

Object-oriented programming

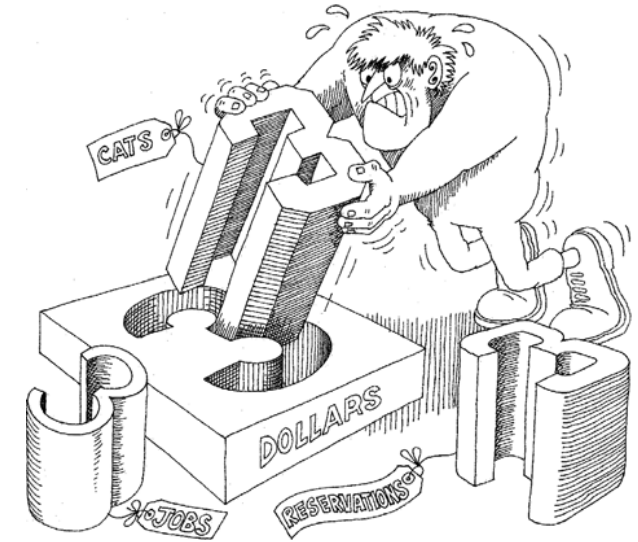
Second semester

Lecture №6

Groovy, first steps

Type system

class=type
static / dynamic typing
strong / weak typing
explicit / implicit typing



"Now! *That* should clear up
a few things around here!"

```
double a=3.6; //explicit typing
var b=a;       //implicit typing, java 10
int c=3.7f;    // weak typing, java is strong typing
```

// static typing	//dynamic typing
int add(int x,int y){	def add(x,y):
return x+y	return x+y
}	

Typing is the enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways.

What Is Groovy?

- Open Source
- Agile Dynamic Language
 - Others...
 - JavaScript
 - Ruby
 - Python
- Integrates Very Well With Java
 - Runs On The JVM
 - Call Groovy From Java
 - Call Java From Groovy
 - Leverage Powerful Existing Java Libraries

http://groovy-lang.org/

- Learn
- Documentation
- Download
-



A multi-faceted language for the Java platform

Apache Groovy is a **powerful, optionally typed and dynamic** language, with **static-typing and static compilation** capabilities, for the Java platform aimed at improving developer productivity thanks to a concise, **familiar and easy to learn syntax**. It integrates smoothly with any Java program, and immediately delivers to your application powerful features, including scripting capabilities, **Domain-Specific Language** authoring, runtime and compile-time **meta-programming** and **functional programming**.



Flat learning curve

Concise, readable and expressive syntax, easy to learn for Java developers



Smooth Java integration

Seamlessly and transparently integrates and interoperates with Java and any third-party libraries



Vibrant and rich ecosystem

Web development, reactive applications, concurrency / asynchronous / parallelism library, test frameworks, build tools, code analysis, GUI building



Powerful features

Closures, builders, runtime & compile-time meta-programming, functional programming, type inference, and static compilation



Domain-Specific Languages

Flexible & malleable syntax, advanced integration & customization mechanisms, to integrate readable business rules in your applications



Scripting and testing glue

Great for writing concise and maintainable tests, and for all your build and automation tasks

