Object-oriented programming

Second semester

Lecture №9

Groovy, DSL.

What is wrong with Java?

- Java is unnecessarily verbose. Anyone who has ever tried to read from or write to a disk file in Java (two very common tasks) knows that such a simple job takes at least ten lines of code
- Operators in Java (such as +, *, and -) can operate on primitive types only and not on objects (with the exception of String concatenation using the + operator. This can cause confusion to newcomers to the language and makes working with collections (which are essential in any language) unnecessarily painful
- Java has no language-level support for collections (that is, it has no literal declaration for collections such as lists or maps, as it has for arrays)

a(b).c(d) -> a b c d

```
show = { println it }
square_root = { Math.sqrt(it) }

def please(action) {
  [the: { what ->
    [of: { n -> action(what(n)) }]
  }]
}

// equivalent to: please(show).the(square_root).of(100)
please show the square_root of 100
// ==> 10.0
```

```
// equivalent to: turn(left).then(right)
turn left then right

// equivalent to: take(2.pills).of(chloroquinine).after(6.hours)
take 2.pills of chloroquinine after 6.hours

// equivalent to: paint(wall).with(red, green).and(yellow)
paint wall with red, green and yellow

// with named parameters too
// equivalent to: check(that: margarita).tastes(good)
check that: margarita tastes good

// with closures as parameters
// equivalent to: given({}).when({}).then({})
given { } when { } then { }

// equivalent to: select(all).unique().from(names)
select all unique() from names

// equivalent to: take(3).cookies
// and also this: take(3).getCookies()
take 3 cookies
```

Example DSL

```
import com.google.common.base.*
def result = Splitter.on(',').trimResults(CharMatcher.is('_' as char)).split("_a ,_b_ ,c__").iterator().toList()
```

```
import com.google.common.base.*
def split(string) {
  [on: { sep ->
    [trimming: { trimChar ->
      Splitter.on(sep).trimResults(CharMatcher.is(trimChar as char)).split(string).iterator().toList()
    }]
```

```
 def result = split "_a ,_b_ ,c__" on ',' trimming '_\'
```

Operator overloading

a + b a.plus(b)
a – b a.minus(b)
a * b a.multiply(b)
a ** b a.power(b)
a / b a.div(b)
a % b a.mod(b)
a | b a.or(b)
a & b a.and(b)
a ^ b a.xor(b)
a++ or ++a a.next()
a-- or –a a.previous()
a[b] a.getAt(b)
a[b] = c a.putAt(b, c)
a << b a.leftShift(b)
a >> b a.rightShift(b)
a >>> b a.rightShiftUnsigned(b)
switch(a) { case(b) : } b.isCase(a)

if(a) a.asBoolean()
~a a.bitwiseNegate()
-a a.negative()
+a a.positive()
a as b a.asType(b)
a == b a.equals(b)
a != b !a.equals(b)
a <=> b a.compareTo(b)
a > b a.compareTo(b) > 0
a >= b a.compareTo(b) >= 0
a < b a.compareTo(b) < 0
a <= b a.compareTo(b) <= 0

GroovyShell

```
def binding = new Binding()
def shell = new GroovyShell(binding)
binding.setVariable('x',1)
binding.setVariable('y',3)
shell.evaluate 'z=2*x+y'
assert binding.getVariable('z') == 5
```

The Script class

```
abstract class MyBaseClass extends Script {
    String name
    public void greet() { println "Hello, $name!" }
}
```

```
def config = new CompilerConfiguration()
config.scriptBaseClass = 'MyBaseClass'
def shell = new GroovyShell(this.class.classLoader, config)
shell.evaluate """
    setName 'Judith'
    greet()
"""
```