Softwares de Sistemas
Aula – Java Virtual Machine

Software Básico, turma A

Prof. Marcelo Ladeira
CIC/UnB

Baseado no livro "The Java Virtual Machine Specification", Second Edition, Tim Lindholm & Frank Yellin.

Sumário

- Estruturação da JVM
  - Introdução
  - Tipos de dados
  - Tipos primitivos e valores de dados
  - Tipos de referência e valores de dados
  - Estruturas de dados em tempo de execução
  - Frames
  - Representação de objetos
- Instruções da JVM

Introdução

- **Executa código Objeto**
  - **código objeto compilado no formato .class**
  - **formato binário independente do hardware e do sistema operacional**
    - **armazenado em um arquivo no formato class.**
  - **formato class**
    - **define completamente a representação de uma classe ou interface.**

Tipos de Dados

- **Primitivos**
  - **numérico (integral e ponto flutuante), booleano e o tipo** returnAddress
  - **assume apenas valores primitivos**
- **De referência (ponteiros para objetos)**
  - **assumem apenas valores de referência**
- **Ambos os tipos de valores podem ser armazenados em variáveis, passados como argumentos, retornados por métodos e utilizados em operações específicas.**

Tipos de Dados

- **Comentários**
  - **A JVM não realiza checagem de tipo**
    - **Java é fortemente tipada e toda checagem deve ser feita durante a compilação**
  - **valores de tipos primitivos não possuem tags e nem requerem checagem dinâmica para serem distinguidos de valores de tipos de referência.**
    - **as instruções da JVM distinguem os tipos de seus operandos usando instruções projetadas para operar com valores de tipos específicos.**
      - iadd**,** ladd**,** fadd **e** dadd **(**..., *value1, value2* ⇒ ..., *result* **)**
        - **somam 2 valores numéricos e produz um resultado numérico.**
        - **cada qual é específica para um tipo de operando:** int**,** long**,** float**, e** double**, respectivamente.**

Tipos de Dados

- **Comentários**
  - **Um objeto é uma instância de uma classe ou um array alocado dinamicamente.**
    - **uma referência para um objeto é um dado do tipo de referência.**
      - **valores de tipo de referência podem ser vistos como ponteiros para objetos**
      - **mais de uma referência pode existir para um objeto**
    - **objetos são sempre operados, passados ou testados via valores de tipos de referência**
      - **não existem ponteiros em Java.**

Se existir mais de uma referência para um objeto, como o coletor de lixos sabe quando desalocá-lo?

Tipos Primitivos e Valores de Dados

- **Tipos integrais (como em Java)**
  - byte**, de -128 a 127 (-2^7 to 2^7-1), inclusive**
  - short**, de -32768 a 32767 (-2^{15} to 2^{15}-1), inclusive**
  - int**, de -2147483648 a 2147483647 (-2^{31} to 2^{31}-1), inclusive**
  - long**, de -9223372036854775808 a 9223372036854775807 (-2^{63} to 2^{63}-1), inclusive**
  - char**, de 0 a 65535 inclusive**
- **Tipos ponto flutuante (IEEE 754)**
  - float**, cujos valores são elementos do conjunto de valores** float
  - double**, cujos valores são elementos do conjunto de valores** double
- **Tipos booleanos**
  - **valores verdade** true **(1) e** false **(0)**
  - **Não há instruções JVM específicas para tipo booleano**
    - **Expressões booleanas em Java são mapeadas como expressões** int **em JVM**
    - **Arrays do tipo booleano são mapeados em** bytes **pela JVM (8 bits)**

Tipos Primitivos e Valores de Dados

- **Tipo returnAddress**
  - **é o único tipo primitivo que não está associado diretamente com um tipo de dados da linguagem Java.**
  - **é usado pelas instruções** *jsr (jump subroutine)*, *ret (return from subroutine)* **e** *jsr_w* (*jump subroutine – wide index*) **da JVM**
    - **seus valores são ponteiros para código de operação dessas instruções de desvio incondicional da JVM.**
      - jsr branchbyte1 branchbyte2 (... ⇒ ..., *address* **) – Offset de 16 bits**
      - jsr_w branchbyte1 branchbyte2 branchbyte3 branchbyte4 **– Offset de 32 bits**
      - ret index **– Offset de um** byte **índice para o vetor de variável** locais **do frame que contém o offset de retorno**

offset de retorno (tipo returnAddress) é o endereço da instrução seguinte.

Tipos de Referência e Valores de Dados

- Tipos de classes
  - Valores são referências a instâncias de classes criadas dinamicamente
- Tipos de array
  - Valores são referências a arrays criados dinamicamente
- Tipos de interface
  - Valores são referências a instâncias de classes que implementam interfaces ou arrays
- Valor de referência especial null
  - referência vazia (indica que não aponta um objeto)
  - não tem tipo em runtime mas pode ser moldada para qualquer tipo de referência
  - a especificação da JVM não define um valor concreto para codificar null

Estruturas de Dados em Tempo de Execução

- Alocadas na iniciação da JVM
  - Heap
  - Área de métodos
    - Pool de constantes
- Alocadas na iniciação de threads
  - Registro pc
  - Pilha da JVM
  - Pilha de método nativo

Estrutura de Dados em Tempo de Execução

- **Registro pc**
  - **Cada thread da JVM possui seu próprio registro pc (contador de programa).**
  - **Em um certo instante, cada thread executa o código de um único método que é denominado método corrente**
    - **método corrente é não nativo (método Java)**
      - **o registro pc contem o endereço da instrução da JVM sendo executada.**
    - **método corrente é nativo (método não Java)**
      - **o valor do registro pc é indefinido.**
      - **pc pode conter um** returnAddress **ou um ponteiro nativo na plataforma especifica em que a JVM foi implementada**

Estrutura de Dados em Tempo de Execução

- **Pilha da JVM**
  - **armazena** frames **associados a cada chamada de um método em uma thread.**
    - **Permite manipular variáveis locais, resultados parciais e participa da chamada e retorno de um método.**
    - **A implementação da JVM deve permitir pilha de tamanho fixo ou expansão e contração dinâmica como requerido pela computação.**
      - **Pode gerar as seguintes exceções**
        - StackOverflowError **se a computação em uma thread requer uma pilha maior do que a permitida.**
        - OutOfMemoryError **se não tem memória suficiente para expandir dinamicamente uma pilha ou para criar uma pilha para uma nova thread.**

O tamanho máximo de uma pilha é definido pela opção não padronizada "-Xss<size>" da JVM (programa java)

Estrutura de Dados em Tempo de Execução

- **Heap**
  - **O heap é criado na iniciação da JVM e compartilhado por todas as threads da JVM.**
    - **É utilizado para alocar memória para os objetos (todas as instâncias de classes e arrays)**
    - **Objetos não são explicitamente desalocados**
      - **Objetos sem caminhos de acessos são coletados por um coletor de lixo.**
    - **A implementação da JVM deve permitir controlar o tamanho inicial do heap, se pode ser dinamicamente expandido ou contraído e o seu tamanho máximo e mínimo.**
    - **Pode gerar a seguinte exceção**
      - OutOfMemoryError **se uma computação requer mais heap do que o que pode ser disponibilizado.**

Os valores inicial e máximo do heap são definidos, respectivamente, pelas opções "Xms<size>" e "Xmx<size>" da JVM (programa java)

Estrutura de Dados em Tempo de Execução

- **Área de Métodos**
  - **Essa área é criada na iniciação da JVM e compartilhada por todas as threads da JVM.**
    - **análoga a área de armazenamento para código compilado de uma linguagem convencional**
      - **armazena estruturas de dados em base de classe**
        - **pool de constantes, atributos e dados de métodos, código de métodos e código de construtores.**
    - **Pode gerar a seguinte exceção**
      - OutOfMemoryError **se o requisito para a área de memória do método não pode ser atendido.**

