Manipulação de bytecodes Java

Eduardo Oliveira de Souza
esouza@ime.usp.br

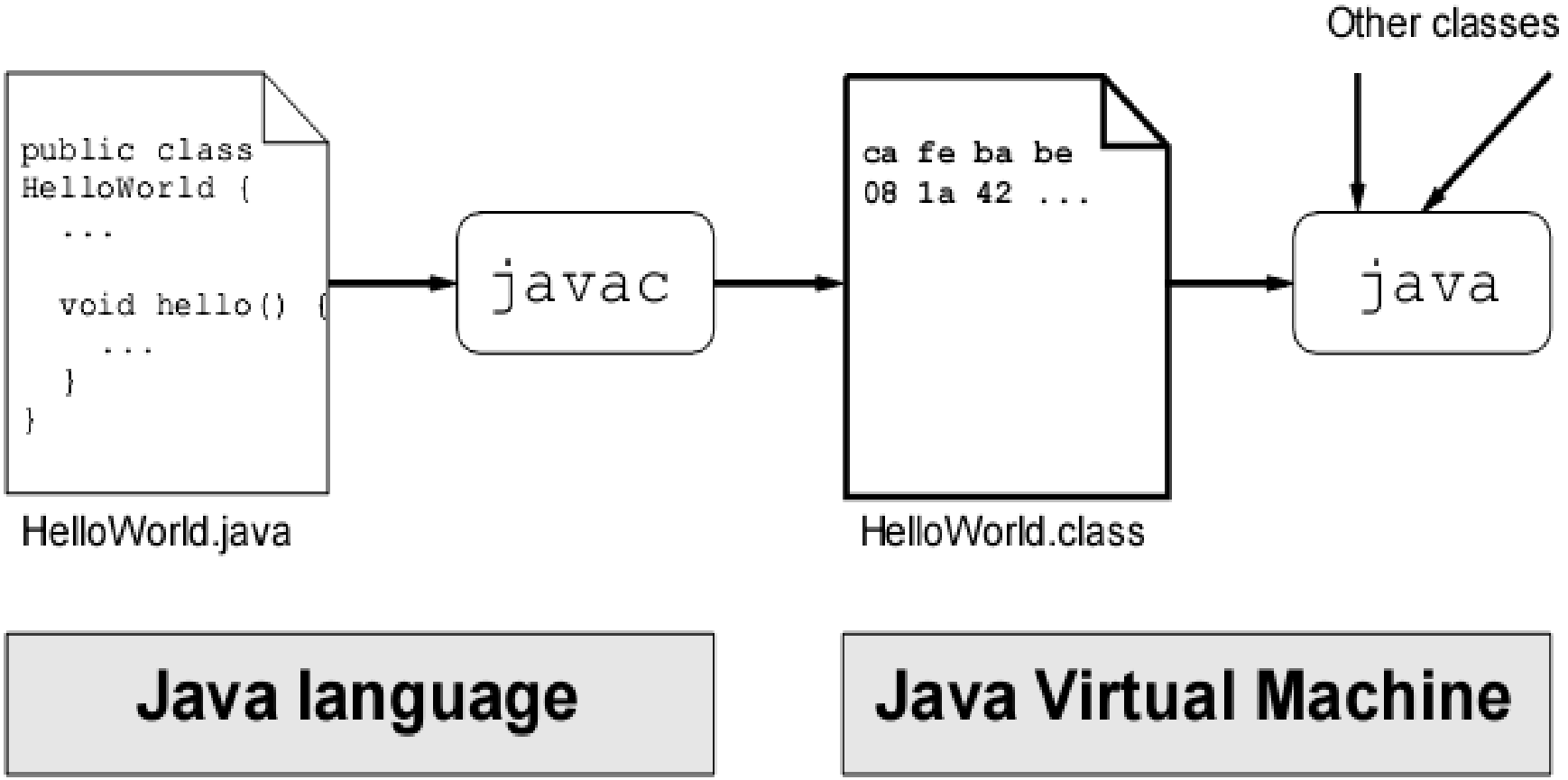
Roteiro

- *Bytecodes*
- Manipulação de *bytecodes*
- Ferramentas
- Considerações Finais
- Bibliografia

