Operator Overloading

Operator Overloading

- ✓ Math operators in java: '+', '-', '*', '/', '%', '='
- ✓ Math operators are defined for **primitive** types: **int**, long, float, **double**
- ✓ Java has built-in **wrapper** classes (Objects): **Integer**, Long, Float, **Double**
  - ✓ Java supports auto-boxing: Double x = 2.3 equivalent to
    - Double x = **new** Double(2.3)
    - Double x = Double.**valueOf**(2.3) [simple factory]
  - ✓ Java supports auto-unboxing: Double x = 2.3; double y = x ;
    - y is primitive, x is object. **Compiler** does this: double y = x.**doubleValue**()
    - Double **x** = 2.3; Double **y** = 1.2;
    - This works: **double z = x + y**, but how? z is primitive, x & y are objects [auto-unboxing]
    - BUT this also works: Double z = x + y , even this works: Double z = x + 1.2

Code injection (replacement)

Operator overloading: a math operator is acting on two objects

In this case, the defined behavior is "auto-unboxing" and then "auto-boxing"

Operator Overloading

- ✓ Unfortunately, java compiler **ONLY** allows operator overloading for **built-in wrapper** classes
  - Integer, Double, Float → auto-boxing & auto-unboxing
- ✓ Java **does not allow** general operator overloading for any object
  - But why?
    - **Type safety.** Arbitrary operator overloading may lead to **runtime** exceptions.
    - OOP is for modeling real world objects. Math operation between general objects does not make sense.
- ✓ However, when dealing with **math** operator overloading makes a lot sense between math objects. Here are some examples:

Vector x, Vector y → Vector z = x + y

Vector x, Vector y → Vector z = x / y

Function x, Function y → Function z = x * y

Function x → Function y = x + 2.3

Matrix x, Matrix y → Matrix y = 2.3*x

Matrix x, Matrix y → Matrix z = x + y

Matrix x, Matrix y → Matrix z = x * y

Matrix x, Matrix y → Matrix z = x / y

Does not make sense

Operator Overloading

- ✓ How to **add operator overloading** capability to java:
  - This is IDE dependent
  - Read this paper:

Using Generics in java

    - Title: "**java modular extension for operator overloading**"
    - https://wireilla.com/papers/ijpla/V4N2/4214ijpla01.pdf
- ✓ Eclipse:
  - Step 1: Install **Scala IDE for Eclipse** plugin using the update site:
    - http://download.scala-ide.org/sdk/lithium/e47/scala212/stable/site
  - Step 2: install operator overloading support from the github
    - **https://github.com/amelentev/java-oo**
    - Update site: http://amelentev.github.io/eclipse.jdt-oo-site/
  - Restart Eclipse

Operator Overloading

✓ How to **enable** operator overloading for an object? Replace "Object" with the class

```
/**
 * Simple Factory [STATIC METHOD]
 *
 * Object a = 5;
 *
 */
static Vector valueOf(Object2 v);
static Vector valueOf(int v);
static Vector valueOf(long v);
static Vector valueOf(float v) ;
static Vector valueOf(double v);
```

```
/**
 * Subtraction
 * a-b: subtract, b-a: subtractRev
 */
Object  subtract(Object  v);
Object  subtract(int v);
Object  subtractRev(int v);
Object  subtract(long v);
Object  subtractRev(long v);
Object  subtract(float v);
Object  subtractRev(float v);
Object  subtract(double v);
Object  subtractRev(double v);
```

```
/**
 * Division
 * a/b, divideRev: b/a
 */
Object  divide(Object  v);
Object  divide(int v);
Object  divideRev(int v);
Object  divide(long v);
Object  divideRev(long v);
Object  divide(float v);
Object  divideRev(float v);
Object  divide(double v);
Object  divideRev(double v);
```

Operation: **this/v**

Reverse operation **v/this**

```
/**
 * Addition
 * a+b: add or b+a: addRev
 */
Vector add(Vector v);
Vector add(int v);
Object addRev(int v);
Object add(long v);
Object addRev(long v);
Object add(float v);
Object addRev(float v);
Object add(double v);
Object addRev(double v);
```

```
/**
 * Multiplication
 * a*b, multiplyRev: b*a
 */
Object  multiply(Object  v);
Object  multiply(int v);
Object  multiplyRev(int v);
Object  multiply(long v);
Object  multiplyRev(long v);
Object  multiply(float v);
Object  multiplyRev(float v);
Object  multiply(double v);
Object  multiplyRev(double v);
```

```
/**
 * Negate
 * -a : 0-a (binary operation)
 */
Vector negate();
```