Estrutura de Dados em Tempo de Execução

- **Pool de Constantes**
  - **O pool para uma classe ou interface é construído, na área de método, quando a classe ou interface é criada pela JVM.**
    - **armazena estrutura de dados em base de classe ou interface**
      - **representação runtime da tabela** constant_pool **no arquivo .class**
    - **contém diversos tipos de constantes, variando de literais numéricos conhecidos em tempo de compilação a referências a métodos e atributos que devem ser resolvidos em tempo de execução.**
      - **desempenha uma função similar a de uma tabela de símbolos**
    - **Pode gerar a seguinte exceção**
      - OutOfMemoryError **se a construção do pool de constantes para uma classe ou interface requer mais memória do que pode ser disponibilizada na área de método da JVM.**

Estrutura de Dados em Tempo de Execução
Pool de constantes

- **Comentários**
  - **Armazena todas as constantes associadas com a classe ou interface**
    - **inclui os valores das variáveis declaradas como “final”**
  - **Contribui para tornar menores os requisitos de memória pois evita a duplicação de constantes em cada instância (objetos) de uma classe.**
    - **códigos que empilham constantes a partir do pool utilizam um “índice de constante de pool”**
      - **operando de 1 byte (1 a 255) ou 2 bytes (1 a 65535) que segue o código do código de operação**

        ldc index (... ⇒ ..., *value*) **– empilha int, float ou referência a string literal.** Index **é um** byte **para o pool de constante**

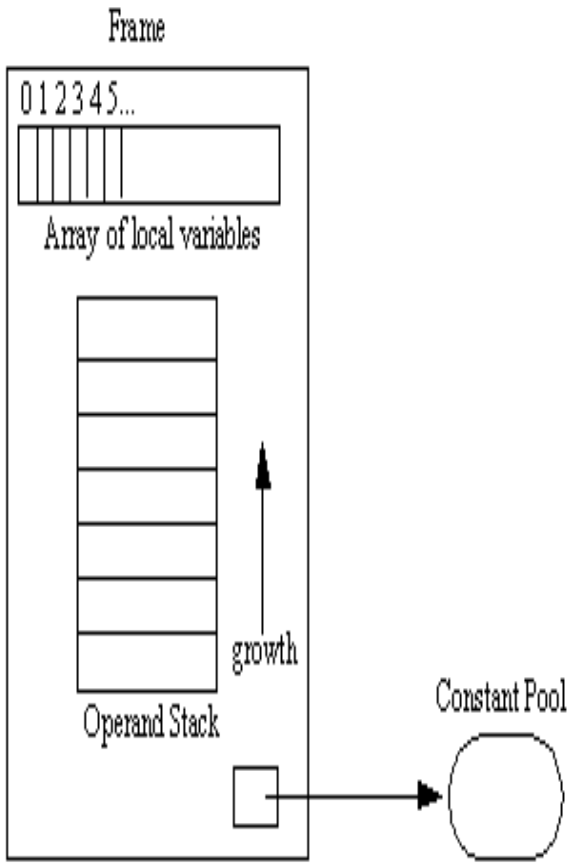
        ldc_w **– idem, para índice de 16 bits para pool de constantes.**

Estrutura de Dados em Tempo de Execução

- **Pilha de Método Nativo**
  - **É utilizada como suporte a métodos nativos (isto é, não Java).**
    - **É uma pilha convencional conhecida coloquialmente como "pilha C"**
    - **É alocada tipicamente por thread quando a thread é criada.**
  - **Pode gerar as seguintes exceções**
    - StackOverflowError **se a computação em uma thread requer uma pilha de método nativo maior do que a permitida.**
    - OutOfMemoryError **se a pilha de método nativo pode ser dinamicamente expandida mas não há memória suficiente ou se não existe memória suficiente para criar a pilha de método nativo para uma nova thread.**

Frames

- Usados para armazenar dados e resultados parciais, para executar ligação dinâmica, retornar valores para métodos e disparar exceções.
  - Um novo frame é criado cada vez que um método é chamado.
    - O frame é destruído quando o método que o chamou encerra normalmente ou abruptamente (lança uma exceção).
  - É alocado a partir da pilha da JVM da thread que o criou.
    - Cada frame possui seu próprio array de variáveis locais, sua pilha de operandos e uma referência para o pool de constantes da classe do método corrente.



Frames

- **Comentários**
  - **Os tamanhos do array de variáveis locais e da pilha de operandos são determinados em tempo de compilação.**
    - **são armazenados junto com o código do método associado com o frame.**
      - **a memória para as estruturas de dados do frame pode ser alocada simultaneamente com a chamada do método**
  - **Frame, método e classe correntes**
    - **frame para o método em execução (método corrente)**
      - **é o único ativo em uma thread (frame corrente).**
        - **a classe que define o método é a classe corrente**
      - **as operações sobre variáveis locais e pilha de operandos são tipicamente referenciadas ao frame corrente.**

Frames

- **Comentários**
  - **Um frame deixa de ser o corrente se seu método chamar outro método ou se finalizar.**
    - **quando um método é chamado, um novo frame é criado.**
      - **esse frame torna-se o corrente quando o controle é transferido para o novo método.**
    - **no retorno do método, o frame corrente passa o valor resultante da chamada do método, se existir, para o frame prévio.**
      - **o frame corrente é descartado e o frame prévio torna-se o novo frame corrente.**
    - **um frame criado por uma thread é local a thread e não pode ser referenciado por qualquer outra thread**

Frames
Variáveis Locais

- **Array (tabela) de variáveis locais**
  - **contêm os parâmetros do método e valores das variáveis locais associadas a classe ou interface que gerou o frame**
    - **seu comprimento é determinado em tempo de compilação e armazenado com a representação binária da classe ou interface junto com o código para o método associado com o frame**
      - **função do número e tamanho das variáveis locais e parâmetros formais**
    - **Os parâmetros são armazenados primeiro, iniciando no índice 0.**
      - **se o frame é para um método construtor ou de instância, o ponteiro é armazenado no índice 0 e é seguido pelos parâmetros.**
      - **se é para um método estático, o primeiro parâmetro formal é armazenado no índice 0, e assim por diante.**
  - **Uma variável local pode armazenar um valor do tipo** boolean**,** byte**,** char**,** short**,** int**,** float**, de referência ou** returnAddress**.**
  - **um par de variáveis locais pode armazenar um valor do tipo** long **ou** double**.**

Um inteiro é considerado um índice no array de variáveis locais sss assume um valor entre 0 e o tamanho do array menos 1.

Frames
Pilha de Operandos (32 bits)

- **Usada para empilhar e desempilhar valores que são operandos de instruções da JVM.**
  - **tamanho determinado em tempo de compilação.**
  - **a pilha é criada vazia quando o frame é criado**
    - **algumas instruções empilham valores na pilha.**
    - **outras retiram operandos da pilha, manipula-os e empilham o resultado.**
      - **Por exemplo, iadd retira os dois inteiros no topo da pilha, soma-os e empilha o resultado**

        (..., *value1, value2* ⇒ ..., *result*)
    - **valores retornados pelos métodos são recebidos via pilha de operandos.**

Frames
Ligação Dinâmica

- a referência para o pool de constantes permite suportar ligação dinâmica do código do método
  - o código no formato .class para um método usa referência simbólica para os métodos que serão chamados e para as variáveis a serem acessadas.
    - ligação dinâmica traduz as referências simbólicas
      - em referências concretas a métodos
        - se necessário, classes são carregadas para resolver os símbolos ainda não resolvidos
      - em offsets apropriados em estruturas de armazenamento com a localização em tempo de execução das variáveis
  - essa ligação tardia dos métodos e variáveis tornam alterações em outras classes que um método usa menos prováveis de quebrar o código desse método.

