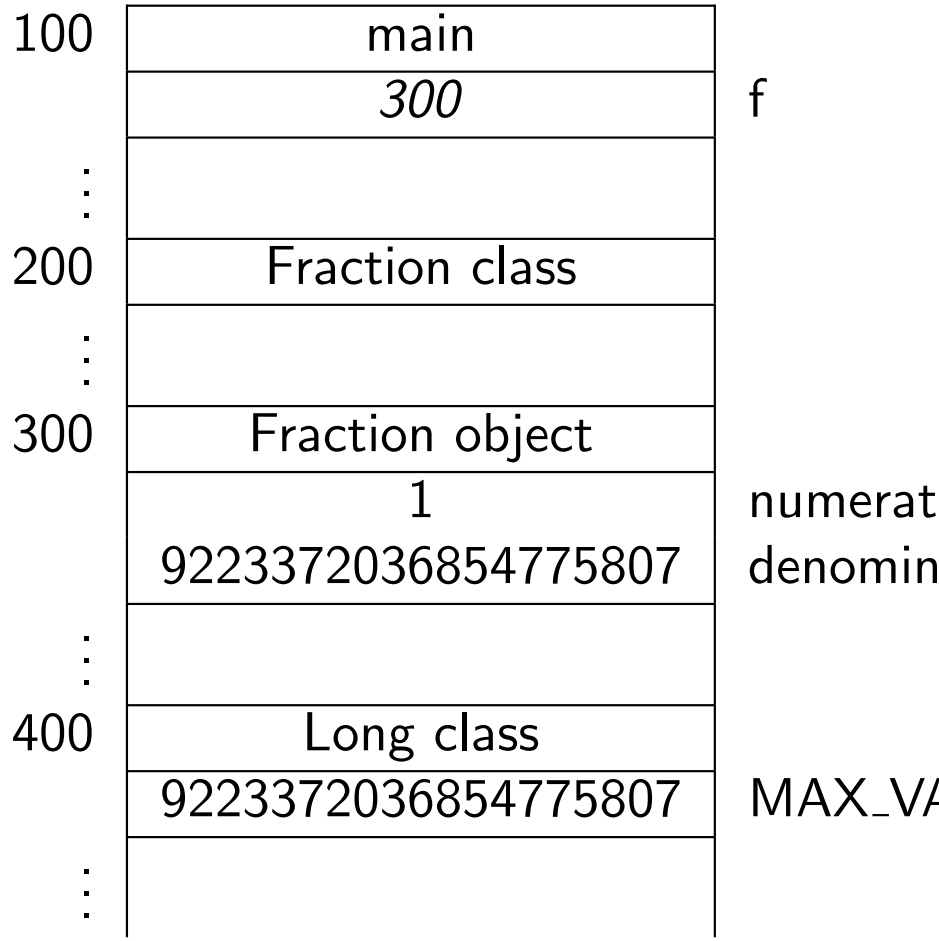
Yet more memory diagrams

```
Fraction f = new Fraction(1, Long.MAX_VALUE);
```



Yet more memory diagrams

```
Fraction f = new Fraction(1, Long.MAX_VALUE);
```



Static versus non-static features

- Static attributes contain data related to the class (and all its objects).
- Non-static attributes contain data related to individual objects.
- Static methods manipulate data related to the class (and all its objects).
- Non-static methods manipulate data related to individual objects.

Static versus non-static features

Let `IPhone` be a class representing iPhones.

Question

The attribute `generation` of type `int` describes which generation an iPhone is. Is this attribute static or non-static?

Static versus non-static features

Let `IPhone` be a class representing iPhones.

Question

The attribute `generation` of type `int` describes which generation an iPhone is. Is this attribute static or non-static?

Answer

Non-static, since this data is related each individual iPhone.

Static versus non-static features

Let `IPhone` be a class representing iPhones.

Question

The attribute `generation` of type `int` describes which generation an iPhone is. Is this attribute static or non-static?

Answer

Non-static, since this data is related each individual iPhone.

Question

The attribute `number` of type `int` describes the number of iPhones that have been sold. Is this attribute static or non-static?

Static versus non-static features

Let `IPhone` be a class representing iPhones.

Question

The attribute `generation` of type `int` describes which generation an iPhone is. Is this attribute static or non-static?

Answer

Non-static, since this data is related each individual iPhone.

Question

The attribute `number` of type `int` describes the number of iPhones that have been sold. Is this attribute static or non-static?

Answer

Static, since this data is not related to an individual iPhone but to all iPhones.

Passing of arguments

Question

What is the difference between pass-by-value and pass-by-reference?

