# Object-oriented programming

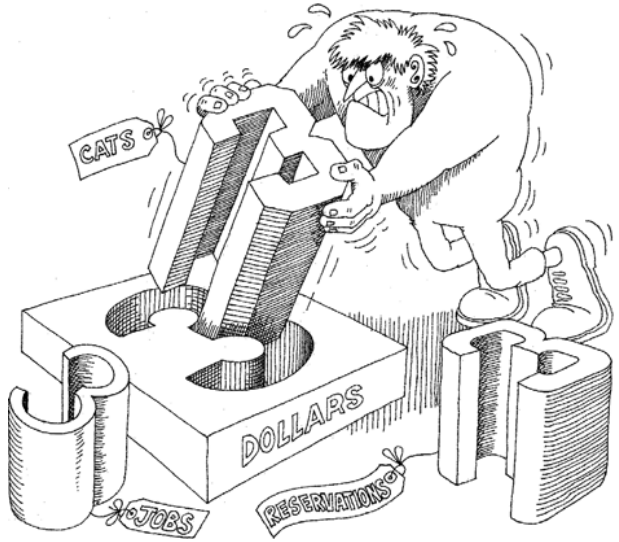
Second semester

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
}
```

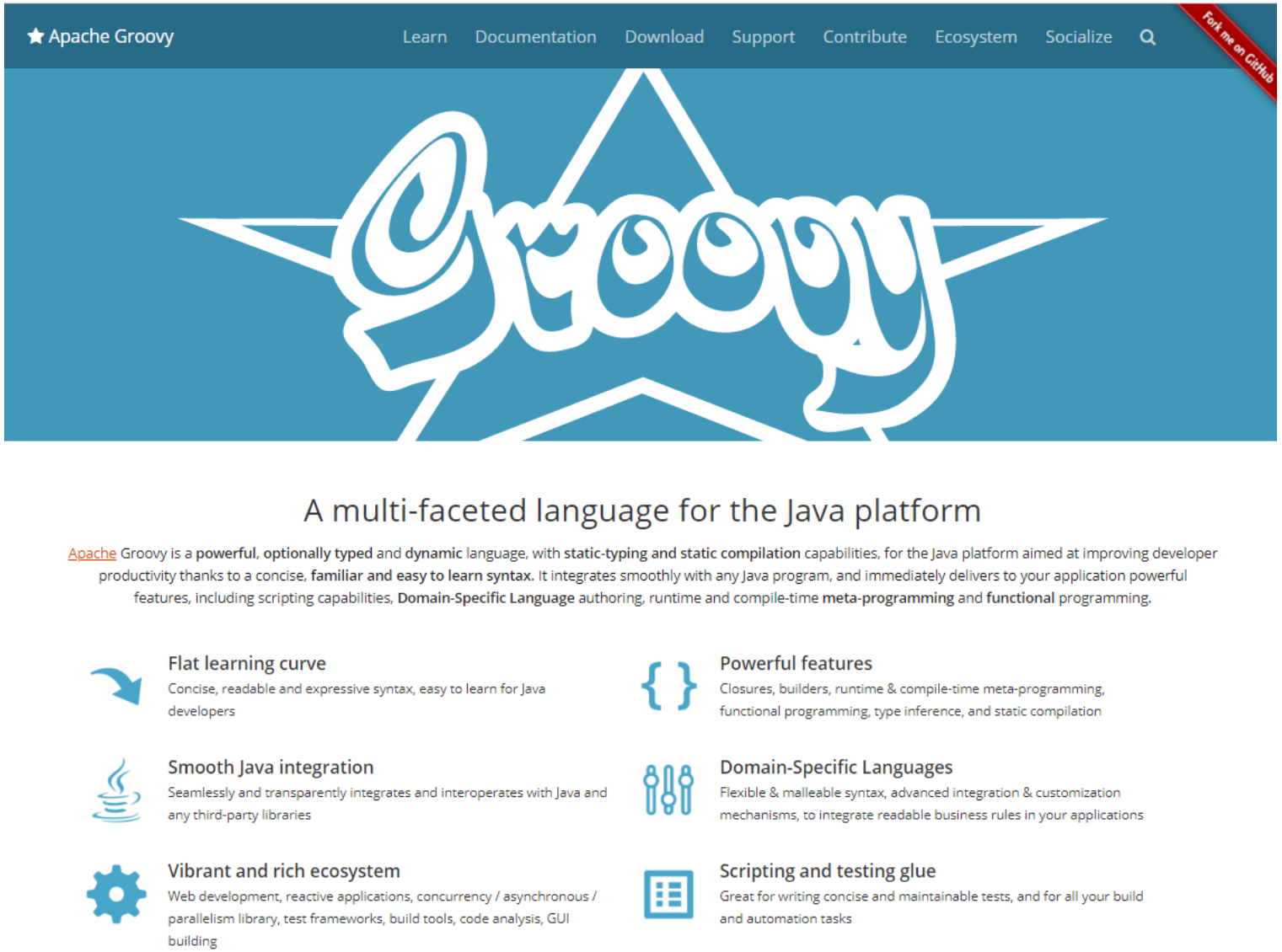
**Typing is the enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways.**

# 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
- ....