Frames
Término Normal de Chamada de Método

- **O término normal ocorre se a execução do método não provocar o lançamento de exceção**
  - **um valor pode ser retornado para o método chamador**
    - **o método chamado executa uma instrução de retorno**
      - **com o tipo apropriado para o tipo do valor sendo retornado**
        - **se nenhum valor for retornado, executa** return **apenas.**
  - **o frame corrente é usado para restaurar o contexto para o método chamador**
    - **inclui suas variáveis locais, pilha de operandos e o pc do método chamador atualizado para apontar para a próxima instrução seguinte a invocação do método chamado.**
  - **execução continua normalmente no frame do método chamador com o valor retornado (se existente) empilhado na pilha de operandos desse frame.**

Frames

- **Término Abrupto de Chamada de Método**
  - **O término abrupto ocorre se a execução do método provocar o lançamento de uma exceção que não é manipulada dentro do método**
    - **um método que termina abruptamente não pode retornar um valor para o método chamador**
      - **O processamento é encerrado**
- **Informação adicional**
  - **Um frame pode ser estendido com informação adicional específica tal como informação de debug**

Representação de Objetos

- **A JVM não determina nenhuma estrutura particular interna para representar objetos**
  - **implementação da Sun para a JVM**
    - **uma referência para uma instância de classe é um ponteiro para um *handler* que é um par de ponteiros**
      - **um ponteiro para a tabela de métodos do objeto e para a classe que representa o tipo do objeto.**
      - **um ponteiro para a memória alocada para os dados do objeto a partir do heap.**

Interpretador

- **Instrução da JVM**
  - **Ignorando exceções, o ciclo interno de um interpretador da JVM pode ser descrito como;**

```
do {
   busque um opcode;
   if (∃ operandos) busque operandos;
   execute a ação associada ao opcode;
} while (! fim);
```

Conjunto de Instruções

- **Instrução da JVM**
  - **um byte de *opcode* especificando a operação**
    - **zero ou mais *operandos* que representam argumentos ou dados que serão usados na operação.**
      - **o número e tamanho dos operandos são determinados pelo opcode.**
      - **se o operando é maior que um byte em tamanho, é armazenado em *big-endian* ordem (byte mais significativo primeiro).**
        - **para construir operando de 16 bits use (byte1 << 8) | byte2**
  - **alinhadas por byte**
    - **Exceto *tableswitch* (acessa tabela de saltos por índice e desvia) e *lookupswitch* (acessa tabela de saltos por batimento de chave e desvia)**
      - **utilizam 0 a 3 bytes (NULOS) como preenchimento (padding)**
        - **alinha o primeiro operando em fronteira de 4 bytes**

Qual a razão de apenas um byte para código de operação? Como a JVM sabe quais os tipos dos operandos?

Conjunto de Instruções

- **Tableswitch**

```
oxAA                /* opcode */
0 a 3 bytes         /* alinha 4 em 4 bytes do início do método */
32 bits signed      /* default – usado para calcular target default */
32 bits signed      /* low – índice inferior da tabela de offsets */
32 bits signed      /* high – índice superior da tabela de offsets */
x offsets           /* x = high–low+1 offsets de 32 bits com sinal */
```

  - **Comentários**
    - **use (byte1 << 24) | (byte2 << 16) | (byte3 << 8) | byte4 para construir um offset de 32 bits com sinal. Os bytes são considerados sem sinais.**
    - **operação na pilha de operandos**

      .., index ⇒ ...
    - **se *index* < *low* ou *index* > *high* então *target* = *default* + offset do opcode da instrução *tableswitch***
    - **senão *target* = offset[index-low] + offset do opcode da instrução *tableswitch***
    - ***target* é um offset em relação ao MÉTODO que contém a instrução *tableswitch***

index é do tipo inteiro.

Conjunto de Instruções Tableswitch

.., index ⇒ ...
***tableswitch* opcode**
<0-3 byte pad> **/* alinha operandos a seguir em fronteira de 4 bytes */**
defaultbyte1
defaultbyte2
defaultbyte3
defaultbyte4
lowbyte1
lowbyte2
lowbyte3
lowbyte4
highbyte1
highbyte2
highbyte3
highbyte4
jump offsets...

Próxima instrução a ser executada tem endereço target + endereço da instrução atual.

Conjunto de Instruções

- **Lookupswitch**

  oxAB /* opcode */
  0 a 3 bytes /* alinha 4 em 4 bytes do início do método */
  32 bits signed /* default – usado para calcular *target* default */
  32 bits signed /* npairs – número de pares da tabela de offset */
  npairs pares /* cada par tem a forma <match><offset> */

  - Comentários
    - use (byte1 << 24) | (byte2 << 16) | (byte3 << 8) | byte4 para construir um valor de 32 bits. Os bytes são considerados sem sinais.
    - *npairs* é inteiro maior ou igual a zero. *match* e *key* são do tipo inteiros.
    - a tabela de pares é ordenada por *match* crescente.
    - operação na pilha de operandos

      .., key ⇒ ...
    - se *key* não casa com nenhum *match* então *target* = *default* + offset do opcode da instrução *lookupswitch*
    - senão *target* = offset associado ao *match* casado + offset do opcode da instrução *lookupswitch*
    - *target* é um offset em relação ao MÉTODO que contém a instrução *lookupswitch*

Conjunto de Instruções Lookupswitch

..., *key* ⇒ ...

lookupswitch opcode

<0-3 byte pad> **/* alinha operandos a seguir em fronteira de 4 bytes */**

defaultbyte1

defaultbyte2

defaultbyte3

defaultbyte4

npairs1

npairs2

npairs3

npairs4

pares *match-offset*

- **são pares de 4 bytes cada, ordenados por** match

Próxima instrução a ser executada tem endereço target + endereço da instrução atual.

Conjunto de Instruções

- **Tipos e a JVM**
  - **Em geral o tipo dos operandos é explicitado no mnemônico da instrução por uma letra de prefixo**
    - ***i*** *int* ***l*** *long* ***s*** *short* ***b*** *byte* ***c*** *char* ***f*** *float* ***d*** *double*
    - ***a***

      representa o tipo de referência
  - **Instruções com tipo não ambíguo não possuem prefixo de tipo**
    - **por exemplo, *arraylength* opera em objeto do tipo array**

      ..., *arrayref* ⇒ ..., *length*
  - **Instruções que não operam em operandos com tipos**
    - **por exemplo, *goto***

      goto branchbyte1 branchbyte2

      ... ⇒ ...

O conjunto de instruções da JVM não é ortogonal. Se fosse, apenas um byte de opcode seria suficiente?

Conjunto de Instruções

- ***Template* de instrução com tipo**
  - **Uma instrução específica com informação de tipo é construída substituindo o T no template pela letra na coluna de tipo.**
    - **se a célula determinada por um coluna de tipo e uma linha de *template* for vazia, não existe instrução que suporte aquele tipo de operação.**
      - **por exemplo, load de** int **é** iload**. Não existe um load de** byte**.**
    - **para a maioria das instruções não existe uma forma para os tipos integrais** byte**,** char **e** short.
      - **não existe instrução com operando** boolean
        - byte**, e** short **são tratados como** int **com extensão de sinal**
        - char **e** boolean **são tratados como** int **com extensão de zero**

Sumário do Conjunto de Instruções

opcode	*byte*	*short*	*int*	*long*	*float*	*double*	*char*	*reference*
Tipush	*bipush*	*sipush*						
Tconst			*iconst*	*lconst*	*fconst*	*dconst*		*aconst*
Tload			*iload*	*lload*	*fload*	*dload*		*aload*
Tstore			*istore*	*lstore*	*fstore*	*dstore*		*astore*
Tinc			*iinc*					
Taload	*baload*	*saload*	*iaload*	*laload*	*faload*	*daload*	*caload*	*aaload*
Tastore	*bastore*	*sastore*	*iastore*	*lastore*	*fastore*	*dastore*	*castore*	*aastore*
Tadd			*iadd*	*ladd*	*fadd*	*dadd*		
Tsub			*isub*	*lsub*	*fsub*	*dsub*		
Tmul			*imul*	*lmul*	*fmul*	*dmul*		
Tdiv			*idiv*	*ldiv*	*fdiv*	*ddiv*		
Trem			*irem*	*lrem*	*frem*	*drem*		
Tneg			*ineg*	*lneg*	*fneg*	*dneg*		
Tshl			*ishl*	*lshl*				