Passing of arguments

Question

What is the difference between pass-by-value and pass-by-reference?

Answer

In pass-by-value, the values of the arguments are passed, whereas in pass-by-reference, the addresses of the arguments are passed.

Pass-by-value

Question

What is the output produced by the following code snippet?

```
int x = 0;
int y = 1;
Magic.swap(x, y);
output.println(x);
output.println(y);
```

Pass-by-value

Question

What is the output produced by the following code snippet?

```
int x = 0;
int y = 1;
Magic.swap(x, y);
output.println(x);
output.println(y);
```

Answer

```
0
1
```

Pass-by-value or pass-by-reference?

Question

The code snippet

```
Fraction f = new Fraction(0, 1);
Fraction g = new Fraction(1, 1);
Magic.swap(f, g);
output.println(f);
output.println(g);
```

produces the output

```
1/1
0/1
```

Can this output be a result of pass-by-value?

Pass-by-value or pass-by-reference?

Question

The code snippet

```
Fraction f = new Fraction(0, 1);
Fraction g = new Fraction(1, 1);
Magic.swap(f, g);
output.println(f);
output.println(g);
```

produces the output

```
1/1
0/1
```

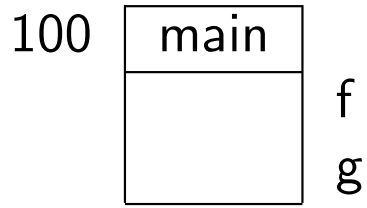
Can this output be a result of pass-by-value?

Answer

Yes!

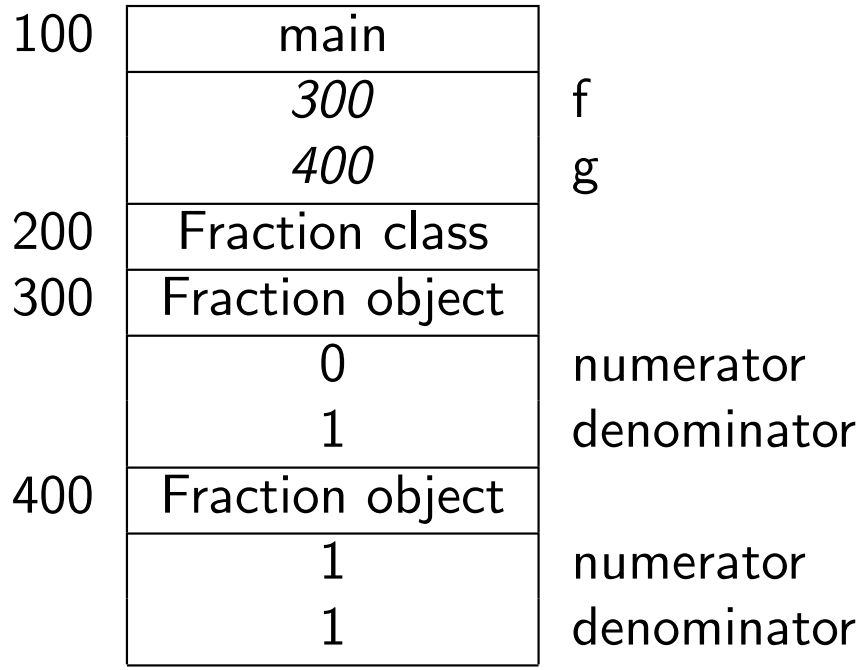
Pass-by-value

```
Fraction f = new Fraction(0, 1);
Fraction g = new Fraction(1, 1);
Magic.swap(f, g);
output.println(f);
output.println(g);
```



