Object-oriented programming

Second semester

Lecture №7

Groovy

Default imports

- java.io.*
- java.lang.*
- java.math.BigDecimal
- java.math.BigInteger
- java.net.*
- java.util.*
- groovy.lang.*
- groovy.util.*

Multi-methods

```
int method(String arg) {
    return 1;
}
int method(Object arg) {
    return 2;
}
Object o = "Object";
int result = method(o);
```

- Java (static)
  - assertEquals(2, result);
- Groovy (dynamic)
  - assertEquals(1, result);

Array initializers

```
//Java
int[] array = {1, 2, 3};              // Java array initializer shorthand syntax
int[] array2 = new int[] {4, 5, 6};  // Java array initializer long syntax


// Groovy
int[] array = [1, 2, 3] // In Groovy, the { … } block is reserved for closures
// Groovy 3.0+ supports the Java-style array initialization long syntax
def array2 = new int[] {1, 2, 3}
```

Package scope visibility

```
class Person {
    // In Groovy, omitting a modifier on a field doesn’t result in a package-private field like in Java
    String name
}
```

```
class Person {
    // In Groovy, omitting a modifier on a field doesn’t result in a package-private field like in Java
    @PackageScope String name
}
```

Public field vs Property

- Encapsulation
- Object Integrity

```
public class Circle {       // class name
  private double radius;          // attributes
  private String color;

  public double getRadius() { ..... }  // methods
  public double setRadius() { ..... }
  public double getArea() { ..... }
  public double setArea() { ..... }
}
```

```
public class Circle {
  public double radius;
  public String color;
}
```

```
public class SpecialCircle extends Circle {
  public double getRadius() { ..... }
  public double setRadius() { ..... }
  public double getArea() { ..... }
  public double setArea() { ..... }
}
```

get/set Property

```
class POGO {

   String property

   void setProperty(String name, Object value) {
      this.@"$name" = 'overridden'
   }
}

def pogo = new POGO()
pogo.property = 'a'

assert pogo.property == 'overridden'
```

```
class SomeGroovyClass {

   def property1 = 'ha'
   def field2 = 'ho'
   def field4 = 'hu'

   def getField1() {
      return 'getHa'
   }

   def getProperty(String name) {
      if (name != 'field3')
         return metaClass.getProperty(this, name)
      else
         return 'field3'
   }
}

def someGroovyClass = new SomeGroovyClass()

assert someGroovyClass.field1 == 'getHa'
assert someGroovyClass.field2 == 'ho'
assert someGroovyClass.field3 == 'field3'
assert someGroovyClass.field4 == 'hu'
```

get/setAttribute

```
class SomeGroovyClass {

   def field1 = 'ha'
   def field2 = 'ho'

   def getField1() {
      return 'getHa'
   }
}

def someGroovyClass = new SomeGroovyClass()

assert someGroovyClass.metaClass.getAttribute(someGroovyClass, 'field1') == 'ha'
assert someGroovyClass.metaClass.getAttribute(someGroovyClass, 'field2') == 'ho'
```

```
class POGO {

   private String field
   String property1

   void setProperty1(String property1) {
      this.property1 = "setProperty1"
   }
}

def pogo = new POGO()
pogo.metaClass.setAttribute(pogo, 'field', 'ha')
pogo.metaClass.setAttribute(pogo, 'property1', 'ho')

assert pogo.field == 'ha'
assert pogo.property1 == 'ho'
```

Automatic Resource Management(ARM) blocks (Java)

```
Path file = Paths.get("/path/to/file");
Charset charset = Charset.forName("UTF-8");
try (BufferedReader reader = Files.newBufferedReader(file, charset)) {
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }

} catch (IOException e) {
    e.printStackTrace();
}
```

ARM blocks (Groovy)

```
new File('/path/to/file').eachLine('UTF-8') {
  println it
}
```

or

```
new File('/path/to/file').withReader('UTF-8') { reader ->
  reader.eachLine {
    println it
  }
}
```