Sumário do Conjunto de Instruções

opcode	*byte*	*short*	*int*	*long*	*float*	*double*	*char*	*reference*
Tshr			*ishr*	*lshr*				
Tushr			**iushr**	**lushr**				
Tand			*iand*	*land*				
Tor			*ior*	*lor*				
Txor			*ixor*	*lxor*				
i2T	*i2b*	*i2s*		*i2l*	*i2f*	*i2d*		
l2T			*l2i*		*l2f*	*l2d*		
f2T			*f2i*	*f2l*		*f2d*		
d2T			*d2i*	*d2l*	*d2f*			
Tcmp				*lcmp*				
Tcmpl					*fcmpl*	*dcmpl*		
Tcmpg					*fcmpg*	*dcmpg*		
if_TcmpOP			*if_icmpOP*					*if_acmpOP*
Treturn			*ireturn*	*lreturn*	*freturn*	*dreturn*		*areturn*

Mapeamento dos Tipos de Java na JVM

Java	JVM	Categoria
boolean	int	**1**
byte	int	**1**
char	int	**1**
short	int	**1**
int	int	**1**
float	float	**1**
reference	reference	**1**
	returnAddress	**1**
long	long	**2**
double	double	**2**

Intruções e Tipos de Operandos

- **Normalmente o opcode determina o tipo dos operandos**
  - **algumas instruções operam na pilha limitadas por categoria e não por tipo**

pop

..., *value* ⇒ ... **/* *value* deve ser da categoria 1 */**

pop2

..., *value2*, *value1* ⇒ ... **/*** value2 **e** value1 **categoria 1 */**

..., *value* ⇒ ... **/*** value **deve ser da categoria 2 */**

swap

..., *value2*, *value1* ⇒ ..., *value1*, *value2* **/* somente categoria 1 */**

Instruções de Carga e Armazenamento

- **Transferem valores entre o array de variáveis locais e a pilha de operandos de um frame de um método**
  - **carrega uma variável local ⇒ pilha de operandos**

    iload, *iload_<n>*, *lload*, *lload_<n>*, *fload*, *fload_<n>*, *dload*, *dload_<n>*, *aload*, *aload_<n>*
  - **carrega um valor da pilha de operandos ⇒ vetor de variáveis locais**

    istore, *istore_<n>*, *lstore*, *lstore_<n>*, *fstore*, *fstore_<n>*, *dstore*, *dstore_<n>*, *astore*, *astore_<n>*
  - **carrega constante na pilha de operandos**

    bipush, *sipush*, *ldc*, *ldc_w*, *ldc2_w*, *aconst_null*, *iconst_m1*, *iconst_<i>*, *lconst_<l>*, *fconst_<f>*, *dconst_<d>*

Instruções de Carga e Armazenamento

- **Transferem valores entre o array de variáveis locais e a pilha de operandos de um frame de um método**
  - **carrega uma variável local ⇒ pilha de operandos**

    iload, *iload_<n>*, *lload*, *lload_<n>*, *fload*, *fload_<n>*, *dload*, *dload_<n>*, *aload*, *aload_<n>*

    - **Comentários**
      - **Instruções com letra entre bicudos denotam famílias e geram instruções específicas com operandos implícitos**
        - **<*n*> – inteiro não negativo no intervalo [0,3]**
        - **<*i*> –** int, **<*l*> –** long,
        - **<*f*> –** float**, <*d*> –** double**.**
        - **formas para o tipo** int **são usadas com valores tipo** byte**,** char **e** short

Instruções de Carga e Armazenamento Exemplos

- **Dado um índice para uma variável local** int, **a carrega na pilha de operandos**

iload index

iload **= 21 (0x15)**

Descrição

O index é um unsigned byte **que deve ser um índice do array de variáveis locais do frame corrente.**

A variável local no índice deve conter um int**. O valor da variável local no índice é empilhado na pilha de operandos.**

O opcode iload pode ser usado em conjunto com a instrução wide **para acessar uma variável local usando um índice de 2 bytes**

- **Dado um índice implícito, carrega uma variável local na pilha de operandos**

iload_**<n>**

iload_0 **= 26 (0x1a)**
iload_1 **= 27 (0x1b)**
iload_2 **= 28 (0x1c)**
iload_3 **= 29 (0x1d)**

Descrição

O <*n*> deve ser um índice no array de variáveis locais do frame corrente. A variável local em <*n*> deve conter um int**. O valor da variável local em <*n*> é empilhado na pilha de operandos.**

Cada instrução *iload_<n>* é a mesma que *iload* com um *index* of <*n*>, exceto que o operando <*n*> é implícito.

Instruções de Carga e Armazenamento Exemplos

- **Dado um índice para uma variável local** long, **a carrega na pilha de operandos**

lload index

lload **= 22 (0x16)**

Descrição

O index é um unsigned byte **que deve ser um índice do array de variáveis locais do frame corrente. A variável local no índice e índice + 1 deve conter um** long**. O valor da variável local no índice é empilhado na pilha de operandos.**

O opcode lload pode ser usado em conjunto com a instrução wide **para acessar uma variável local usando um índice de 2 bytes**

- **Dado um índice implícito, carrega uma variável local na pilha de operandos**

lload_**<n>**

lload_0 **= 30 (0x1e)**
lload_1 **= 31 (0x1f)**
lload_2 **= 32 (0x20)**
lload_3 **= 33 (0x21)**

Descrição

O <*n*> deve ser um índice no array de variáveis locais do frame corrente. A variável local em <*n*> deve conter um long**. O valor da variável local em <*n*> é empilhado na pilha de operandos.**

Cada instrução *lload_<n>* é a mesma que *lload* com um *index* of <*n*>, exceto que o operando <*n*> é implícito.

Instruções de Carga e Armazenamento Exemplos

- **Dado um índice para uma variável local de referência, a carrega na pilha de operandos**

aload index

aload = 25 (0x19)

Descrição

O index é um unsigned byte **que deve ser um índice do array de variáveis locais do frame corrente. A variável local no índice deve conter uma referência. O valor da variável local no índice é empilhado na pilha de operandos.**

Essa instrução não pode ser usada para carregar um tipo returnAddress **de uma variável local na pilha de operandos. Porque?**

O opcode aload pode ser usado em conjunto com a instrução wide **para acessar uma variável local usando um índice de 2 bytes**

- **Dado um índice implícito, carrega uma variável local na pilha de operandos**

aload_**<n>**

aload_0 **= 42 (0x2a)**
aload_1 **= 43 (0x2b)**
aload_2 **= 44 (0x2c)**
aload_3 **= 45 (0x2d)**

Descrição

O <*n*> deve ser um índice no array de variáveis locais do frame corrente. A variável local em <*n*> deve conter uma referência. O valor da variável local em <*n*> é empilhado na pilha de operandos.

Essa instrução não pode ser usada para carregar um tipo returnAddress **de uma variável local na pilha de operandos.**

Cada instrução *aload_<n>* é a mesma que *aload* com um *index* of <*n*>, exceto que o operando <*n*> é implícito.

Instruções de Carga e Armazenamento Exemplos

- **Estende o índice de uma variável local para 2 bytes**

wide <*opcode*> *indexbyte1 indexbyte2*

wide *iinc* ***indexbyte1 indexbyte2 constbyte1 constbyte2***

***wide* = 196 (0xc4)**

Descrição

O <*opcode*> é um dos seguintes: *iload*, *fload*, *aload*, *lload*, *dload*, *istore*, *fstore*, *astore*, *lstore*, *dstore* **ou** *ret*.

Os unsigned bytes ***indexbyte1* e *indexbyte2* são montados como índice de 16-bits de uma variável local no frame corrente.**

Para as instruções *lload*, *dload*, *lstore* **ou** *dstore* **o índice seguinte (index + 1) deve ser um índice para o array de variáveis locais. Os dois apontam para um valor** long **ou** double.

