Programming with Objects

COSC 122 Programming II

Rick Halterman
Valley View University
June 2012

Schedule for the Week

- Today
  - Continue looking at OOP in Java
  - *Geometric Objects* assignment due
- Tuesday
  - More Java OOP and Java graphics programming
  - *Geometric Objects* assignment deadline
- Wednesday
  - More Java graphics programming
  - Review for midterm examination
  - *Arithmetic Expression* assignment due
- Thursday
  - Midterm examination
  - *Arithmetic Expression* assignment deadline

Objects

- Objects encapsulate
  - Data (variables)
  - Code (methods)
- Objects can be quite complex
- In Java, reference variables are associated with objects

Reference Variables

- Can refer to an object
- Objects must be created with the **`new`** operator
  - This is unlike primitive types
- Objects to which no reference variables point are called garbage
  - Garbage is automatically collected by the *garbage collector*
  - The programmer has no control over the garbage collection process

Class

- The structure of an object is defined by its class
- A class is a template or pattern from which objects are created
- An object is an instance of a class
- The terms object and instance are synonymous

Fields

- A ***field*** is a class-level variable
- Fields come in two varieties:
  - Instance variables
  - Class variables

Instance vs. Class Variables

- Each instance has its own copy of instance variables
  - Changing an instance variable in one object does not affect directly the instance variables in any other object of that class
  - The storage space for the instance variables of one object is distinct from the storage space of the instance variables in another object of that class
- All instances of the same class share class variables
  - Changing a class variable in one object does affect directly the class variable with respect to all other objects of that class
  - The class itself provides the storage space for class variables, the individual instances of that class do not
- A class variable is specified with the static keyword

Methods

- A method is a unit of executable code
- Like fields, methods can be associated with instances or the class itself
  - An instance method can use the instance variables and class variables of the class
  - A class method can use only the class variables of the class
- A class method is specified with the **`static`** keyword

Schedule for the Week

- Monday
  - Continue looking at OOP in Java
  - *Geometric Objects* assignment due
- Today
  - More Java OOP and Java graphics programming
  - *Geometric Objects* assignment deadline
- Wednesday
  - More Java graphics programming
  - Review for midterm examination
  - *Arithmetic Expression* assignment due
- Thursday
  - Midterm examination
  - *Arithmetic Expression* assignment deadline

Instance vs. Class Methods

Characteristic	Class Method	Instance Method
Can access a class variable?	Yes	Yes
Can access an instance variable?	No	Yes
Can be invoked via the class?	Yes	No
Can be invoked via an object reference?	Yes	Yes
Can reference `this`?	No	Yes

Accessibility

- Fields and methods can be restricted from access by methods outside the class
- Accessibility modifiers
  - `public`—accessible by all methods in any class
  - `protected`—accessible by methods
    - in this class
    - in subclasses of this class
    - by classes in the same package as this class
  - No specifier—accessible by methods
    - in this class
    - by classes in the same package as this class
  - Sometimes called "package" access
  - `private`—accessible only by methods in this class

Accessibility Summary

For a field or method in a particular class specified as indicated:

Is accessible to a method . . .	`public`	`protected`	"package"	`private`
in the same class?	Yes	Yes	Yes	Yes
in a different class in the same package?	Yes	Yes	Yes	No
in a subclass in a different package?	Yes	Yes	No	No
in a non-subclass in a different package?	Yes	No	No	No

Example Class

```
public class Rational {
    private int numerator;
    private int denominator;
    public void setNumerator(int n) {
       numerator = n;
    }
    public void setDenominator(int d) {
        denominator = d;
    }
    public int getNumerator() {
        return numerator;
    }
    public int getDenominator() {
        return denominator;
    }
}
```

Example Client Code

```
public class RationalTest {
    public static void main(String[] args) {
        Rational fraction;
        fraction = new Rational();
        fraction.setNumerator(1);
        fraction.setDenominator(2);
        System.out.println("The fraction is "
                           + fraction.getNumerator() + "/"
                           + fraction.getDenominator());
    }
}
```

Qualified vs. Unqualified References

- The general form of a method is call made on behalf of an object is:

  objectreference.methodname(*parameterlist*)
- The use of the dot (.) operator makes this a qualified method call
- An unqualified method call does not use the dot operator but uses the method name by itself:

  methodname(*parameterlist*)
- An unqualified method call is possible only when one method calls another method within the same class.
- To invoke a method on behalf of a given object, a qualified call is required.