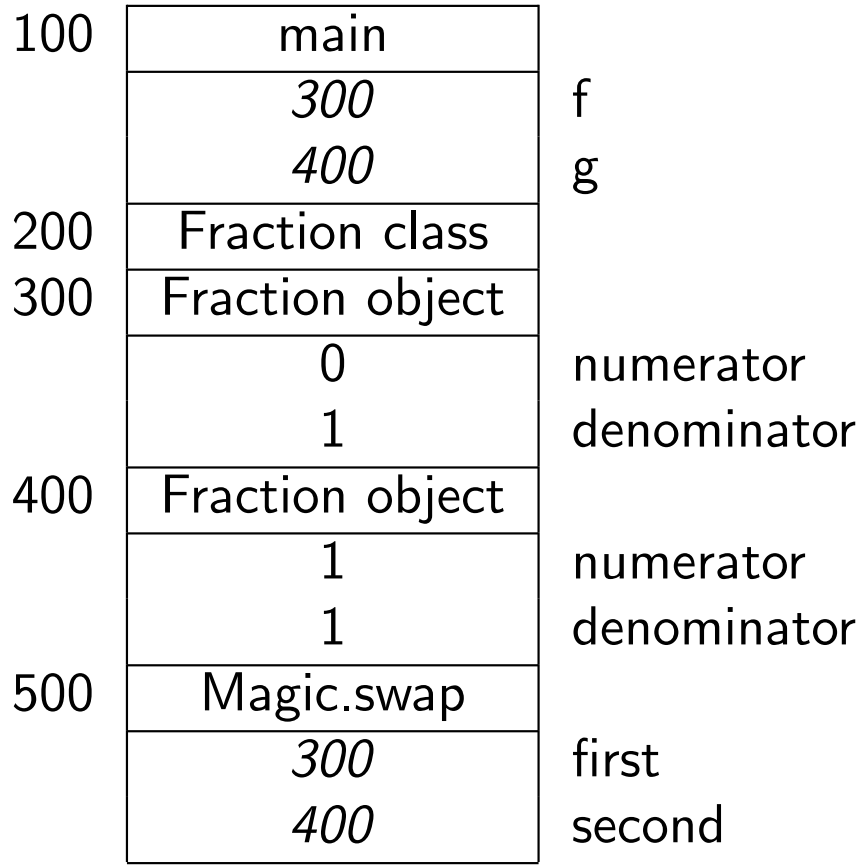
Pass-by-value

```
Fraction f = new Fraction(0, 1);
Fraction g = new Fraction(1, 1);
Magic.swap(f, g);
output.println(f);
output.println(g);
```



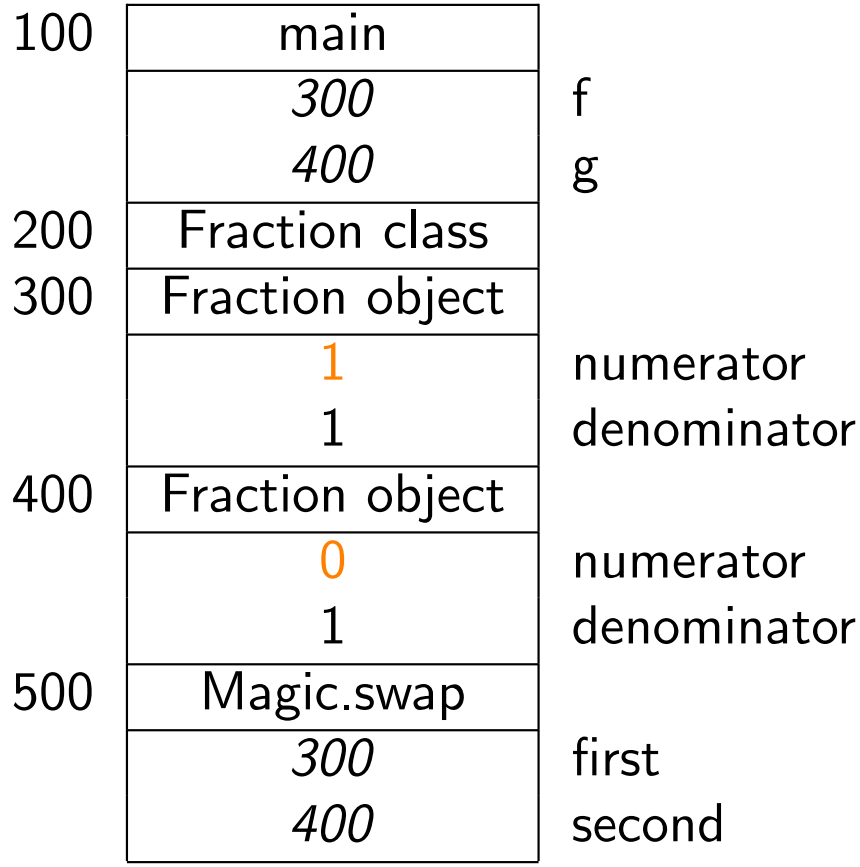
Pass-by-value

```
Fraction f = new Fraction(0, 1);
Fraction g = new Fraction(1, 1);
Magic.swap(f, g);
output.println(f);
output.println(g);
```



Pass-by-value

```
Fraction f = new Fraction(0, 1);
Fraction g = new Fraction(1, 1);
Magic.swap(f, g);
output.println(f);
output.println(g);
```



Pass-by-value

Note that

- the values of f and g are not modified (just like the values of x and y were not modified either),
- but the states of the objects to which f and g refer are modified.