Na segunda forma, os unsigned byte ***constbyte1* e *constbyte2* são montados como uma constante com sinal de 16-bit, onde a constante é obtida como (*constbyte1* << 8) | *constbyte2***

Nesse caso, a variavel local dado por (index*byte1* << 8) | *indexbyte2* é incrementada pelo valor com sinal dado por (*constbyte1* << 8) | *constbyte2*

Instruções de Carga e Armazenamento

- **Transferem valores entre o array de variáveis locais e a pilha de operandos de um frame de um método**
  - **carrega um valor da pilha de operandos ⇒ variável local**

    istore, *istore_<n>*, *lstore*, *lstore_<n>*, *fstore*, *fstore_<n>*, *dstore*, *dstore_<n>*, *astore*, *astore_<n>*

    - **Comentários**
      - **Instruções com letra entre bicudos denotam famílias e geram instruções específicas com operandos implícitos**
        - **<*n*> – inteiro não negativo, <*i*> –** int, **<*l*> –** long,
        - **<*f*> –** float**, <*d*> –** double**.**
        - **Formas tipo** int **são usadas com valores tipo** byte**,** char **e** short

Instruções de Carga e Armazenamento Exemplos

- **Desempilha o** int **no topo da pilha de operandos no array de variáveis locais, na posição dada por índice para variável local** int

istore index

istore **= 54 (0x36)**

Descrição

O index é um unsigned byte **que deve ser um índice do array de variáveis locais do frame corrente. A variável local no índice deve conter um** int**. Desempilha o** int **no topo da pilha de operandos no array de variáveis locais, na posição índice.**

O opcode istore **pode ser usado em conjunto com a instrução** wide **para acessar uma variável local usando um índice de 2 bytes**

- **Desempilha o** int **no topo da pilha de operandos no array de variáveis locais, na posição dada por índice para variável local** int

istore_**<n>**

istore_0 **= 59 (0x3b)**
istore_1 **= 60 (0x3c)**
istore_2 **= 61 (0x3d)**
istore_3 **= 62 (0x3e)**

Descrição

O <*n*> deve ser um índice no array de variáveis locais do frame corrente. A variável local em <*n*> deve conter um int**. O valor da variável local em <*n*> é empilhado na pilha de operandos.**

Cada instrução *iload*_<*n*> é a mesma que *iload* com um *index* of <*n*>, exceto que o operando <*n*> é implícito.

Instruções de Carga e Armazenamento

- **Transferem valores entre o array de variáveis locais e a pilha de operandos de um frame de um método**
  - **carrega uma constante ⇒ pilha de operandos**

    bipush, *sipush*, *ldc*, *ldc_w*, *ldc2_w*, *aconst_null*, *iconst_m1*, *iconst_<i>*, *lconst_<l>*, *fconst_<f>*, *dconst_<d>*

    - **Comentários**
      - **Instruções com letra entre bicudos denotam famílias e geram instruções específicas com operandos implícitos**
        - **<*i*> –** int, **<*l*> –** long,
        - **<*f*> –** float**, <*d*> –** double**.**
        - **Formas tipo** int **são usadas com valores tipo** byte**,** char **e** short

Instruções de Carga e Armazenamento
Carrega Constante na Pilha de Operandos

bipush byte

bipush = 16 (0x10)

- **O byte imediato é estendido com sinal para um valor** int **e empilhado na pilha de operandos**

sipush byte1 byte2

***sipush* = 17 (0x11)**

- **O valor imediato** short **(*byte1* << 8) | *byte2* é estendido com sinal para um** int **e empilhado na pilha de operandos**

Instruções de Carga e Armazenamento
Carrega Constante na Pilha de Operandos

ldc ***index***

***ldc* = 18 (0x12)**

- **O *index* é um** unsigned byte **que deve ser um índice válido para o pool de constantes da classe corrente. O valor no pool de constante deve ser uma constante do tipo** int **ou** float **ou uma referência simbólica para um literal** string**.**
  - **se do tipo** int **ou** float**, o valor numérico da constante é empilhado na pilha de operandos como um** int **ou** float
  - **se uma referência para uma instância da classe String representando um literal** string**, uma referência para tal instância é empilhada na pilha de operandos.**

Instruções de Carga e Armazenamento
Carrega Constante na Pilha de Operandos

ldc_w ***indexbyte1 indexbyte2***

***ldc_w* = 19 (0x13)**

- **Os** unsigned ***indexbyte1* e *indexbyte2* são montados em um índice** unsigned **de 16-bit do pool de constantes da classe corrente, onde o valor do índice é calculado como (*indexbyte1* << 8) | *indexbyte2*. O valor no pool de constante deve ser uma constante do tipo** int **ou** float **ou uma referência simbólica para um literal** string**.**
  - **se do tipo** int **ou** float**, o valor numérico da constante é empilhado na pilha de operandos como um** int **ou** float
  - **se uma referência para uma instância da classe String representando um literal** string**, uma referência para tal instância é empilhada na pilha de operandos.**
- **É idêntica à instrução** ldc **exceto que o índice é de 16 bits.**

Instruções de Carga e Armazenamento
Carrega Constante na Pilha de Operandos

ldc2_w ***indexbyte1 indexbyte2***

***ldc2_w* = 20 (0x14)**

- **Os** unsigned ***indexbyte1* e *indexbyte2* são montados em um índice** unsigned **de 16-bit do pool de constantes da classe corrente, onde o valor do índice é calculado como (*indexbyte1* << 8) | *indexbyte2*. O valor no pool de constante deve ser uma constante do tipo** long **ou** double**.**
  - **o valor numérico da constante é empilhado na pilha de operandos como um** long **ou** double
    - **Não existe uma versão ldc2 com um índice de 8 bits.**

Instruções de Carga e Armazenamento
Carrega Constante na Pilha de Operandos

aconst_null

***aconst_null* = 1 (0x1)**

- **Empilha uma referência** null **para um objeto na pilha de operandos**

iconst_<i>

***iconst_m1* = 2 (0x2), *iconst_0* = 3 (0x3), *iconst_1* = 4 (0x4)**

***iconst_2* = 5 (0x5), *iconst_3* = 6 (0x6), *iconst_4* = 7 (0x7)**

***iconst_5* = 8 (0x8)**

- **Empilha a constante** int **<i> (-1, 0, 1, 2, 3, 4 ou 5) na pilha de operandos.**
- **equivalente a *bipush* <*i*> para o respectivo valor de <*i*>, exceto que o operando <*i*> é implícito.**

Instruções de Carga e Armazenamento
Carrega Constante na Pilha de Operandos

lconst_<l>

***lconst_0* = 9 (0x9)**

***lconst_1* = 10 (0xa)**

- **Armazena a constante** long **0L ou 1L na pilha de operandos.**

fconst_<f>

***fconst_0* = 11 (0xb)**

***fconst_1* = 12 (0xc)**

***fconst_2* = 13 (0xd)**

- **Armazena a constante** float **0.0, 1.0 ou 2.0 na pilha de operandos.**

Instruções de Carga e Armazenamento
Carrega Constante na Pilha de Operandos

dconst_<d>

***dconst_0* = 14 (0xe)**

***dconst_1* = 15 (0xf)**

- **Armazena a constante** double **0.0 ou 1.0 na pilha de operandos.**

Instruções Aritméticas

- **Computam um resultado que é tipicamente uma função de dois valores na pilha de operandos**
  - **o resultado é empilhado na pilha de operandos.**
- **Instruções aritméticas**
  - **Valores inteiros**
  - **Valores de ponto flutuantes**
  - **Não há suporte direto para operações aritméticas em valores do tipo** byte**,** short **e** char **ou** boolean
    - **essas operações são manipuladas por instruções de tipos** int