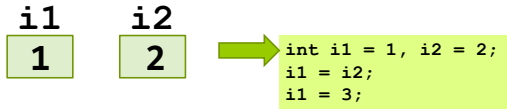
Overloading Methods

- A class can contain multiple methods with the same name
- Multiple methods with the same name are said to be ***overloaded***
- Overloaded methods must have different ***signatures***
- A method signature consists of its
  - name
  - parameter list (order and types)

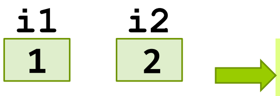
Constructors

- A constructor is called when an object is created
- If no constructor is defined, the compiler provides a default constructor that does nothing
- A constructor is distinguished from a normal method definition in two ways:
  - a constructor has the same name as the class and
  - a constructor does not have a specified return type, not even **`void`**.

Assignment of Primitive Types

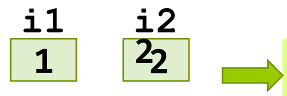


Assignment of Primitive Types



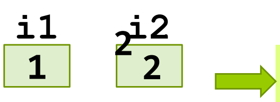

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types



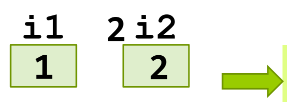

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types



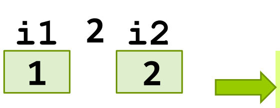

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types



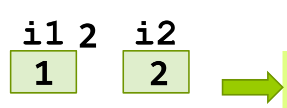

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types




```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types




```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types

i1 i2

1 2 (2)

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types

i1 i2

1 2 (2)

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types

i1 i2

2 2

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types

i1 i2

2 2

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types

i1 i2

2 2

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

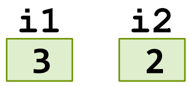
Assignment of Primitive Types

i1 i2

3 2

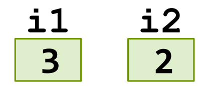
```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types



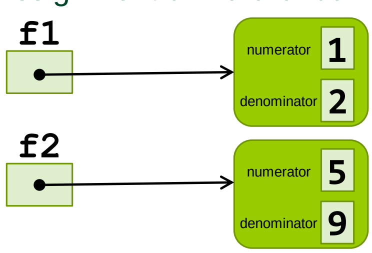

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Primitive Types



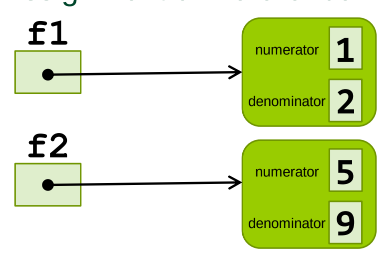

