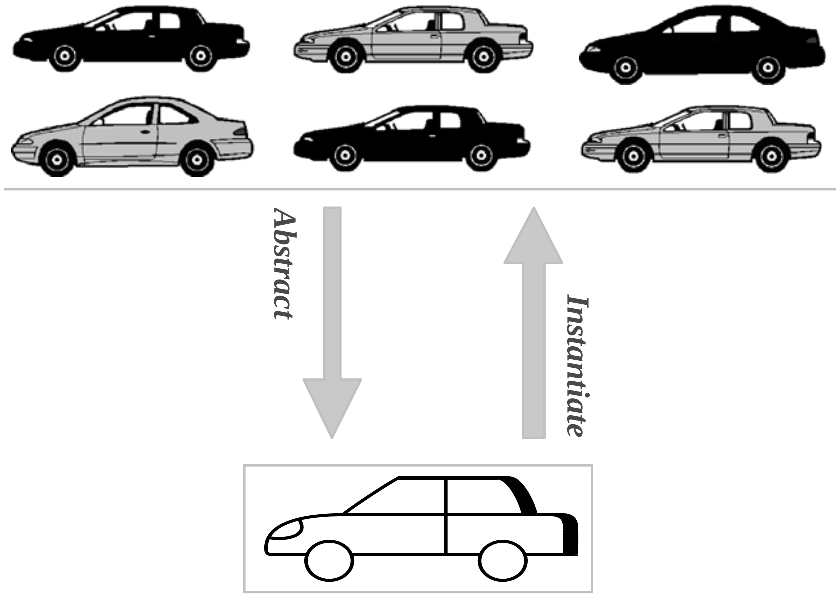
Java By Abstraction: Chapter 4

Using Objects

Some examples and/or figures were borrowed (with permission)
from slides prepared by Prof. H. Roumani

What is an Object

- An object has: attributes, methods, an identity, and a state
- A class has: attributes and methods
- Objects with the same attributes and methods can be replaced with a class that abstracts them:



Objects vs. Primitives

- Primitives
  - Contains a single value
- Objects
  - Can contain numerous attributes
  - Each attribute has its own value
  - Attributes can represent primitives or other objects

Object Reference

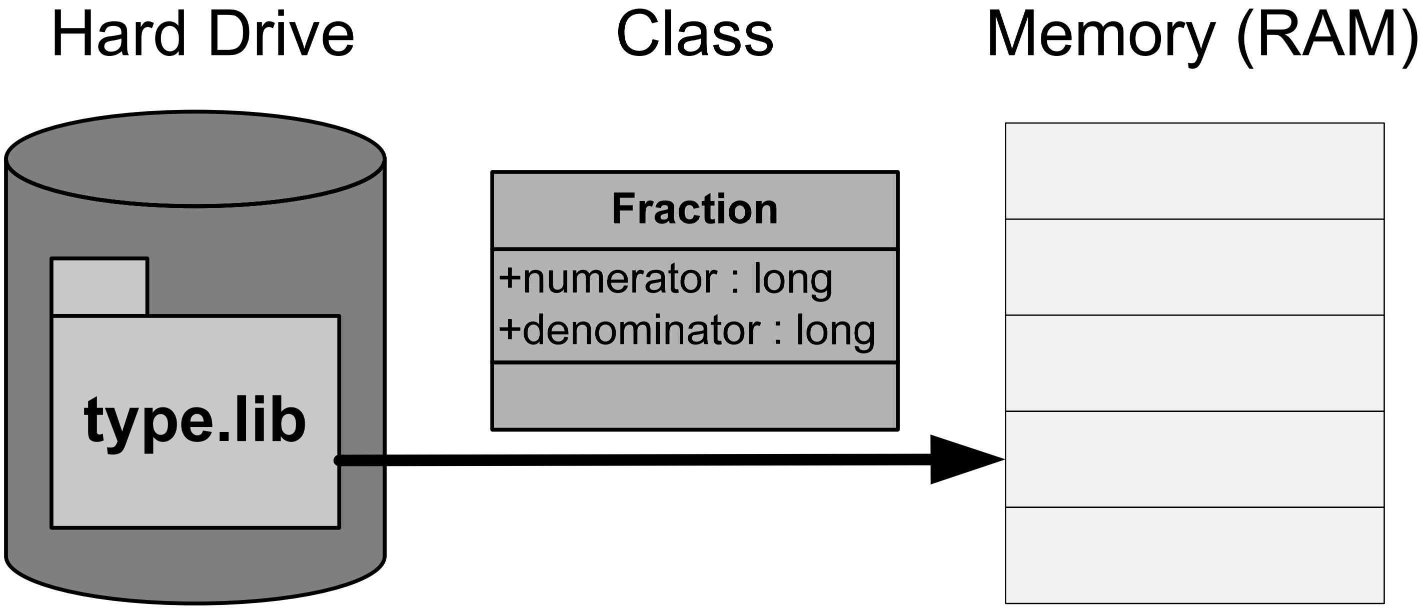
- Variables of non-primitive types are called references
- References hold the memory address of an object, but not the object itself
- Because it is a variable, a references can be changed to point to a different object in memory
- However, the memory address cannot be directly manipulated