Instruções Aritméticas

- **Soma**
  iadd, *ladd*, *fadd*, *dadd*
- **Subtração**
  isub, *lsub*, *fsub*, *dsub*
- **Multiplação**
  imul, *lmul*, *fmul*, *dmul*
- **Divisão**
  idiv, *ldiv*, *fdiv*, *ddiv*
- **Resto**
  irem, *lrem*, *frem*, *drem*
- **Negate**
  ineg, *lneg*, *fneg*, *dneg*
- **Shift**
  ishl, *ishr*, *iushr*, *lshl*, *lshr*, *lushr*.
- **Bitwise AND**
  iand, *land*
- **Bitwise OR**
  ior, *lor*
- **Bitwise exclusive OR**
  ixor, *lxor*
- **Incremento em variável local**
  iinc
- **Comparação**
  dcmpg, *dcmpl*, *fcmpg*, *fcmpl*, *lcmp*

Instruções Aritméticas
Soma

- **Desempilha dois valores do topo da pilha, soma-os e empilha o resultado**

  ..., *value1, value2* ⇒ ..., *result*

  result é calculado como value1 + value2

iadd /* int */

***iadd* = 96 (0x60)**

ladd /* long */

***ladd* = 97 (0x61)**

fadd /* float */

***fadd* = 98 (0x62)**

dadd /* double */

***dadd* = 99 (0x63)**

Instruções Aritméticas
Subtração

- **Desempilha dois valores do topo da pilha, subtrai-os e empilha o resultado**

  ..., *value1, value2* ⇒ ..., *result*

  result é calculado como value1 – value2

isub — /* int */

***isub* = 100 (0x64)**

lsub — /* long */

***lsub* = 101 (0x65)**

fsub — /* float */

***fsub* = 102 (0x66)**

dsub — /* double */

***dsub* = 103 (0x67)**

Instruções Aritméticas
Multiplicação

- **Desempilha dois valores do topo da pilha, multiplica-os e empilha o resultado**

  ..., *value1, value2* ⇒ ..., *result*

  result é calculado como value1 * value2

imul — /* int */

imul = 104 (0x68)

lmul — /* long */

lmul = 105 (0x69)

fmul — /* float */

fmul = 106 (0x6a)

dmul — /* double */

dmul = 107 (0x6b)

Instruções Aritméticas
Divisão

- **Desempilha dois valores do topo da pilha, divide-os e empilha o resultado**

  ..., *value1, value2* ⇒ ..., *result*

  result é calculado como value1 / value2

idiv — /* int */

idiv = 108 (0x6c)

ldiv — /* long */

ldiv = 109 (0x6d)

fdiv — /* float */

fdiv = 110 (0x6e)

ddiv — /* double */

ddiv = 111 (0x6f)

Instruções Aritméticas
Resto

- **Desempilha dois valores do topo da pilha, divide-os como inteiros e empilha o resto**
  - **..., *value1, value2* ⇒ ..., *result***
    - **result é calculado como *value1* - (*value1* / *value2*) * *value2***
      - **ocorre truncamento na divisão value1 / value2**

irem — /* int */
irem = 112 (0x70)

lrem — /* long */
lrem = 113 (0x71)

frem — /* float */
frem = 114 (0x72)

drem — /* double */
drem = 115 (0x73)

Instruções Aritméticas
Negação

- **Desempilha um valor do topo da pilha, nega-o e empilha o resultado**
  - **..., *value* ⇒ ..., *result***
    - **result é similar a zero - value ou seja, -value.**

ineg	/* int */
***ineg* = 116 (0x74)**	
lneg	/* long */
***lneg* = 117 (0x75)**	
fneg	/* float */
***fneg* = 118 (0x76)**	
dneg	/* double */
***dneg* = 119 (0x77)**	

Instruções Aritméticas
Shift (Deslocamento)

- **Desempilha um valor do topo da pilha, gira-o e empilha o resultado**
  - **..., *value1, value2* ⇒ ..., *result***
    - result **é calculado como deslocar** value1 **por** s **bits onde** s **é o valor dos 5 bits de ordem mais baixa de** value2**. Para** long**,** s **é o valor dos 6 bits de ordem mais baixa de** value2**.**

ishl **ou** lshl **/*** int **ou** long **para esquerda */**

***ishl* = 120 (0x78)** **e** ***lshl* = 121 (0x79)**

ishr **ou** lshr **/*** int **ou** long **para direita */**

***ishr* = 122 (0x7a)** **e** ***lshr* = 123 (0x7b)**

iushr **ou** lushr **/*** int **ou** long **sem sinal direita */**

iushr = 124 (0x7c) **e** ***lushr* = 125 (0x7d)**

- **Insere zeros à esquerda**

Instruções Aritméticas
Instruções Lógicas Bit a Bit

- **Desempilha dois valores do topo da pilha, aplica o operador lógico bit a bit e empilha o resultado. Opera somente com** int **ou** long

  ..., *value1, value2* ⇒ ..., *result*

  - **Bitwise AND (**iand **e** land**)**
    iand = 126 (0x7e)
    land = 127 (0x7f)
  - **Bitwise OR (**ior **e** lor**)**
    ior = 128 (0x80)
    lor = 129 (0x81)
  - **Bitwise exclusive OR (**ixor **e** lxor**)**
    ixor = 130 (0x82)
    lxor = 131 (0x83)

Instruções Aritméticas
Incremento em Variável Local

- **Incrementa variável local identificada por índice por uma constante. Não afeta a pilha de operandos**

***iinc* = 132 (0x84)**

unsigned byte **/* index para o vetor variável local */**

signed byte **/* constante a somar à variável */**

- **Comentários**
  - **A variável no vetor de variáveis locais deve ser do tipo** int
  - **Pode ser combinado com wide para criar um índice de 16 bits e constante de 16 bits com sinal**

Instruções Aritméticas
Comparação de float e double

..., *value1*, *value2* ⇒ ..., *result*

- **Desempilha** value2 **e** value1 **do topo da pilha, compara-os e armazena o** result **do tipo** int
  - **1 se** value1**>**value2**, 0 se** value1**=**value2 **ou -1 se** value1**<**value2
- value1 **e** value2 **são** float
  - *fcmpl* = 149 (0x95) **e** *fcmpg* = 150 (0x96)
- value1 **e** value2 **são** doubles
  - *dcmpl = 151 (0x97)* ***e*** *dcmpg = 152 (0x98)*
- **Se ao menos um é NaN**
  - dcmpg **e** fcmpg **empilham 1**
  - dcmpl **e** fcmpl **empilham -1**

Instruções Aritméticas
Comparação de long

..., *value1*, *value2* ⇒ ..., *result*

lcmp = 148 (0x94)

- **Desempilha** value2 **e** value1 **do topo da pilha, compara-os e armazena o** result **do tipo** int
  - **1 se** value1**>**value2
  - **0 se** value1**=**value2
  - **-1 se** value1**<**value2
- value1 **e** value2 **são** long

Para comparação de int existe a instrução if_icmp<cond>

Instruções Aritméticas
Comparação de int com zero

..., *value1* ⇒ ...

If<cond> branchbyte1 branchbyte2

ifeq = 153 (0x99) *ifne* = 154 (0x9a)

iflt = 155 (0x9b) *ifge* = 156 (0x9c)

ifgt = 157 (0x9d) *ifle* = 158 (0x9e)

- **Desempilha** value1 **(do tipo** int**) do topo da pilha, compara com zero e desvia se resultar** true **de acordo com a operação**
  - O offset deve estar contido no método que contém a instrução de desvio.
    - Se retornar false, retoma o processamento na próxima instrução

Instruções Aritméticas
Comparação de dois int

..., value1, value2 ⇒ ...

If_icmp<cond> branchbyte1 branchbyte2

if_icmpeq = 159 (0x9f) *if_icmpne* = 160 (0xa0)
if_icmplt = 161 (0xa1) *if_icmpge* = 162 (0xa2)
if_icmpgt = 163 (0xa3) *if_icmple* = 164 (0xa4)

Desempilha value1 **e** value2 **do topo da pilha, compara-os e desvia se resultar** true **de acordo com a operação**

- O offset deve estar contido no método que contém a instrução de desvio.
  - Se retornar false, retoma o processamento na próxima instrução

Instruções Aritméticas
Comparação de dois objetos

..., value1, value2 ⇒ ...