```
int i1 = 1, i2 = 2;
i1 = i2;
i1 = 3;
```

Assignment of Reference Types



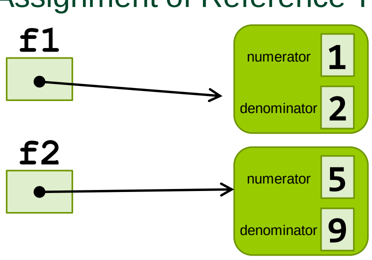

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



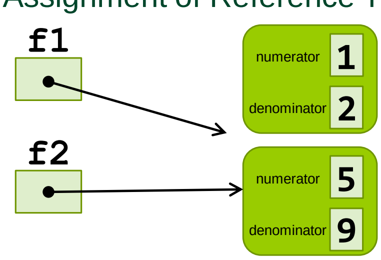

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



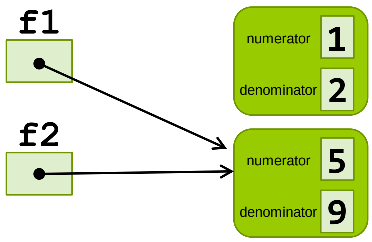

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



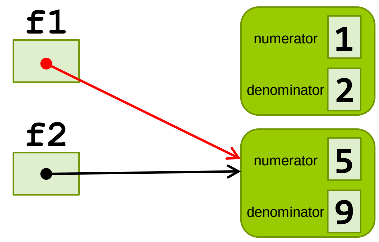

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



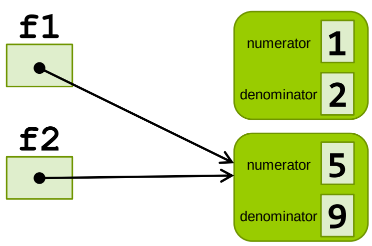

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



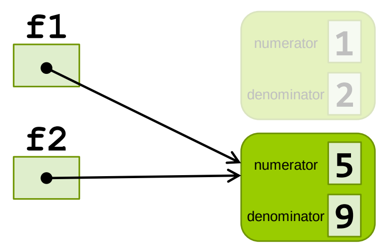

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



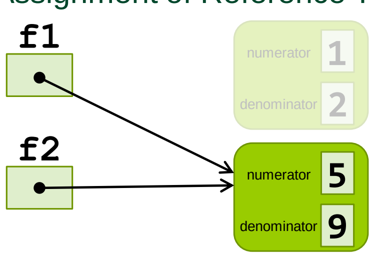

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



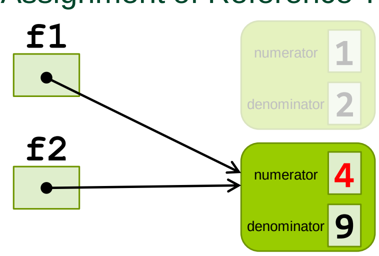

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



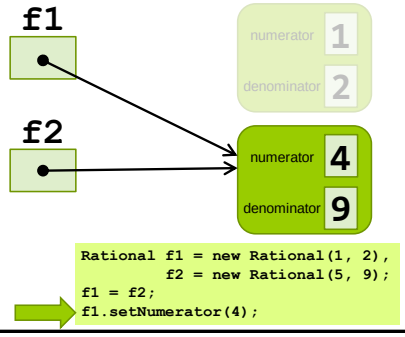

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types



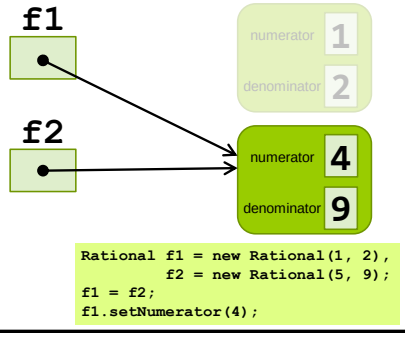

```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

Assignment of Reference Types




```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

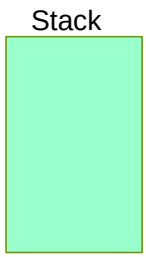
Assignment of Reference Types




```
Rational f1 = new Rational(1, 2),
         f2 = new Rational(5, 9);
f1 = f2;
f1.setNumerator(4);
```

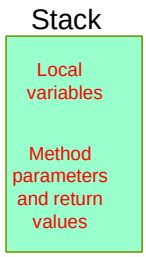
Java Run-time Memory

Java Run-time Memory

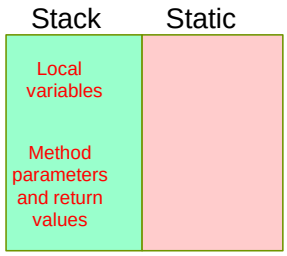


Java Run-time Memory

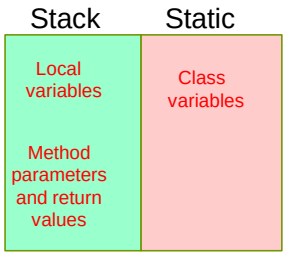
Java Run-time Memory



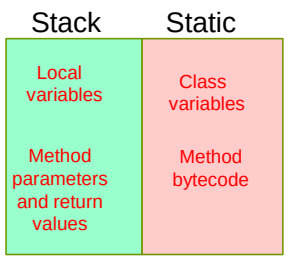
Java Run-time Memory



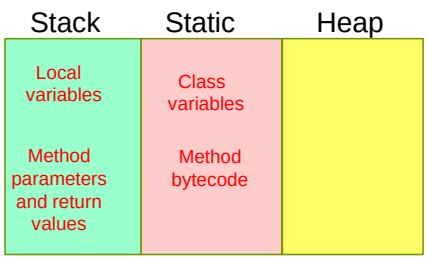
Java Run-time Memory



Java Run-time Memory



Java Run-time Memory



Java Run-time Memory