esJUG
Grupo de Usuários de Java do Estado do Espírito Santo

jumbo GRUPPE

Closures in Java

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and

License for use and distribution

This material is available for non-commercial use and can be derived and/or redistributed, as long as it uses an equivalent license.

Attribution-Noncommercial-Share Alike 3.0 Unported

http://creativecommons.org/licenses/by-nc-sa/3.0/

You are free to share and to adapt this work under the following conditions: (a) You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work); (b) You may not use this work for commercial purposes. (c) If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

About the author – Vítor Souza

- Education:
  - Computer Science graduate, masters in Software Engineering – (UFES, Brazil), starting PhD at U. Trento.
- Java:
  - Developer since 1999;
  - Focus on Web Development;
  - Co-founder and coordinator of ESJUG (Brazil).
- Professional:
  - Substitute teacher at Federal University of ES;
  - Engenho de Software Consulting & Development.
- Contact: vitorsouza@gmail.com

Grupo de usuários de Java do Estado do Espírito Santo

Summary

- What are closures?
- Java and closures;
- The BGGA proposal;
- The FCM proposal;
- The CICE+ARM proposal;
- Discussion.

Grupo de Usuários de Java do Estado do Espírito Santo

What are closures?

- FOLDOC:
  A data structure that holds an expression and an environment of variable bindings in which that expression is to be evaluated.
- Wikipedia:
  A function that is evaluated in an environment containing one or more bound variables.
- Most general definitions:
  - Closed variable environment;
  - 1st class citizen.

Grupo de Usuários de Java do Estado do Espírito Santo

What are closures?

- In practice:
  - Blocks of code that access variables in the external scope;
  - Can be created dynamically;
  - Can be stored in variables;
  - Can be passed as parameters;
  - Can be invoked dynamically.

Simple example (Groovy)

```
def clos = { println "Hello, World!" }
clos()
closureMethod(clos)

def closureMethod(closure) {
  closure()
}
```

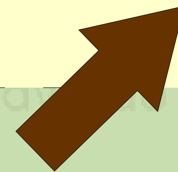
```
$ groovy Teste
Hello, World!
Hello, World!
```

Grupo de Usuários de Java do Estado do Espírito Santo

Parameters and variables (Groovy)

```
def sum = { a, b -> a + b }
def result = sum(7, 5)
println result                        // 12

def doubleResult = { println result * 2 }
doubleResult()                        // 24
result++
doubleResult()                        // 26
```



The closure refers to a variable that is external to it

Origins

- The 50's, MIT: Lisp, lambda expressions (anonymous functions);
- The 70's: Scheme - 1st LISP dialect to do static (lexical) binding, producing a closure (term imported from mathematics);
- Smalltalk: 1st OO language to include closures:
  - Motivation: creation of control constructs in the API.

Grupo de Usuários de Java do Estado do Espírito Santo

Creating a for-each using closures

```
void forEach(array, funct) {
  for (i = 0; i < array.size(); i++)
    funct(array[i])
}

def words = ["Closures", "are", "cool!"]
forEach(words) {
  item -> print item + " "
}
println ""
```

- There would have been no need to change the Java language to include the enhanced for if Java had closures before version 5...

And if I need something specific?

- What if we had a statistical framework that times tasks and stores them in a database?
- Would this control structure ever make into the language?
- Now think transaction management...
- Think the amount of frameworks out there...

```
// Times the execution and stores in the DB
// under the name “block-1”:
time("block-1") {
  // Your code here...
}
```

Grupo de Usuários de Java do Estado do Espírito Santo

What about Java and closures?

- First of all: Java is turing-complete;
- Java has closures in the form of anonymous inner classes or the reflection API:
  - Are more verbose;
  - Do not allow compile-time checking;
  - Make refactoring difficult.

Grupo de Usuários de Java do Estado do Espírito Santo

For each in Java using AICs

```
interface Closure<T> {
  void invoke(T arg);
}
public class Test {
  static <T> void forEach(T[] arr, Closure<T> fun) {
    for (int i = 0; i < arr.length; i++) {
      fun.invoke(arr[i]);
    }
  }
  public static void main(String[] args) {
    String[] words = new String[] { "A", "B", "C" };
    forEach(words, new Closure<String>() {
      public void invoke(String arg) {
        System.out.print(arg + " ");
      }
    });
  }
}
```

For each in Java using AICs

- Further complications:
  - The `invoke()` method in the Closure interface does not declare any exceptions. You can't throw them;
  - The `invoke()` method does not declare any return type. You can't return anything;
  - Variables used in an AIC have to be final. To simulate a return value, you'd have to use an array of size 1.