If_acmp<cond> branchbyte1 branchbyte2

if_acmpeq = 165 (0xa5)

if_acmpne = 166 (0xa6)

Desempilha value1 **e** value2 **do topo da pilha compara-os e desvia se resultar** true **de acordo com a operação.** value1 **e** value2 **são referências.**

- O offset deve estar contido no método que contém a instrução de desvio.
  - Se retornar false, retoma o processamento na próxima instrução

Instruções de Conversão de Tipo Promoção

..., *value* ⇒ ..., *result*

- int para long, float ou double
  - *i2l = 133 (0x85)*
  - *i2f = 134* (0x86)
  - *i2d = 135 (0x87)*
- long para float ou double
  - *l2f = 137 (0x89)*
  - *l2d = 138 (0x8a)*
- float para double
  - *f2d = 141 (0x8d)*

Instruções de Conversão de Tipo Estreitamento

..., *value* ⇒ ..., *result*

- Despreza os bits superiores e pode mudar o sinal

- int para byte, char ou short

  i2b = 145 (0x91) /* estende para inteiro com sinal */

  i2c = 146 (0x92) /* estende para inteiro sem sinal */

  i2s = 147 (0x93) /* estende para inteiro com sinal */

- long para int

  l2i = 136 (0x88)

- float para int ou long

  f2i = 139 (0x8b) *f2l* = 140 (0x8c)

- double para int, long or float

  d2i = 142 (0x8e) *d2l* = 143 (0x8f) *d2f* = 144 (0x90)

Instruções de Criação e Manipulação de Objetos

- Criar uma nova instância de classe
  new
- Criar um novo array
  newarray, *anewarray*, *multianewarray*
- Acessar fields de classes ou de instâncias de classes
  getstatic, *putstatic, getfield*, *putfield*
- Carregar um componente de um array na pilha de operandos
  baload, *caload*, *saload*, *iaload*, *laload*, *faload*, *daload*, *aaload*.
- Armazenar um valor da pilha de operandos como um componente de um array
  bastore, *castore*, *sastore*, *iastore*, *lastore*, *fastore*, *dastore*, *aastore*.
- Obter o comprimento de um array
  arraylength.
- Verificar propriedades de instâncias de classes ou arrays
  instanceof, *checkcast*.

Criação e Manipulação de Objetos Instância de Classe

- **new indexbyte1 indexbyte2**

  187 (0xbb) **/* opcode */**

  16 bits unsigned **/* index – índice para o pool de constante */**

  - **Comentários**
    - **use (indexbyte1 << 8) | byte2 para construir um valor de 16 bits. Os bytes são considerados sem sinais.**
    - **index aponta para um nome (referência simbólica).**
    - **o tipo desse nome deve ser classe.**
    - **memória para a nova instância é alocada a partir do coletor de lixo.**
    - **as variáveis de instância são iniciadas para o valor default.**
    - **a referência *objectref* para a instância é armazenada na pilha.**
    - **operação na pilha de operandos**

      ... ⇒ ..., *objectref*

Criação e Manipulação de Objetos
Objeto

- **newarray atype**

  188 (0xbc) **/* opcode */**

  byte **/* atype – tipo do array a ser criado */**

  - **Comentários**
    - **Codificação do tipo do array**

T_BOOLEAN	4	T_CHAR	5
T_FLOAT	6	T_DOUBLE	7
T_BYTE	8	T_SHORT	9
T_INT	10	T_LONG	11

    - **memória para um array do tipo** atype **com** count **elementos é alocada a partir do coletor de lixo**
      - count **deve ser um** int **positivo**
    - **os elementos são iniciados com seus valores padrão**
    - **a referência *arrayref* para o array é armazenada na pilha**
    - **operação na pilha de operandos**

      ... count ⇒ ..., *arrayref*

Criação e Manipulação de Objetos
Objeto

- **multianewarray indexbyte1 indexbyte2 dimensions**

  197 (0xc5) **/* opcode */**

  16 bits unsigned **/* index – índice para o pool de constante */**

  unsigned byte **/* dimensions – número de dimensões */**

  - **Comentários**
    - **use (indexbyte1 << 8) | byte2 para construir um valor de 16 bits. Os bytes são considerados sem sinais.**
    - **index aponta para um nome (referência simbólica).**
    - **o tipo desse nome deve ser classe, array ou interface**
    - **memória para um array desse tipo com** $count_i$ **elementos na i-dimensão é alocada a partir do coletor de lixo**
      - **todos os** count **deve ser** int **positivo maiores ou iguais a um**
    - **os elementos são iniciados com** null
    - **a referência *arrayref* para o array é armazenada na pilha**
    - **operação na pilha de operandos**

      ... count1, [count2, ...] ⇒ ..., *arrayref*

Criação e Manipulação de Objetos
Acessar Field de Classe

- **getstatic indexbyte1 indexbyte2**

  178 (0xb2) **/* opcode */**

  16 bits unsigned **/* index – índice para pool de constante */**

  - **Comentários**
    - **use (indexbyte1 << 8) | byte2 para construir um valor de 16 bits. Os bytes são considerados sem sinais.**
    - **index aponta para um nome (referência simbólica) para um** field **que dá o nome e descritor do** field **assim como a referência simbólica para a classe ou interface que o contém.**
    - **a classe ou interface é iniciada (se não já o foi) e o valor do** field **é obtido e inserido na pilha de operandos**
    - **operação na pilha de operandos**

      ... ⇒ ..., *value*

Criação e Manipulação de Objetos
Acessar Field de Classe

- **putstatic indexbyte1 indexbyte2**

  179 (0xb3) **/* opcode */**

  16 bits unsigned **/* index – índice para pool de constante */**

  - **Comentários**
    - **use (indexbyte1 << 8) | byte2 para construir um valor de 16 bits. Os bytes são considerados sem sinais.**
    - **index aponta para um nome (referência simbólica) para um** field **que dá o nome e descritor do** field **assim como a referência simbólica para a classe ou interface que o contém.**
    - **a classe ou interface é iniciada (se não já o foi) e o valor do** field **é iniciado com** value **retirado da pilha de operandos**
      - **se o tipo do descritor do field for boolean, byte, char, short ou int então *value* deve ser um int.**
      - **se o tipo do descritor for float, long ou double então *value* deve ser um float, long ou double, respectivamente.**
    - **operação na pilha de operandos**

      ..., *value* ⇒ ...

Criação e Manipulação de Objetos
Acessar Field de Instância

- **getfield indexbyte1 indexbyte2**

  180 (0xb4) **/* opcode */**

  16 bits unsigned **/* index – índice para pool de constante */**

  - **Comentários**
    - Objectref, **referência para o objeto que contém o field, é retirada da pilha de operandos.**
    - **use (indexbyte1 << 8) | byte2 para construir um valor de 16 bits. Os bytes são considerados sem sinais.**
    - **index aponta para um nome (referência simbólica) para um** field **que dá o nome e descritor do** field **assim como a referência simbólica para a classe que o contém.**
    - **o valor do** field **é obtido e inserido na pilha de operandos.**
    - **se** protected**,** objectref **deve ser para a classe atual ou uma de suas subclasses.**
    - **operação na pilha de operandos**

      ..., *objectref* ⇒ ..., *value*

Criação e Manipulação de Objetos
Acessar Field de Instância

- **putfield indexbyte1 indexbyte2**

  181 (0xb5) **/* opcode */**

  16 bits unsigned **/* index – índice para pool de constante */**

  - **Comentários**
    - value **e** objectref **são retirados da pilha de operandos.**
    - **use (indexbyte1 << 8) | byte2 para construir um valor de 16 bits. Os bytes são considerados sem sinais.**
    - **index aponta para um nome (referência simbólica) para um** field **que dá o nome e descritor do** field **assim como a referência simbólica para a classe que o contém.**
    - **o valor do** field **é iniciado com** value **retirado da pilha de operandos**
      - **se o tipo do descritor do field for boolean, byte, char, short ou int então *value* deve ser um int.**
      - **se o tipo do descritor for float, long ou double então *value* deve ser um float, long ou double, respectivamente.**
    - **se** protected**,** objectref **deve ser para a classe atual ou uma de suas subclasses.**
    - **operação na pilha de operandos**

      ..., *objectref*, *value* ⇒ ...