Object Constructor

- Use the keyword **new** to instantiate an object (i.e., reserve memory for it)
- Invoke the class's **constructor** to initialize the object's state (i.e., the value of its attributes)
- Constructors look like methods, but…
  - Have no return type (not even void)
  - Have the same name as their class
- Multiple constructors could exist for a single class, providing differing initializations

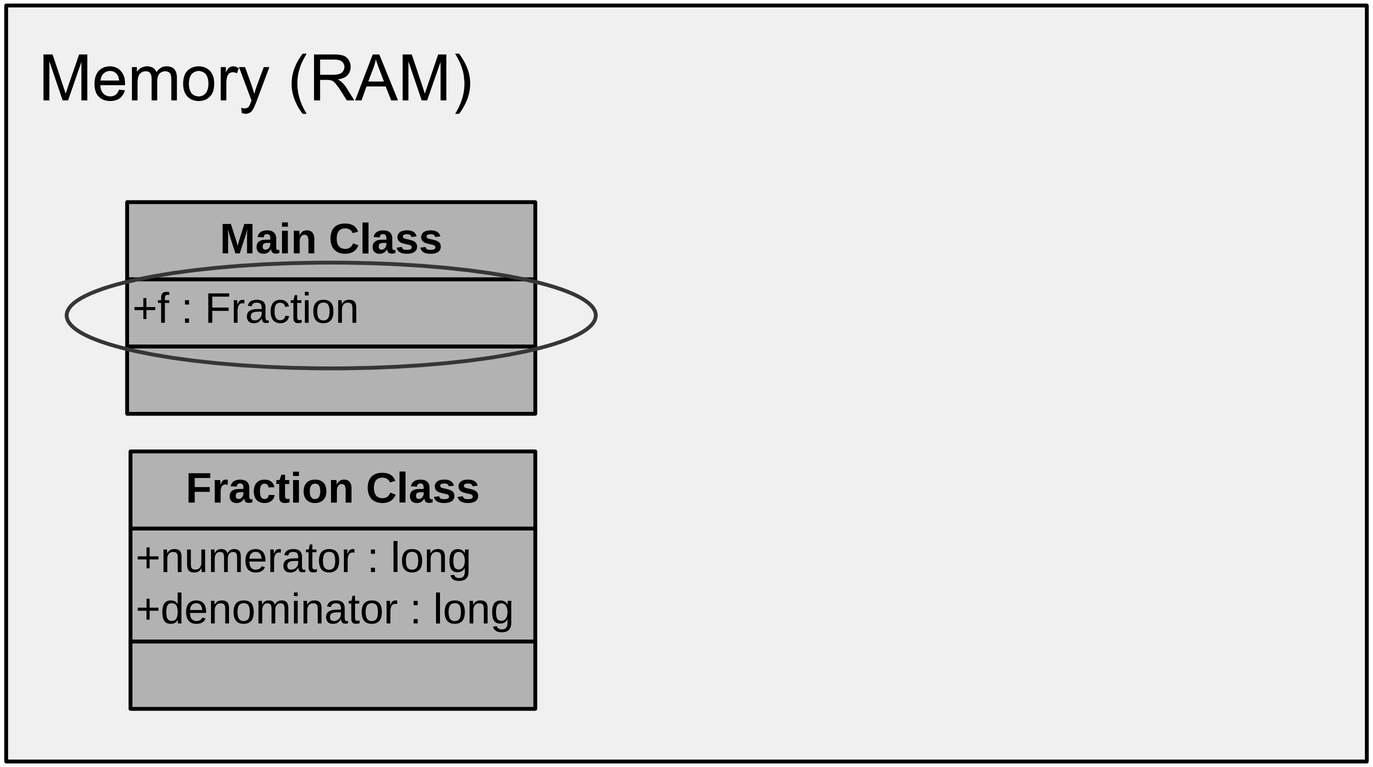
Object Creation in Memory

1. Locate the class



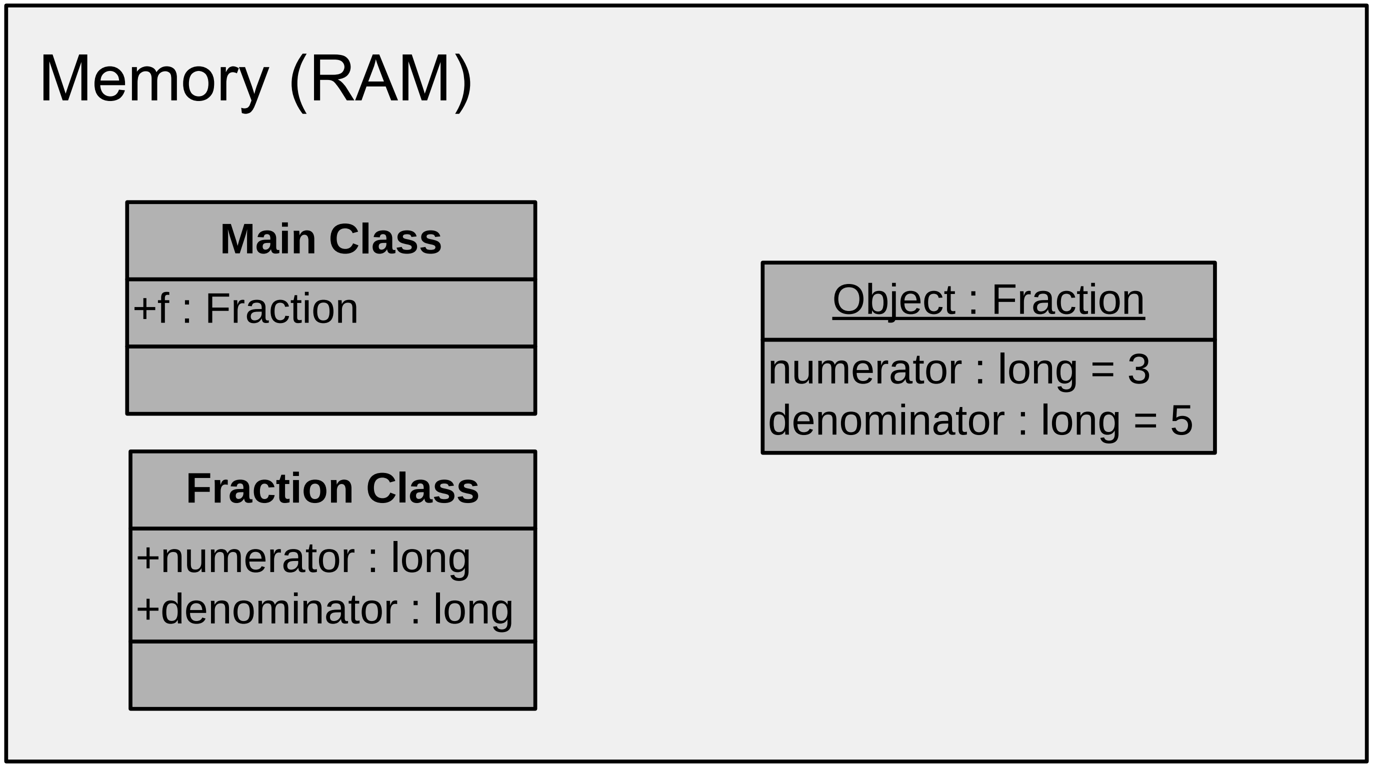
Object Creation in Memory

2. Declare a reference



Object Creation in Memory

3. Instantiate the class



Memory (RAM)