Grupo de Usuários de Java do Estado do Espírito Santo

Should we add closures to Java, then?

- That question generated a lot of debate and three main propositions:
  - BGGA (a.k.a. “Full Closures”):
    - Name after the authors: Gilad **B**racha, Neal **G**after, James **G**osling and Peter von der **A**hé.
  - FCM:
    - First Class Methods;
    - Authors: Stephen Coleburn and Stefan Schulz.
  - CICE+ARM:
    - Concise Instance Creation Expressions + Automatic Resource Management;
    - Authors: Bob Lee, Doug Lea and Joshua Bloch.

ESJUG
Grupo de Usuários de Java do Estado do Espírito Santo

The prototypes

- All proposals come with working prototypes;
- BGGA is being included in OpenJDK 7;
- Installation:
  - Install the latest JDK 6;
  - Copy the files from the prototype over the JDK.
- Useful links:
  - BGGA: www.javac.info
  - OpenJDK: openjdk.java.net/projects/closures
  - FCM: docs.google.com/View?docid=ddhp95vd_0f7mcns
  - CICE+ARM: www.slm888.com/javac

Grupo de Usuários de Java do Estado do Espírito Santo

The BGGA proposal

- Can create closures in Java:
  `{ parameters => commands expression }`
- Changes the type system to add Function Types:
  `{ params => return-type throws exceptions }`
- Variables that are modified inside the closure receive the `@Shared` annotation;
- Can assign closures to variables of a compatible interface type;
- Unrestricted closures allow for control constructs syntax.

BGGA – Hello, World!

```
// Using only the expression.
String msg = { => "Hello, BGGA!" }.invoke();
System.out.println(msg);

// Using only a command.
{ => System.out.println("Hello, BGGA!"); }.invoke();

// Using parameters.
{ String msg =>
System.out.println(msg); }.invoke("Hello, BGGA!");

// Local variables can be declared inside!
int a = { int x => int y = x + 1; y * 2}.invoke(9);
System.out.println(a); // 20
```

BGGA – Function types

```
static void doIt() throws Exception {
  throw new Exception("doIt() threw exception!");
}
public static void main(String[] args) {
  {int, int => int} sum = { int a, int b => a + b };
  print(sum, 7, 8);
  { => void throws Exception } it = { => doIt(); };
  try {
    it.invoke();
  } catch (Exception e) {
    System.out.println(e);
  }
}
static void print({int, int => int} cl, int a, int b)
{
  int x = cl.invoke(a, b);
  System.out.println(x);
}
```

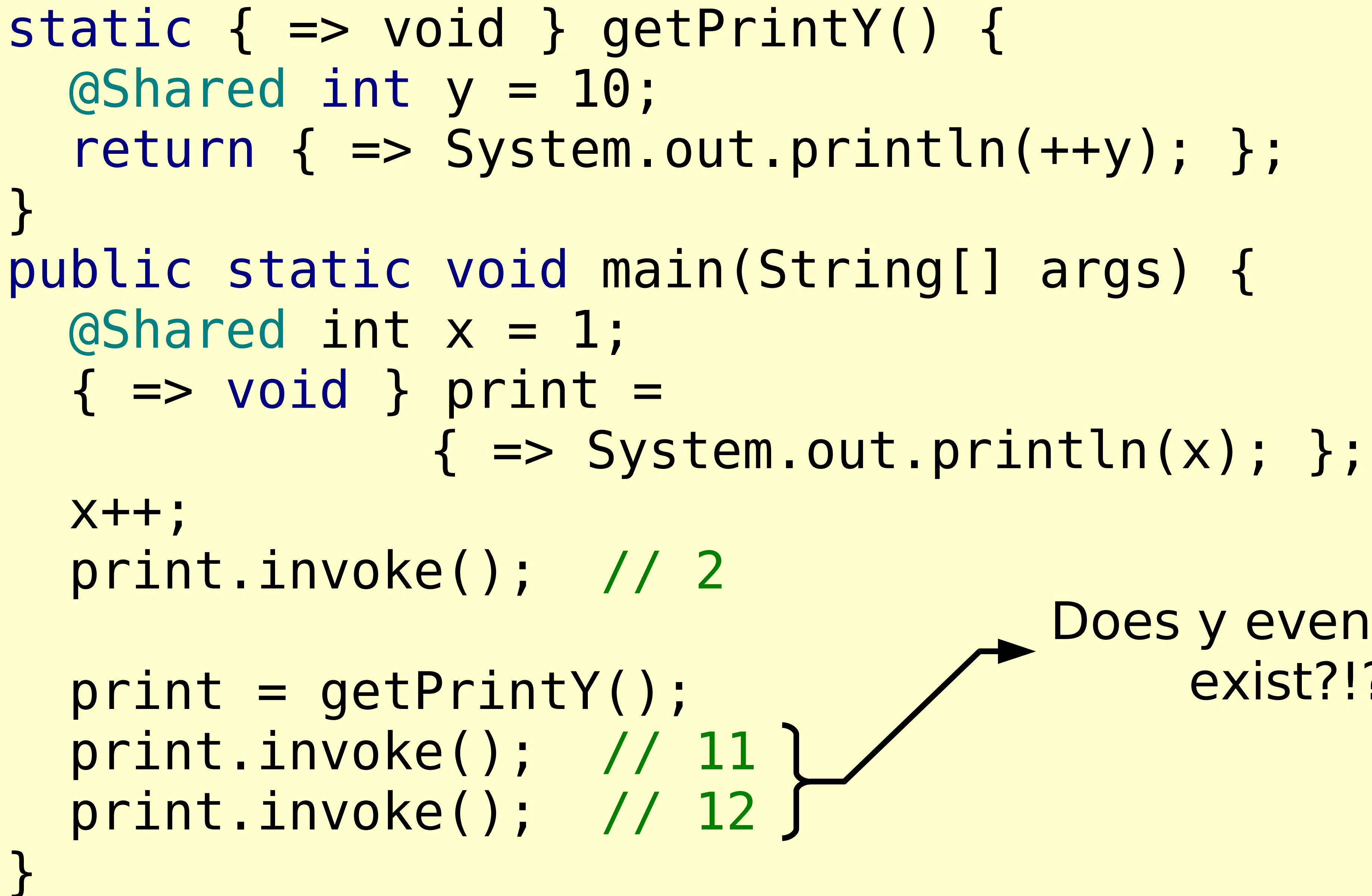
BGGA – Function types hierarchy

```
// { => Integer } is subtype
// of { => Number }
{ => Number } p1;
p1 = { => Integer.valueOf(19) };

// { Object => void } is subtype
// of { String => void }
{ String => void } p2;
p2 = { Object o => System.out.println(o); };
```