Look at the published paper for more methods

Operator Overloading

✓ **Example**: defining a function class with operator overloading enabled

```
public interface OperatorOverloading<T> {

    // addition
    T add(double v) ;
    T addRev(double v) ;
    T add(T v) ;
    T addRev(T v) ;

    // subtraction
    T subtract(double v) ;
    T subtractRev(double v) ;
    T subtract(T v) ;
    T subtractRev(T v) ;

    // multiplication
    T multiply(double v) ;
    T multiplyRev(double v) ;
    T multiply(T v) ;
    T multiplyRev(T v) ;

    // division
    T divide(double v) ;
    T divideRev(double v) ;
    T divide(T v) ;
    T divideRev(T v) ;

    // negate
    T negate() ;

}
```

Since **factory** methods (valueOf) are **static**, implement them in the class

Using "**generics**"

Interface for operator overloading

Vector class implements interface

Example: V = (x, y)

Create a 2D Vector class

Implement operator overloading

Check the result

V/2.1 **2.1/V**

Operator Overloading

✓ **What actually happens behind the scenes in Eclipse?**

- Look at the bytecode in Eclipse

Java code

```
Vector2D v3 = v1 - v2 ;

// compiler: replaces v1-v2 with this code and generates the bytecode
v3 = v1.subtract(v2) ;
```

Bytecode

```
L4
   LINENUMBER 12 L4
   ALOAD 1
   ALOAD 2
   INVOKEVIRTUAL demo_op_ov/Vector2D.subtract (Ldemo_op_ov/Vector2D;)Ldemo_op_ov/Vector2D;
   ASTORE 3
L5
   LINENUMBER 15 L5
   ALOAD 1
   ALOAD 2
   INVOKEVIRTUAL demo_op_ov/Vector2D.subtract (Ldemo_op_ov/Vector2D;)Ldemo_op_ov/Vector2D;
   ASTORE 3
```

get the object reference for v1

get the object reference for v2

Method descriptor

Invoke this method **(method binding)**

argument type

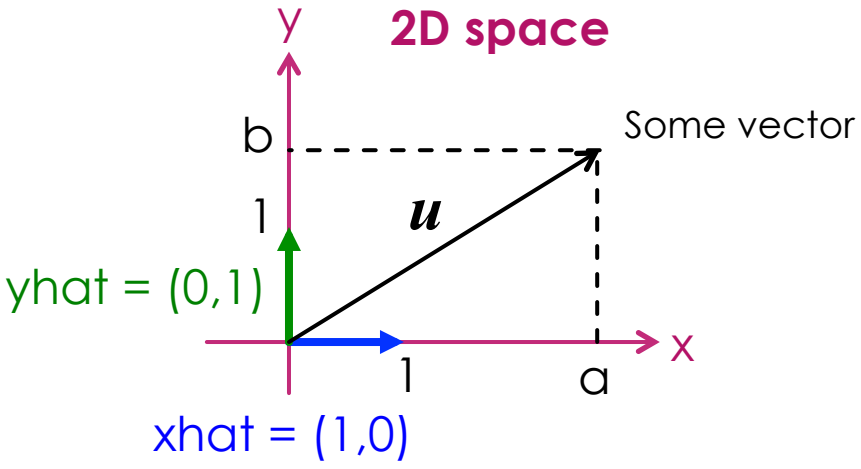
return type

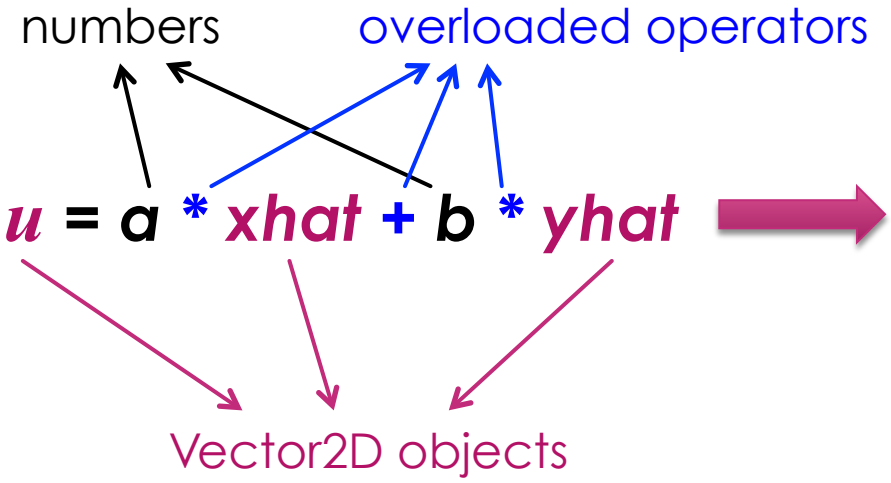
Exactly same bytecode

Operator Overloading

✓ **Defining a 2D vector space**

- Only need to define the basis of two **unit vectors**
  - **xhat** = (1, 0), **yhat** = (0, 1), uhat = u/ | u | : direction
- Use the <u>power of operator overloading</u>
- **xhat** & **yhat** → public static final Vector2D
- Expand arbitrary function in terms of xhat and yhat





Very nice (and **natural**) way of creating new vectors