# Installing Groovy

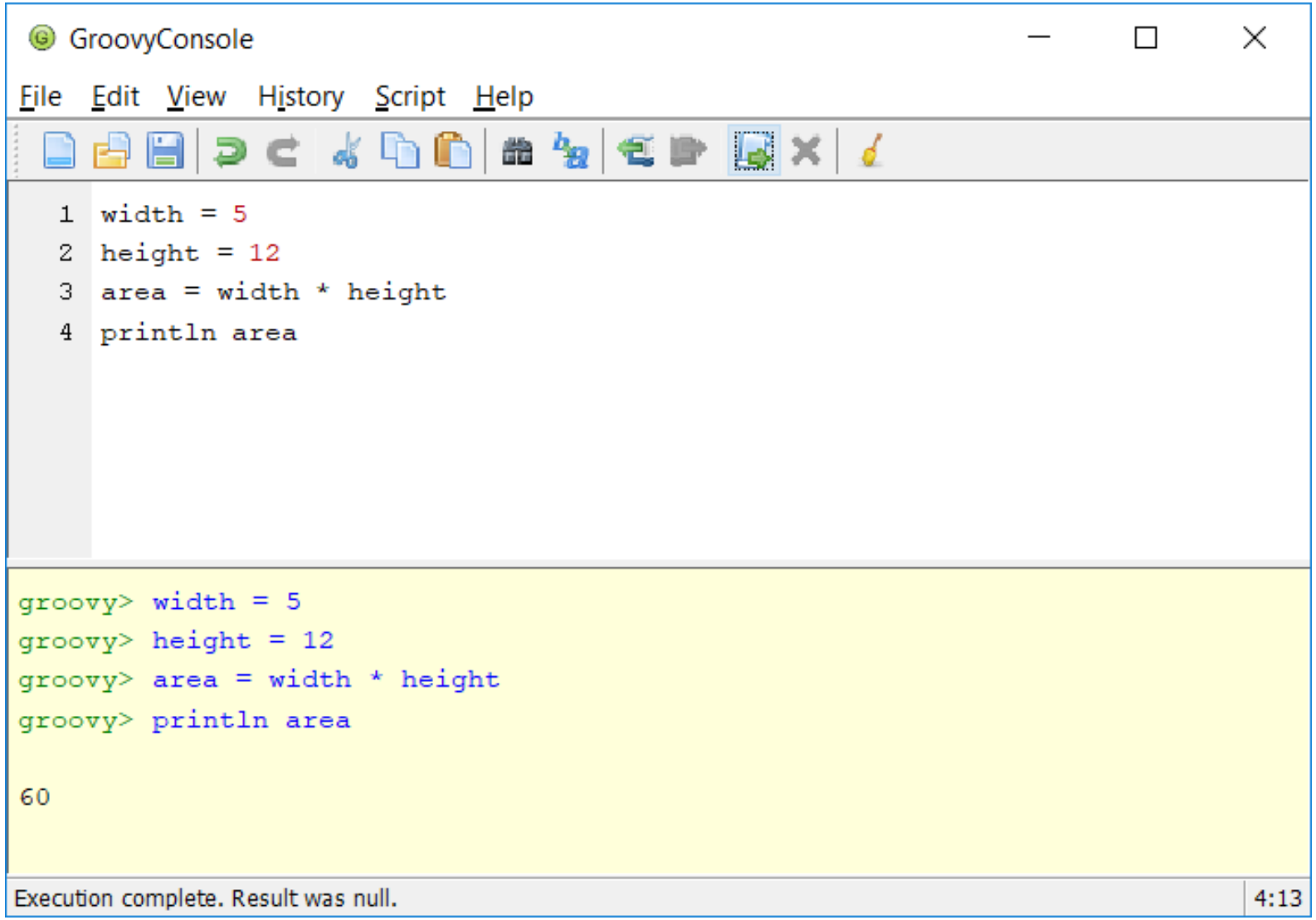
- Download Release
- Extract The Archive
- Set GROOVY_HOME
- Add %GROOVY_HOME%/bin to %PATH%

```
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



# 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)}
```