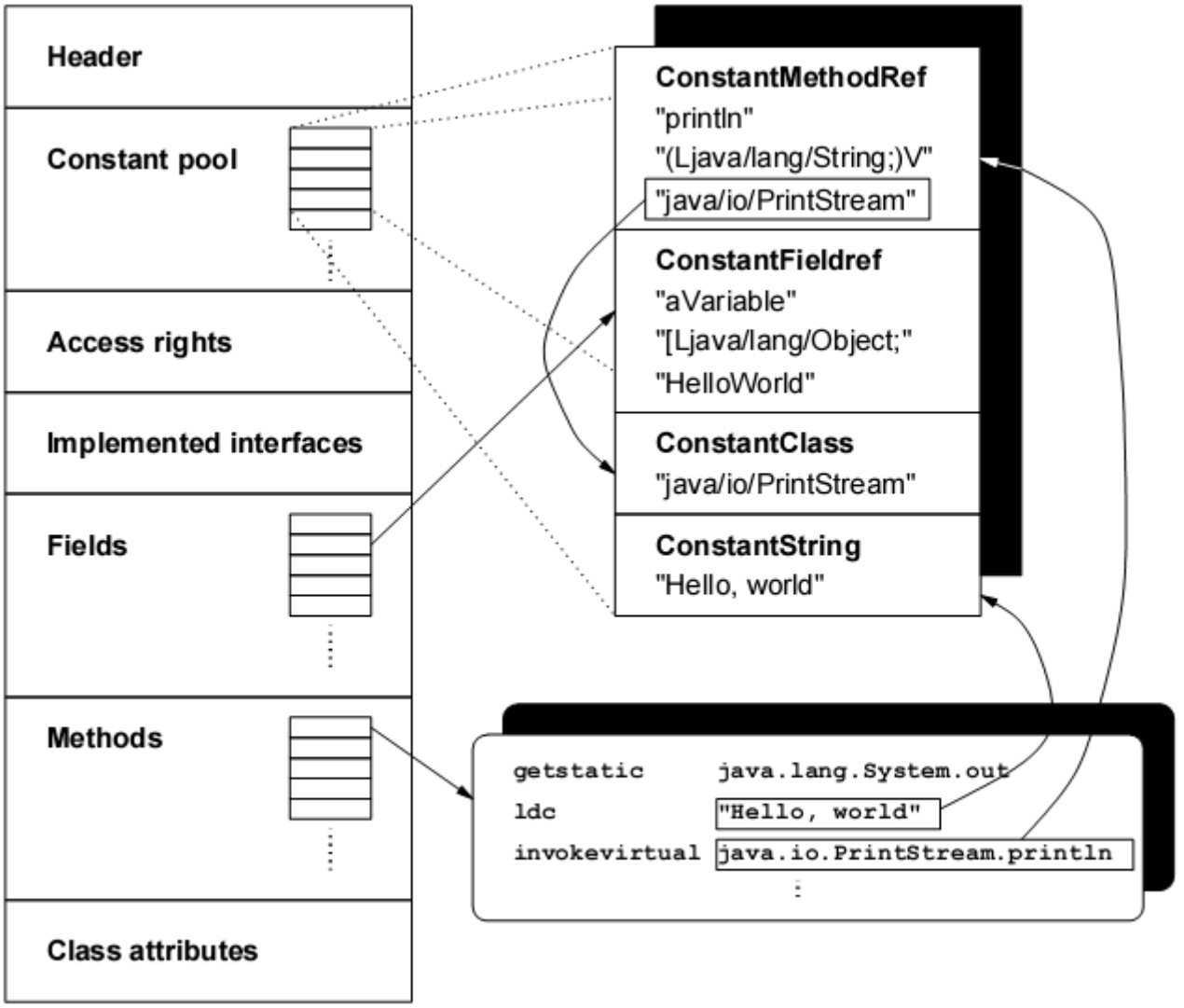
O que são *bytecodes*?

- Código intermediário entre código-fonte e aplicação final.
- Conjunto de instruções pequeno e fácil de entender.
- Grande vantagem: Portabilidade

Compilação e execução de classes Java



Estrutura de um arquivo Java *class*



HelloWorld.class

Conjunto de instruções de bytecodes Java

- Composto de 212 instruções
- 44 *opcodes* reservados para futuras extensões ou otimizações da na JVM
- Instruções podem ser divididas em grupos:
  - Operações de pilha;
  - Operações aritméticas;
  - Controle de fluxo;
  - Operações de carga e armazenamento;
  - Acesso a campos;

Conjunto de instruções de bytecodes Java

- Grupos de instruções (cont.)
  - Invocação de método;
  - Alocação de objetos;
  - Conversão e checagem de tipo.

Código Java vs Java *Class*

```
import java.io.*

public class Faculty {
   private static BufferedReader in = new
      BufferedReader(new
               InputStreamReader(System.in));

   public static final int fac(int n) {
      return (n == 0)? 1 : n * fac(n - 1);
   }

   ...
```

Código Java vs Java *Class*

```
public static final int readInt() {
   int n = 4711;
   try {
      System.out.print("Please enter a number> ");
      n = Integer.parseInt(in.readLine());
   } catch (IOException e1){
      System.err.println(e1);
   } catch (NumberFormatException e2){
      System.err.println(e2);
   }
}
```

Código Java vs Java *Class*

- Método Fac

```
0:  iload_0
1:  ifne            #8
4:  iconst_1
5:  goto            #16
8:  iload_0
9:  iload_0
10: iconst_1
11: isub
12: invokestatic    Faculty.fac (I)I (12)
15: imul
16: ireturn
```

Código Java vs Java *Class*

- Método readInt

```
0:  sipush      4711
3:  istore_0
4:  getstatic       java.lang.System.ou Ljava/io/PrintStream;
7:  ldc             "Please enter a number> "
9:  invokevirtual   java.io.PrintStream.print (Ljava/lang/String;)V
12: getstatic       Faculty.in Ljava/io/BufferedReader;
15: invokevirtual   java.io.BufferedReader.readLine  ()Ljava/lang/String;
18: invokevirtual   java.lang.Integer.parseInt (Ljava/lang/String;)I
21: istore_0
22: goto            #44

...
```