Main Class

+f : Fraction

Fraction Class

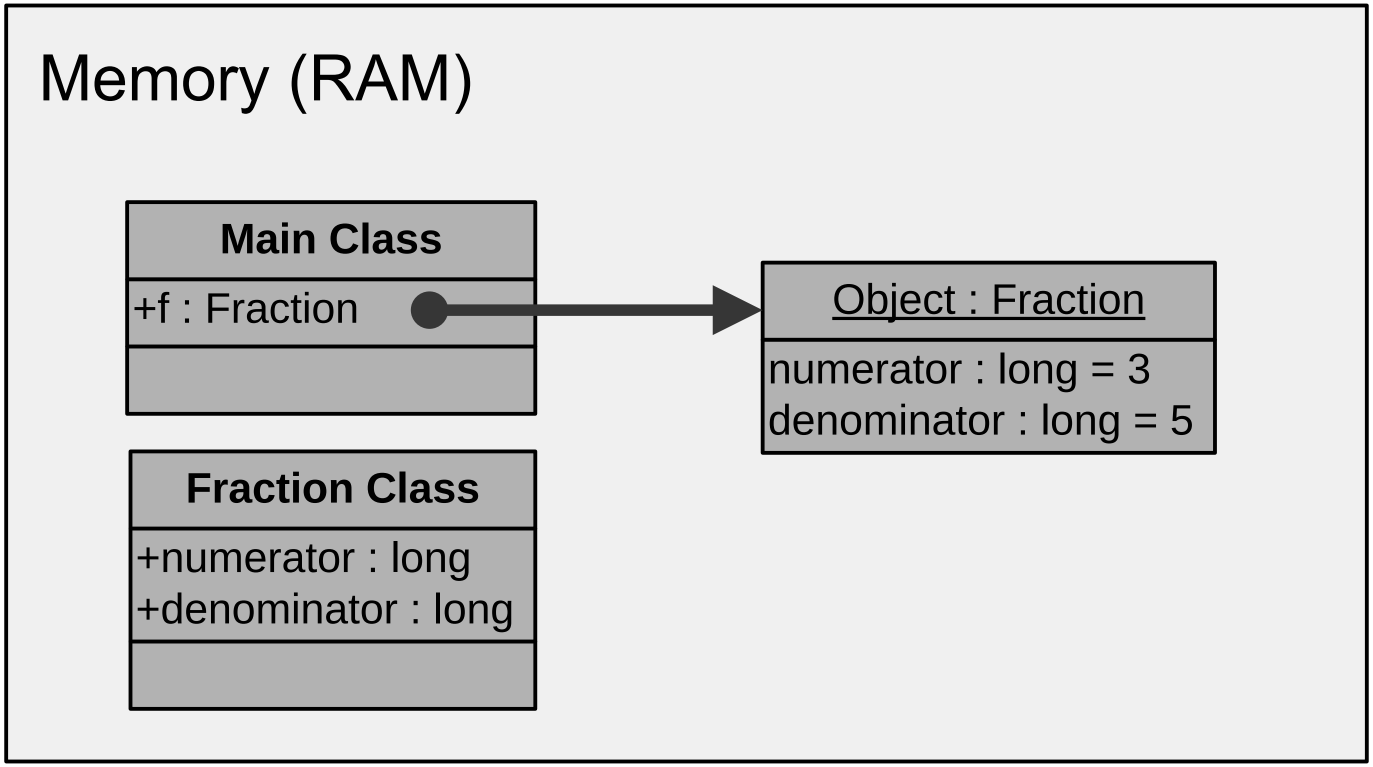
+numerator : long
+denominator : long

Object : Fraction

numerator : long = 3
denominator : long = 5

Object Creation in Memory

4. Assign a reference



Using Objects (Example)

```
…
int width = 8;
int height = 5;
Rectangle3 r = new Rectangle3();
r.width = width;
r.height = height;
int rArea = r.getArea();
System.out.println(rArea);
…
```