Can anyone tell me why?

BGGA – Using shared variables



BGGA – Using compatible interfaces

```
interface Adder {
  int add(int x, int y);
}

/* In some other class' main method: */
Adder adder = { int a, int b => a + b };
int y = adder.add(10, 5);
System.out.println(y);
```

Grupo de Usuários de Java do Estado do Espírito Santo

Are you thinking java.lang.Runnable?

BGGA – Control constructs

```
static void time(String s, { ==> void } f) {
  long t1 = System.nanoTime();
  f.invoke();
  long t = System.nanoTime() - t1;
  System.out.println("Block " + s +
                     " executed in " + t + "ns");
}

public static void main(String[] args) {
  time("block-1") {
    int[] vet = new int[1000];
    for (int i = 0; i < 1000; i++)
      vet[i] = i * i;
  }
}
```

The FCM proposal

- Makes methods 1st class citizens:
  - Methods: `Class#method-name(params);`
  - Constructors: `Class#(params);`
  - Properties: `Class#property-name.`
- Uses reflection under the hood;
- Unlike reflection, visibility rules apply;
- Also adds new type: Method-type
  `#(return-type(param-types) throws exceptions)`
- invoke(), compatible interfaces, anonymous methods (closures).

Grupo de Usuários de Java do Estado do Espírito Santo

FCM – Method literals

```
import java.lang.reflect.*;

public class ExampleFCM {
  String message;

  ExampleFCM(String message) {
    this.message = message;
  }

  void print() {
    System.out.println(message);
  }

  // Continues...
```

FCM – Method literals

```
  // Continued:
  public static void main(String[] args)
                                 throws Exception {
    Constructor<ExampleFCM> constr;
    constr = ExampleFCM#(String);

    ExampleFCM obj;
    obj = constr.newInstance("Hello, FCM!");

    Method method = ExampleFCM#print();
    method.invoke(obj);
  }
}
```

FCM – Bound method references

```
  public static void main(String[] args)
                                    throws Exception {
    // [...]
    ExampleFCM obj;
    obj = constr.newInstance("Hello, FCM!");

    // Before:
    Method method = ExampleFCM#print();
    method.invoke(obj);

    // After:
    Method boundMethod = obj#print();
    boundMethod.invoke();
  }
}
```

FCM – Method-types

```
static int add(int a, int b) {
  return a + b;
}

/* In the main method: */

// Declaring a variable of Method-type
#(int(int, int)) method;
method = ExampleFCM#add(int, int);
System.out.println(method.invoke(10, 5));

// Assigning to a compatible interface.
// Assume same Adder interface shown before.
Adder adder = method;
System.out.println(adder.add(6, 9));
```