Código Java vs Java *Class*

- Método readInt(cont)

```
25: astore_1
26: getstatic       java.lang.System.err Ljava/io/PrintStream;
29: aload_1
30: invokevirtual   java.io.PrintStream.println (Ljava/lang/Object;)V
33: goto            #44
36: astore_1
37: getstatic       java.lang.System.err Ljava/io/PrintStream;
40: aload_1
41: invokevirtual   java.io.PrintStream.println (Ljava/lang/Object;)V
44: iload_0
45: ireturn
```

Manipulação de *bytecodes*

- O que é a manipulação de *bytecodes*?
- Por que manipular *bytecodes*?

Razões para manipular *bytecodes*

- Adicionar ou remover instruções
- Mudar o fluxo de controle
- Buscar por certos padrões de código usando expressões regulares.

Exemplos de utilização da manipulação de bytecodes

- Otimização de código;
- Implementação de tipos parametrizados em Java;
- Extensões para a linguagem Java;
- Ferramentas de análise em tempo de execução
- Análise estática de *bytecodes;*
- Delegação automatizada;

Exemplos de utilização da manipulação de bytecodes

- Implementação de conceitos de *Aspect-Oriented Programming* (AOP);
- Metaprogramação;

Ferramentas

- BCEL API (JavaClass) – Markus Dahm – 1998
- JikesBT (Alphaworks – IBM)
- ASM
- Jinline
- Javassist
- SERP
- gnu.bytecode
- Cojen

Ferramentas

- Soot
- Jen
- JBET
- Cglib
- Jiapi

BCEL

- BCEL = The Byte Code Engineering Library
- Atualmente faz parte do projeto Jakarta (http://jakarta.apache.org/bcel).
- Atualmente é o mais usado
- Presentes em projetos importantes:
  - AspectJ
  - Jboss

BCEL

- Composto de 3 partes
  - Um pacote que contém classes que refletem o formato de arquivo class e não é destinado a modificações de bytecodes;
  - Um pacote para gerar e modificar dinamicamente objetos JavaClass;
  - Vários códigos de exemplo e utilitários.

Exemplo de uso do BCEL

- Lendo um arquivo class em um objeto *JavaClass*

```
JavaClass clazz =
  Repository.lookupClass("java.lag.String");
```

Exemplo de uso do BCEL

- Acessando dados de um arquivo *class*

```
System.out.println(clazz);
printCode(clazz.getMethods());
...
public static void printCode(Method[] methods) {
   for(int i=0; i < methods.length; i++) {
   System.out.println(methods[i]);
   Code code = methods[i].getCode();
   if(code != null)  //Método não abstrato
      System.out.println(code); }}
```

Sentença `if` em *bytecodes*

- Código Java

```
if(a == null)
   a = b;
```

- Código em *bytecodes*

```
InstructionList il = new InstructionList();
IfInstruction i = new IFNONNULL(null);
// Carrega a variável local 0(a) na pilha
il.append(new ALOAD(0));
il.append(i); // Usa condição negada
// Carrega a variável local 0(b) na pilha
il.append(new ALOAD(1));
il.append(new ASTORE(0)); // Armazena em a
// Define alvo auxiliar para o caso de a != null
i.setTarget(il.append(new NOP());
```

ASM

- Trabalho de Eric Bruneton, Romain Lenglet and Thierry Coupaye – 2002
- Vantagens:
  - Mais enxuto. Enquanto SERP tem ~ 80 classes para os tipos de instruções e BCEL tem ~ 270 classes, ASM tem somente 13 classes.
  - Mais rápido que SERP e BCEL
- Desvantagens:
  - Não é tão simples de manipular

SERP

- Criado por Abel White
- Licença BSD
- Vantagens
  - Facilidade de uso
  - Poder
  - Gerência de *pool* de constantes

SERP

- Desvantagens
  - Velocidade
  - Memória
  - Modificações Multi-thread
  - Modificações em modo protegido

Considerações finais

- A existência de bytecodes Java permite que se faça alterações no Software, mesmo sem ter acesso ao código-fonte, seja melhorando o software ou inserindo novas funcionalidades ou conceitos.
- Existem várias ferramentas e APIs que fazem manipulação de *bytecodes,* umas mais simples, outras mais complexas, umas mais fáceis de manipular, outras exigindo um conhecimento maior.

Bibliografia

- Dahm, M., *Byte Code Engineering with BCEL API*, Technical Report, Abril , 2001
- Dahm, M., *Byte Code Engineering,* Proceedings JIT'99, Springer, 1999
- Tanter, E., Devillechaise, M.S., Noyé, J., Piquer, J., Altering Java Semantics via Bytecode Manipulation, GPCE 2002, LNCS 2487, 2002, p. 283-298.
- Bruneton, E., Lenglet, R., Coupaye, T., *Adaptable and extensible component systems*, novembro, 2002, Grenoble, França
- White, A., SERP (http://serp.sourceforge.net) – acesso em 20 de outubro de 2006.