Installing Groovy

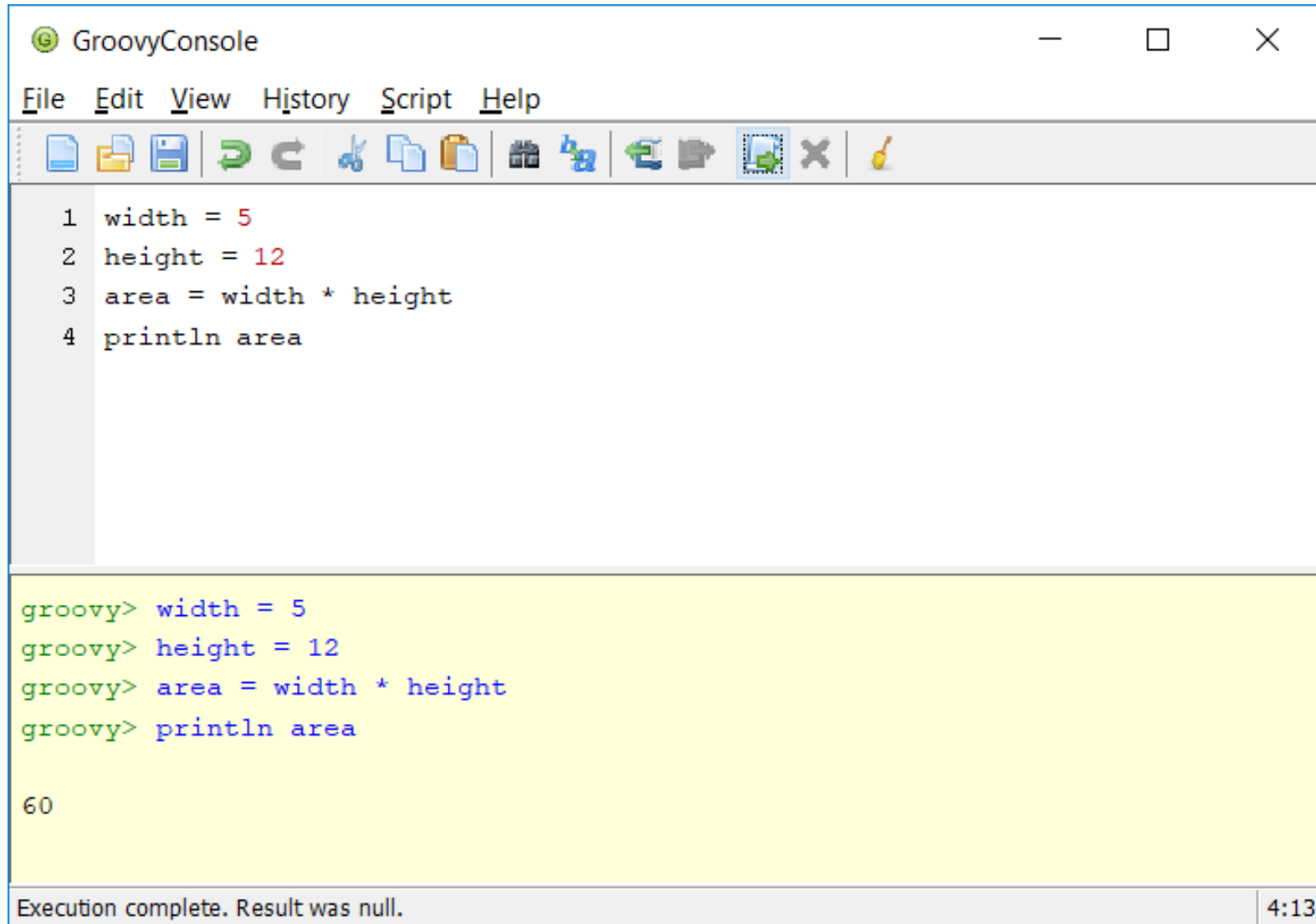
- Download Release
- Extract The Archive
- Set GROOVY_HOME
- Add %GROOVY_HOME%/bin to %PATH%

```
groovy e "a=10;b=4;c=a*b;println c"  
40
```

groovysh

- C:\bin\groovy-3.0.2\bin>groovysh.bat
- Groovy Shell (3.0.2, JVM: 12.0.2)
- Type ':help' or ':h' for help.
- -----
- groovy:000> width = 5
- ===> 5
- groovy:000> height = 12
- ===> 12
- groovy:000> area = width * height
- ===> 60
- groovy:000> println area
- 60
- ===> null

groovyConsole



The screenshot shows the GroovyConsole application window. The title bar reads "GroovyConsole". The menu bar includes "File", "Edit", "View", "History", "Script", and "Help". The toolbar contains icons for file operations (new, open, save, save as), execution (run, stop, refresh), and editing (undo, redo, copy, paste, delete). The main text area contains a Groovy script with four lines: `width = 5`, `height = 12`, `area = width * height`, and `println area`. The output area, which has a yellow background, shows the same four lines of code being executed, followed by the output `60`. The status bar at the bottom indicates "Execution complete. Result was null." and the time "4:13".

```
1 width = 5
2 height = 12
3 area = width * height
4 println area
```

```
groovy> width = 5
groovy> height = 12
groovy> area = width * height
groovy> println area

60
```