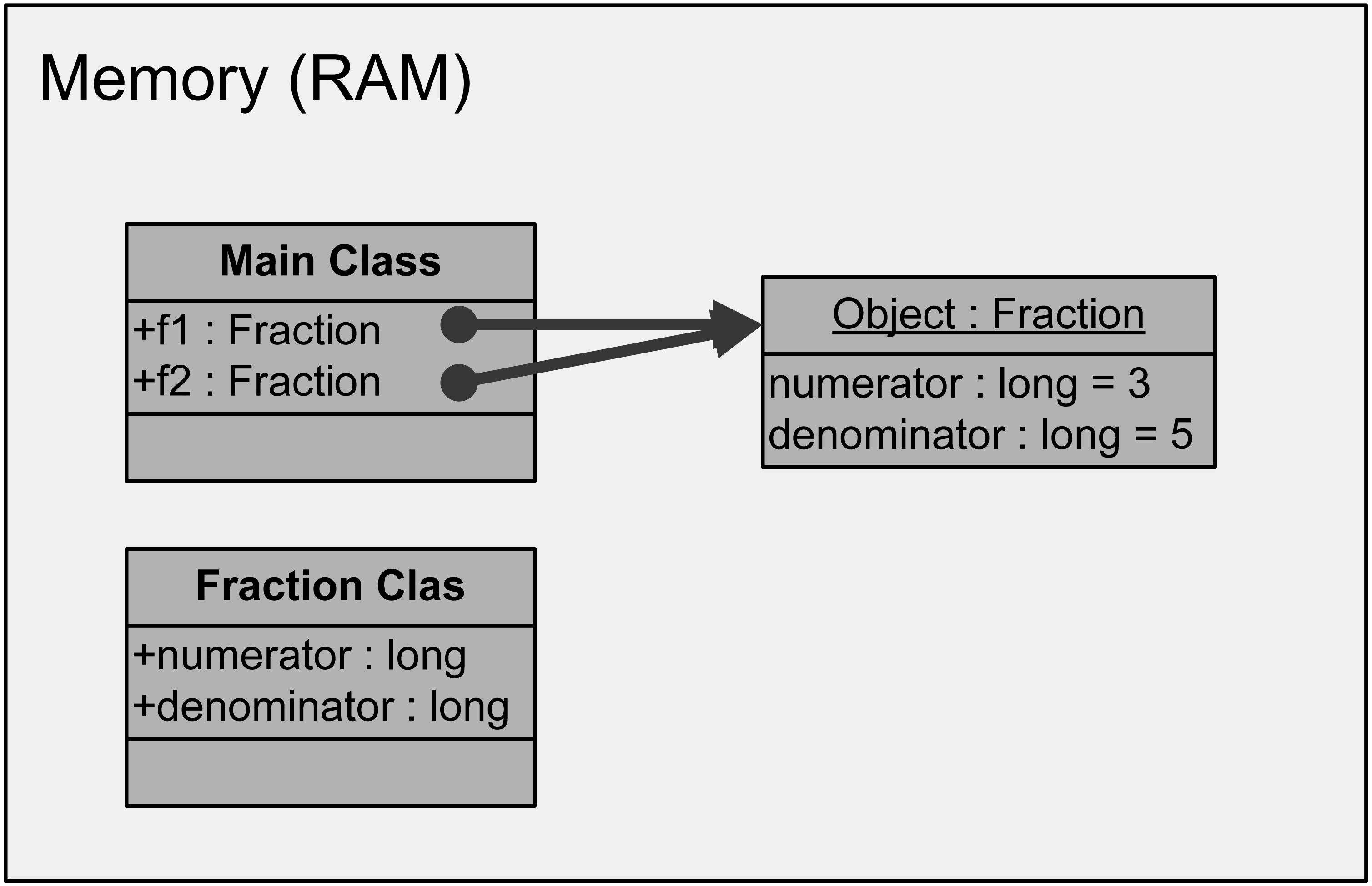
Multiple References to an Object

- A reference can only point to one object at a time
- Multiple references can point to the same object
- Example

```
Fraction f1;
f1 = new Fraction(3, 5);
Fraction f2;
f2 = f1;   // both point to the same object
```

- State changes via one reference affects the object
- Object changes are visible via any reference to it

Multiple References to an Object



Memory (RAM)