Inner classes

```
class A {
   static class B {}
}

new A.B()
```

```
import java.util.concurrent.CountDownLatch
import java.util.concurrent.TimeUnit

CountDownLatch called = new CountDownLatch(1)

Timer timer = new Timer()
timer.schedule(new TimerTask() {
   void run() {
      called.countDown()
   }
}, 0)

assert called.await(10, TimeUnit.SECONDS)
```

```
//Java
public class Y {
   public class X {}
   public X foo() {
      return new X();
   }
   public static X createX(Y y) {
      return y.new X();
   }
}

//Groovy
public class Y {
   public class X {}
   public X foo() {
      return new X()
   }
   public static X createX(Y y) {
      return new X(y)
   }
}
```

Lambda expressions and the method reference operator

```
// Java
Runnable run = () -> System.out.println("Run");
list.forEach(System.out::println);

// Groovy
Runnable run = { println 'run' }
list.each { println it } // or list.each(this.&println)
```

GStrings

- As double-quoted string literals are interpreted as GString values, Groovy may fail with compile error or produce subtly different code if a class with String literal containing a dollar character is compiled with Groovy and Java compiler.

- While typically, Groovy will auto-cast between GString and String if an API declares the type of a parameter, beware of Java APIs that accept an Object parameter and then check the actual type.

String and Character literals

```
assert 'c'.getClass()==String
assert "c".getClass()==String
assert "c${1}".getClass() in GString



// for single char strings, both are the same
assert ((char) "c").class==Character
assert ("c" as char).class==Character

// for multi char strings they are not
try {
  ((char) 'cx') == 'c'
  assert false: 'will fail - not castable'
} catch(GroovyCastException e) {
}
assert ('cx' as char) == 'c'
assert 'cx'.asType(char) == 'c'
```

```
char a='a'
assert Character.digit(a, 16)==10 : 'But Groovy does boxing'
assert Character.digit((char) 'a', 16)==10

try {
  assert Character.digit('a', 16)==10
  assert false: 'Need explicit cast'
} catch(MissingMethodException e) {
}
```

Primitives and wrappers

Because Groovy uses Objects for everything, it autowraps references to primitives. Because of this, it does not follow Java's behavior of widening taking priority over boxing. Here's an example using int

```
int i
m(i)

void m(long l) {
  println "in m(long)"
}

void m(Integer i) {
  println "in m(Integer)"
}
```

What do you call function?

Behaviour of ==

In Java == means equality of primitive types or identity for objects. In Groovy == means equality in all cases. It translates to a.compareTo(b) == 0, when evaluating equality for Comparable objects, and a.equals(b) otherwise. To check for identity (reference equality), use the is method: a.is(b). From Groovy 3, you can also use the === operator (or negated version): a === b (or c !== d)

GroovyObject

```
package groovy.lang;

public interface GroovyObject {

    Object invokeMethod(String name, Object args);

    Object getProperty(String propertyName);

    void setProperty(String propertyName, Object newValue);

    MetaClass getMetaClass();

    void setMetaClass(MetaClass metaClass);
}
```

Groovy.lang.GroovyObject is the main interface in Groovy as the Object class is in Java. GroovyObject has a default implementation in the groovy.lang.GroovyObjectSupport class and it is responsible to transfer invocation to the groovy.lang.MetaClass object.

GroovyInterceptable

The groovy.lang.GroovyInterceptable interface is marker interface that extends GroovyObject and is used to notify the Groovy runtime that all methods should be intercepted through the method dispatcher mechanism of the Groovy runtime.

```
package groovy.lang;

public interface GroovyInterceptable extends GroovyObject {
}
```

invokeMethod

```
class SomeGroovyClass {

    def invokeMethod(String name, Object args) {
        return "called invokeMethod $name $args"
    }

    def test() {
        return 'method exists'
    }
}

def someGroovyClass = new SomeGroovyClass()

assert someGroovyClass.test() == 'method exists'
assert someGroovyClass.someMethod() == 'called invokeMethod someMethod []'
```

```
class Interception implements GroovyInterceptable {

    def definedMethod() { }

    def invokeMethod(String name, Object args) {
        'invokedMethod'
    }
}
class InterceptableTest extends GroovyTestCase {

    void testCheckInterception() {
        def interception = new Interception()

        assert interception.definedMethod() == 'invokedMethod'
        assert interception.someMethod() == 'invokedMethod'
    }
}
```

Runtime metaprogramming