Execution complete. Result was null. 4:13

Groovy Scripts

```
// mygroovyscript.groovy
```

```
width = 5
```

```
height = 12
```

```
area = width * height
```

```
println area
```

```
groovy mygroovyscript.groovy
```

```
60
```


groovyc

groovyc Compiles Groovy To Bytecode

- Compiled Code Runs As Normal Java Code
- CLASSPATH
 - groovyall[version].jar
 - in \$GROOVY_HOME/embeddable/

Step 1

Java vs Groovy

Code is money

```
public class Hello {  
    String name;  
  
    public void sayHello() {  
        System.out.println("Hello "+getName()+"!");  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public static void main(String[] args) {  
  
        Hello hello = new Hello();  
        hello.setName("world");  
        hello.sayHello();  
    }  
}
```

Step 2

one difference is that things
are public by default

```
class Hello {  
  
    String name;  
  
    void sayHello() {  
        System.out.println("Hello "+getName()+"!");  
    }  
  
    void setName(String name) {  
        this.name = name;  
    }  
  
    String getName() {  
        return name;  
    }  
  
    static void main(String[] args) {  
  
        Hello hello = new Hello();  
        hello.setName("world");  
        hello.sayHello();  
    }  
}
```

Step 3

Accessors and mutators
are automatically created
for your class variables

```
class Hello {  
  
    String name;  
  
    void sayHello() {  
        System.out.println("Hello "+getName()+"!");  
    }  
  
    static void main(String[] args) {  
  
        Hello hello = new Hello();  
        hello.setName("world");  
        hello.sayHello();  
    }  
}
```

Step 4

System.out.println can be shortened to just println as a convenience

```
class Hello {  
  
    String name;  
  
    void sayHello() {  
        println("Hello "+getName()+"!");  
    }  
  
    static void main(String[] args) {  
  
        Hello hello = new Hello();  
        hello.setName("world");  
        hello.sayHello();  
    }  
}
```

Step 5

There's also a difference in how Groovy deals with String objects—using double quotation marks with strings allows for variable substitution

```
class Hello {  
  
    String name;  
  
    void sayHello() {  
        println("Hello $name!");  
    }  
  
    static void main(String[] args) {  
  
        Hello hello = new Hello();  
        hello.setName('world');  
        hello.sayHello();  
    }  
}
```

Step 6

Groovy also allows dot notation for getting and setting fields of classes

```
class Hello {  
  
    String name;  
  
    void sayHello() {  
        println("Hello $name!");  
    }  
  
    static void main(String[] args) {  
  
        Hello hello = new Hello();  
        hello.name = 'world';  
        hello.sayHello();  
    }  
}
```

Step 7

Typing information is also optional; instead of specifying a type, you can just use the keyword `def`. Use of `def` is mandatory for methods, but it is optional for parameters on those methods. You should also use `def` for class and method variables. While we're at it, let's take out those semicolons; they're optional in Groovy.

```
class Hello {  
  
    def sayHello(name) {  
        println("Hello $name!")  
    }  
  
    static def main(args) {  
        Hello hello = new Hello()  
        def name = 'world'  
        hello.sayHello(name)  
    }  
}
```


Step 8

Groovy is a scripting language, there's automatically a wrapping class. This wrapping class means that we can get rid of our own wrapping class, as well as the main method

```
def sayHello(name) {  
    println("Hello $name!")  
}  
def name = 'world'  
sayHello(name)
```

Step 9

Groovy is (invisibly) creating an ArrayList

```
def sayHello(name) {  
    println("Hello $name!")  
}  
  
def names = ["Masha", "Sasha", "Koly"]  
  
names += 'Natasha' names -= 'Koly'  
  
names.sort()  
  
for (name in names) {  
    sayHello(name)  
}
```

Step 10

the name -> preamble
defines a single parameter
that the closure takes

```
def sayHello(name) {  
    println("Hello $name!")  
}
```

```
def names = ["Masha", "Sasha", "Koly"]
```

```
names += 'Natasha' names -= 'Koly'  
names.sort()
```

```
def clos = {name -> sayHello(name)}  
names.each(clos)
```

Step 11

By default, the first parameter of a closure is named it.

```
def sayHello(name) {  
    println("Hello $name!")  
}
```

```
def names = ["Masha", "Sasha", "Koly"]
```

```
names += 'Natasha' names -= 'Koly'  
names.sort()
```

```
names.each {sayHello(it)}
```