FCM – Anonymous methods

```
// Creating an anonymous method.
List<String> list = ...
Collections.sort(list,
               #(String str1, String str2) {
  return str1.length() - str2.length();
});

// With no parameters, parentheses
// can be omitted.
Executor exec = ...
exec.execute(#{
  // Your code here...
});
```

FCM – Java Control Abstraction

```
public static void
          usingFileReader(#(void(FileReader)) block :
          File file) throws IOException {
  FileReader reader = null;
  try {
    reader = new FileReader(file);
    block.invoke(reader);
  }
  finally {
    if (reader != null) {
      try {
        reader.close();
      } catch (IOException e) {
        // Ignore...
      }
  }
} // Continues...
```

FCM – Java Control Abstraction

```
// Continued, somewhere else:
usingFileReader(FileReader reader : file) {
  // Read file...
}
```

Grupo de Usuários de Java do Estado do Espírito Santo

The CICE+ARM proposal

- Java already has closures (anonymous inner classes), we just need some syntactic sugar;
- CICE proposes to simplify AICs syntax:
  - SAM (single abstract method) classes;
  - External variables are automatically final and can be changed if declared public.
- ARM proposes to simplify resource management:
  - Streams, readers/writers, connections... Anything that has to be disposed after use;
  - `java.io.Closable?`

Grupo de Usuários de Java do Estado do Espírito Santo

CICE

```
// Without CICE:
List<String> list = ... ;
Collections.sort(list, new
Comparator<String>() {
  public int compare(String s1, String s2) {
    return s1.length() - s2.length();
  }
});

// With CICE:
List<String> list = ... ;
Collections.sort(list,
  Comparator<String>(String s1, String s2)
{ return s1.length() - s2.length(); });
```

Without ARM

```
BufferedInputStream is1 = null;
BufferedInputStream is2 = null;
try {
  is1 = ...;
  is2 = ...;
  // Use streams for something...
} finally {
  try {
    if (is1 != null)
      is1.close();
  } finally {
    if (is2 != null)
      is2.close();
  }
}
```

With ARM

```
do (BufferedInputStream is1 = ..., is2 = ...)
{
  // Use streams for something...
}
```

Grupo de Usuários de Java do Estado do Espírito Santo

Discussion – Pro-closures arguments

- Closures will:
  - Simplify code and avoid non-elegant solutions;
  - Allow frameworks to specify control constructs;
  - Avoid polluted interfaces* (e.g. could simplify the `java.util.concurrent` API);
  - Closures are complex now as recursive functions were complex some day. For fans of Ruby, Python, Groovy, Scala and C# 2.0, it's not complex anymore...
  - Simplify the evolution of the Java programming language (see for each example).

* see http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166ydocs/jsr166y/forkjoin/Ops.html

Discussion – Anti-closures arguments

- Closures:
  - Are already present in Java in the form of AICs. We just need to simplify the syntax;
  - Are complex by themselves and, if combined with other complex stuff (Generics) becomes too much;
  - Are more related to functional programming and if you mix both programming styles things become nasty;
  - (BGGA) have confusing syntax. { and } are already used with other meanings;
  - Will lead to dialects because of control constructs;
  - Are being added for marketing reasons.

Grupo de Usuários de Java do Estado do Espírito Santo

Discussion

- Joshua Bloch: closures are against the “Feel of Java” (Gosling 1996 talk);
- Gosling says he's misinterpreted the talk. Closures weren't added from the beginning because of time constraints;
- Patrick Naughton confirms Gosling's version, saying that Bill Joy wanted Closures on Java since version 1.0*;
- BGGA has already a draft for a JSR. All authors (except Bloch and Gosling) support it.

* see http://www.blinkenlights.com/classiccmp/javaorigin.html

```
public class ClosuresPuzzler {
  static void test1b({ => void } closure) {
    closure.invoke();
    int x = 20;
    closure.invoke();
    System.out.println(x);
  }
  static void test1() {
    @Shared int x = 10;
    test1b({ => System.out.print((x++) + ", "); });
  }
  static void test2() {
    List<{ => int }> closures = new ArrayList<{ => int }>();
    @Shared int i = 0;
    while (i++ < 10)
      closures.add({ => i });
    int total = 0;
    for ({ => int } c : closures)
      total += c.invoke();
    System.out.println(total);
  }
  public static void main(String[] args) {
    test1();
    test2();
  }
}
```

Grupo de Usuários de Java do Estado do Espírito Santo

jumbo GRUPPE

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and