Criação e Manipulação de Objetos

- **Carregar um componente de um array na pilha de operandos**

  ***baload*, *caload*, *saload*, *iaload*, *laload*, *faload*, *daload*, *aaload*.**

- **Armazenar um valor da pilha de operandos como um componente de um array**

  ***bastore*, *castore*, *sastore*, *iastore*, *lastore*, *fastore*, *dastore*, *aastore*.**

- **Obter o comprimento de um array**

  ***arraylength*.**

- **Verificar propriedades de instâncias de classes ou arrays**

  ***instanceof*, *checkcast*.**

Criação e Manipulação de Objetos
Carregar um Elemento de um Array na Pilha

- Acessa um elemento de um array no heap e o copia para a pilha de operandos
- Essas instruções não possuem operandos
- Operação da pilha

  ..., *arrayref*, *index* ⇒ ..., *value*
- Ação

  value = arrayref [index]

Instrução	opcode	Tipo
baload		byte
caload		char
saload		short
iaload		int
laload		long
faload		float
daload		double
aaload		reference

Outras Instruções

- **Gerenciamento da pilha de operandos**
  - ***pop*, *pop2*, *dup*, *dup2*, *dup_x1*, *dup2_x1***
    - **Desempilha topo (cat 1): ..., *value* ⇒ ...,**
    - **Desempilha topo (cat 2): ..., *value* ⇒ ..., ou se topos forem cat 1 *value1, value2* ⇒ ...,**
    - **Duplica o topo (cat 1): ..., *value* ⇒ ..., *value, value***
    - **Duplica o topo (cat 2): ..., *value* ⇒ ..., *value, value* ou se topos forem cat 1 ..., *value2, value1* ⇒ ..., *value2, value1, value2, value1***
    - **Duplica o topo e insere dois slots abaixo (topos são cat 1)**
      ..., *value2, value1* ⇒ ..., *value1, value2, value1*
    - **Duplica o(s) topo(s) e insere dois ou três slots abaixo**
      ..., *cat1, cat2* ⇒ ..., *cat2, cat1, cat2*
      ..., value3, *value2, value1* ⇒ ..., *value2, value1, value3, value2, value1*
      onde value3, *value2, value1* são cat 1

Outras Instruções

- **Gerenciamento da pilha de operandos**
  - ***dup_x2*, *dup2_x2*, *swap***
    - Duplica o topo e insere dois ou três slots abaixo
      ..., value3, *value2, value1* ⇒ ..., *value1, value3, value2, value1*
      onde value3, *value2, value1* são cat 1
      ..., *cat2, cat1* ⇒ ..., *cat1, cat2, cat1*
    - Duplica o(s) topo(s) e insere dois, três ou quatro slots abaixo
      ..., value4, value3, *value2, value1* ⇒ ..., value2, *value1, value4, value3, value2, value1*
      onde value3, *value2, value1* são cat 1
      ..., *value3*, *value2, cat2* ⇒ ..., *cat2, value3, value2, cat2*
      onde *value3* e *value2* são cat 1
      ..., *cat2*, *value2, value1* ⇒ ..., value2, *value1, cat2, value2, value1*
      onde *value2, value1* são cat 1
      ..., *value2, value1* ⇒ ..., *value1, value2, value1*
      onde *value2 e value1* são cat 2
    - Troca os elementos do topo da pilha (cat 1 apenas)

Outras Instruções

- **Transferência de controle**
  - **Desvio condicional**
    - ***ifeq*, *iflt*, *ifle*, *ifne*, *ifgt*, *ifge*, *ifnull*, *ifnonnull*, *if_icmpeq*, *if_icmpne*, *if_icmplt*, *if_icmpgt*, *if_icmple*, *if_icmpge*, *if_acmpeq*, *if_acmpne*.**
  - **Desvio condicional composto**
    - ***tableswitch*, *lookupswitch*.**
  - **Desvio incondicional**
    - ***goto*, *goto_w*, *jsr*, *jsr_w*, *ret*.**

Outras Instruções

- **Chamada de métodos e instruções de retorno**
  - ***invokevirtual***
    - **Chama um método de um objeto. É a forma normal de chamar um metodo em Java.**
  - ***invokeinterface***
    - **Chama um metodo que é implementado por uma interface.**
  - ***invokespecial***
    - **Chama um método de instância requerendo tratamento especial (método de iniciação, método privado ou método de superclass)**
  - ***invokestatic***
    - **Chama um método de classe (static).**

Outras Instruções

- **Chamada de métodos e instruções de retorno**
  - ***invokevirtual* = 182 (0xb6)**
    - **Formato**

      invokevirtual indexbyte1 indexbyte2
    - **Operação**

      ..., objectref, [arg1, [arg2 ...]] ⇒ *...*
    - **Descrição**
      - **O índice formado pelos dois bytes apontam para uma entrada no Constant Pool que dá o nome e o descritor do método sendo chamado, o que define o número de argumentos que possui. O método não pode ser um método construtor. Esse método deve ser da classe do objeto *objecref* ou de super classe. Os argumentos são retirados da pilha e um frame é construído para executar esse método com os argumentos passados.**

Sumário das Instruções da JVM

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	nop	aconst_null	iconst_m1	iconst_0	iconst_1	iconst_2	iconst_3	iconst_4	iconst_5	lconst_0	lconst_1	fconst_0	fconst_1	fconst_2	dconst_0	dconst_1
1	bipush	sipush	ldc	ldc_w	ldc2_w	iload	lload	fload	dload	aload	iload_0	iload_1	iload_2	iload_3	lload_0	lload_1
2	lload_2	lload_3	fload_0	fload_1	fload_2	fload_3	dload_0	dload_1	dload_2	dload_3	aload_0	aload_1	aload_2	aload_3	iaload	laload
3	faload	daload	aaload	baload	caload	saload	istore	lstore	fstore	dstore	astore	istore_0	istore_1	istore_2	istore_3	lstore_0
4	lstore_1	lstore_2	lstore_3	fstore_0	fstore_1	fstore_2	fstore_3	dstore_0	dstore_1	dstore_2	dstore_3	astore_0	astore_1	astore_2	astore_3	iastore
5	lastore	fastore	dastore	aastore	bastore	castore	sastore	pop	pop2	dup	dup_x1	dup_x2	dup2	dup2_x1	dup2_x2	swap
6	iadd	ladd	fadd	dadd	isub	lsub	fsub	dsub	imul	lmul	fmul	dmul	idiv	ldiv	fdiv	ddiv
7	irem	lrem	frem	drem	ineg	lneg	fneg	dneg	ishl	lshl	ishr	lshr	iushr	lushr	iand	land

Sumário das Instruções da JVM

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
8	ior	lor	ixor	lxor	iinc	i2l	i2f	i2d	l2i	l2f	l2d	f2i	f2l	f2d	d2i	d2l
9	d2f	i2b	i2c	i2s	lcmp	fcmpl	fcmpg	dcmpl	dcmpg	ifeq	ifne	iflt	ifge	ifgt	ifle	if_icmpeq
a	if_icmpne	if_icmplt	if_icmpge	if_icmpgt	if_icmple	if_acmpeg	if_acmpne	goto	jsr	ret	table switch	lookup switch	ireturn	lreturn	freturn	dreturn
b	areturn	return	get static	put static	getfield	putfield	invoke virtual	invoke special	invoke static	invokeinterface		new	new array	anew array	array length	athrow
c	check cast	instance of	monitor enter	monitor exit	wide	multi anew array	ifnull	ifnon null	goto_w	jsr_w	break point					
d																
e																
f															impdep1	impdep2

Os três opcode em destaque são reservados pela Sun.