Main Class

+f1 : Fraction
+f2 : Fraction

<u>Object : Fraction</u>

numerator : long = 3
denominator : long = 5

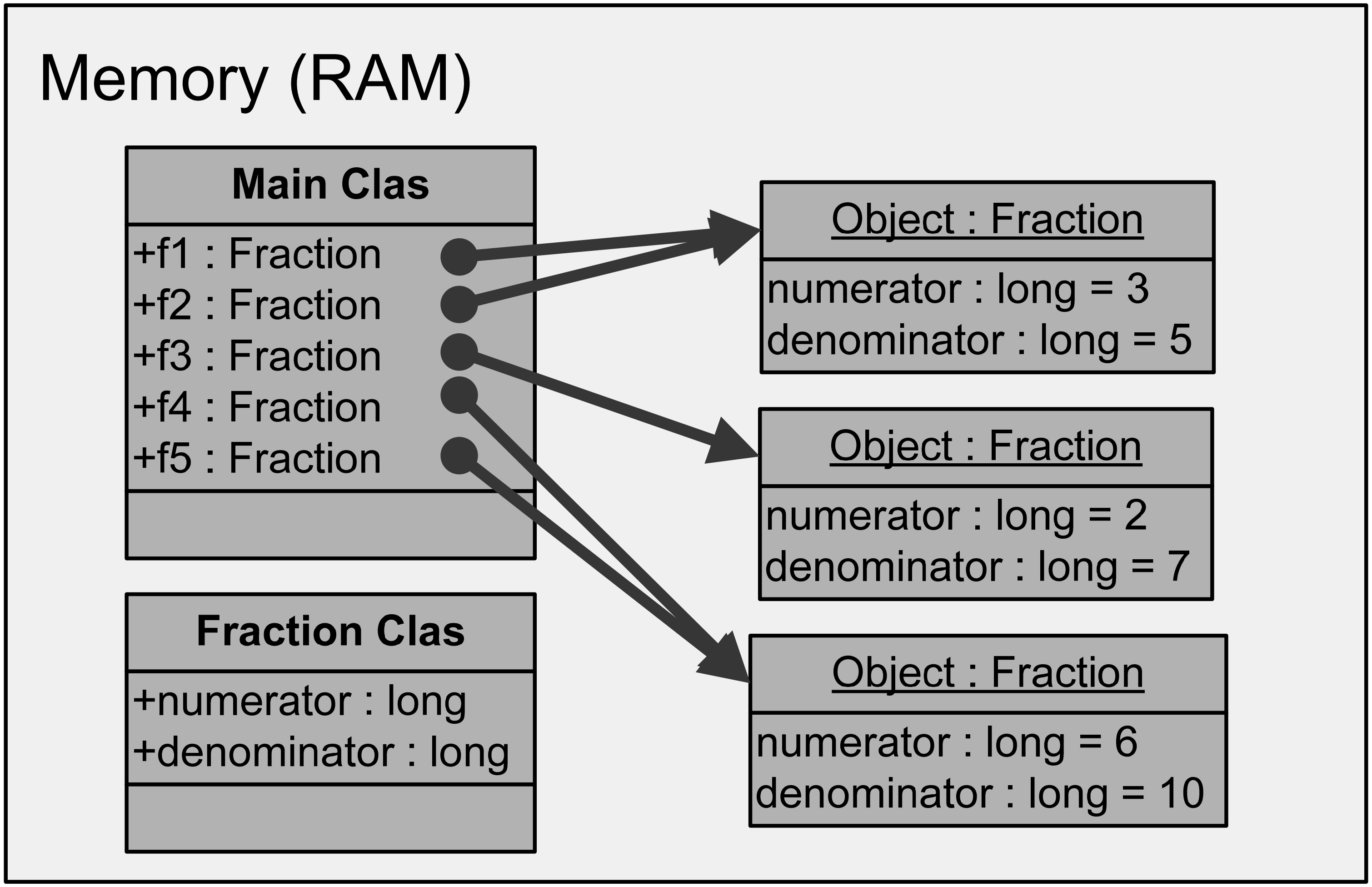
Fraction Clas

+numerator : long
+denominator : long

Object Equality

- Comparison using == operator only check memory address, not object state
- Comparison of object state requires use of the equals() method
- Example
  - *objRef1*.equals(*objRef2*);
- Definition of object equality defined by class implementer (in API)

Object Equality



Obligatory Methods

- The equals() method
  - Determines equality
  - Default: compare memory address
- The toString() method
  - Returns textual representation of the object
  - Default: object type, followed by memory address
  - Implicitly called by print methods
- Default behaviour are typically overridden by the class implementer

Accessor and Mutator Methods

- Accessor methods
  - Allow clients to determine an object’s state
  - Names typically begin with “get”
  - E.g., getNumerator(), getDenominator()
- Mutator methods
  - Allow clients to change an object’s state
  - Names typically begin with “set”
  - E.g., setFraction(long numerator, long denominator)

Attribute Privacy

- Facilitated by using accessor and mutator methods
  - Enhances encapsulation
  - Provides means to check and enforce pre-conditions and post-conditions
- Use of accessor and mutator
  - Read/write access with contracts
- Use of a accessor only
  - Read only access with contracts
- Use of a mutator only
  - Write only access with contracts

Classes with Static Features

- Stored in the class's memory region, not object's
- Changes in value affect all objects of that class
- Example:
  - Because isQuoted is static, setting it to false affects both objects

```
Fraction f = new Fraction(3, 2);
f.isQuoted = true;
Fraction g = new Fraction(5, 2);
g.isQuoted = false;
System.out.println(f.toProperString());
System.out.println(g.toProperString());
```

- Should be invoked on the class, not the object

Object Deletion (…sort of)

- In Java, the programmer cannot remove an object from memory
- Can orphan an object by removing referent to it
- Example

```
Fraction x = new Fraction(3, 5);
Fraction y = x;
y = new Fraction(4, 7);
x = null;
```

- Orphaned objects are cleared via garbage collection