



Curso Java Básico

Uma introdução prática usando
BlueJ



Agrupando objetos

Introdução a Coleções

Principais conceitos a serem abordados

- Coleções
- Loops
- Iteradores
- Arrays
- Objetos anônimos
- Encadeamento de chamadas de métodos

O requisito para agrupar objetos

- Várias aplicações envolvem coleções de objetos:
 - agendas pessoais;
 - catálogos de bibliotecas; e
 - sistema de registro de alunos.
- O número de itens armazenados varia.
 - Itens adicionados.
 - Itens excluídos.

Um bloco de notas

- Modelaremos uma aplicação de anotações pessoais com os recursos:
 - Notas podem ser armazenadas.
 - Notas individuais podem ser visualizadas.
 - Não há um limite para o número de notas.
 - A quantidade de notas armazenadas deve ser informada.

Exercício

- **Bloco de notas pessoal**

- Inicie o **BlueJ**, abra o projeto *notebook1* e crie uma instância de *Notebook*.
- Armazene algumas notas usando o método *storeNote*. Chame *numberOfNotes* e verifique se o número retornado corresponde à quantidade de notas armazenadas.
- Chame o método *showNote* informando 0 (zero) para imprimir a primeira nota, 1 (um) para a segunda e assim por diante (mostraremos depois a razão).⁶

Bibliotecas de classes

- Acervo de classes úteis.
- Não temos de escrever tudo a partir do zero.
- O Java chama suas bibliotecas de *pacotes*.
- Agrupar objetos é um requisito recorrente.
 - O pacote `java.util` contém as classes para fazer isso.

A instrução *import*

- A instrução `import` permite se referir a tipos (classes, interfaces, etc) de outros pacotes sem necessidade de usar seu nome qualificado.

- nome qualificado:

```
graphics.Rectangle myRect = new graphics.Rectangle();
```

- `import`:

```
import graphics.Rectangle;
```

```
...
```

```
Rectangle myRect = new Rectangle();
```


Pacotes e importação

- Podemos importar classes individuais
import java.util.ArrayList;
import java.util.Random;
- Podemos importar o pacote completo
import java.util.;*
- Classes do pacote `java.lang` não precisam ser importadas.

Código-fonte: Notebook

```
import java.util.ArrayList;  ← Instrução de importação
/**
 * ...
 */
public class Notebook
{
    // Campo para um número arbitrário de notas.
    private ArrayList<String> notes;
    /**
     * Realiza qualquer inicialização
     * necessária para o notebook.
     */
    public Notebook()
    {
        notes = new ArrayList<String>();
    }
    ...
}
```

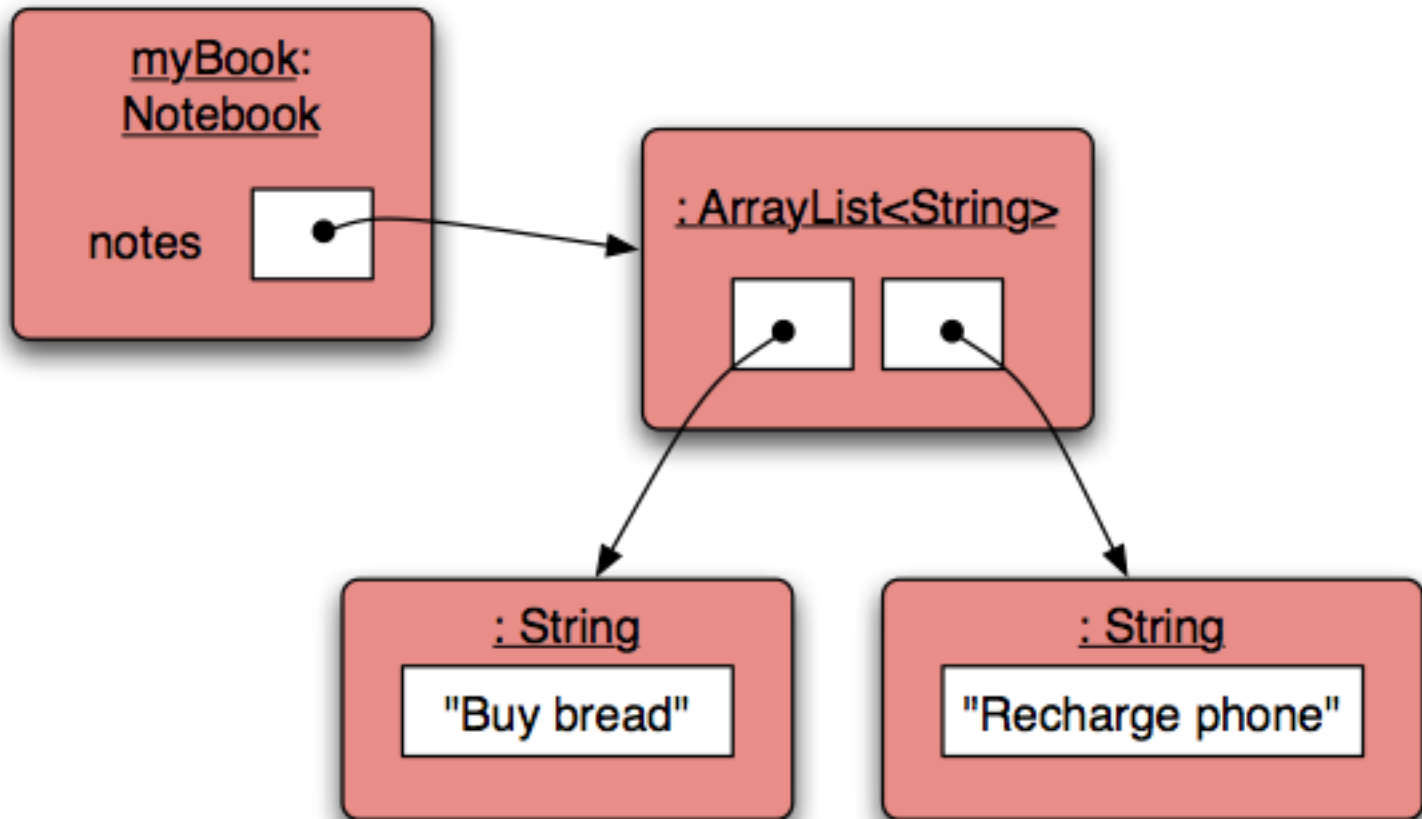
Classe genérica ou
parametrizada



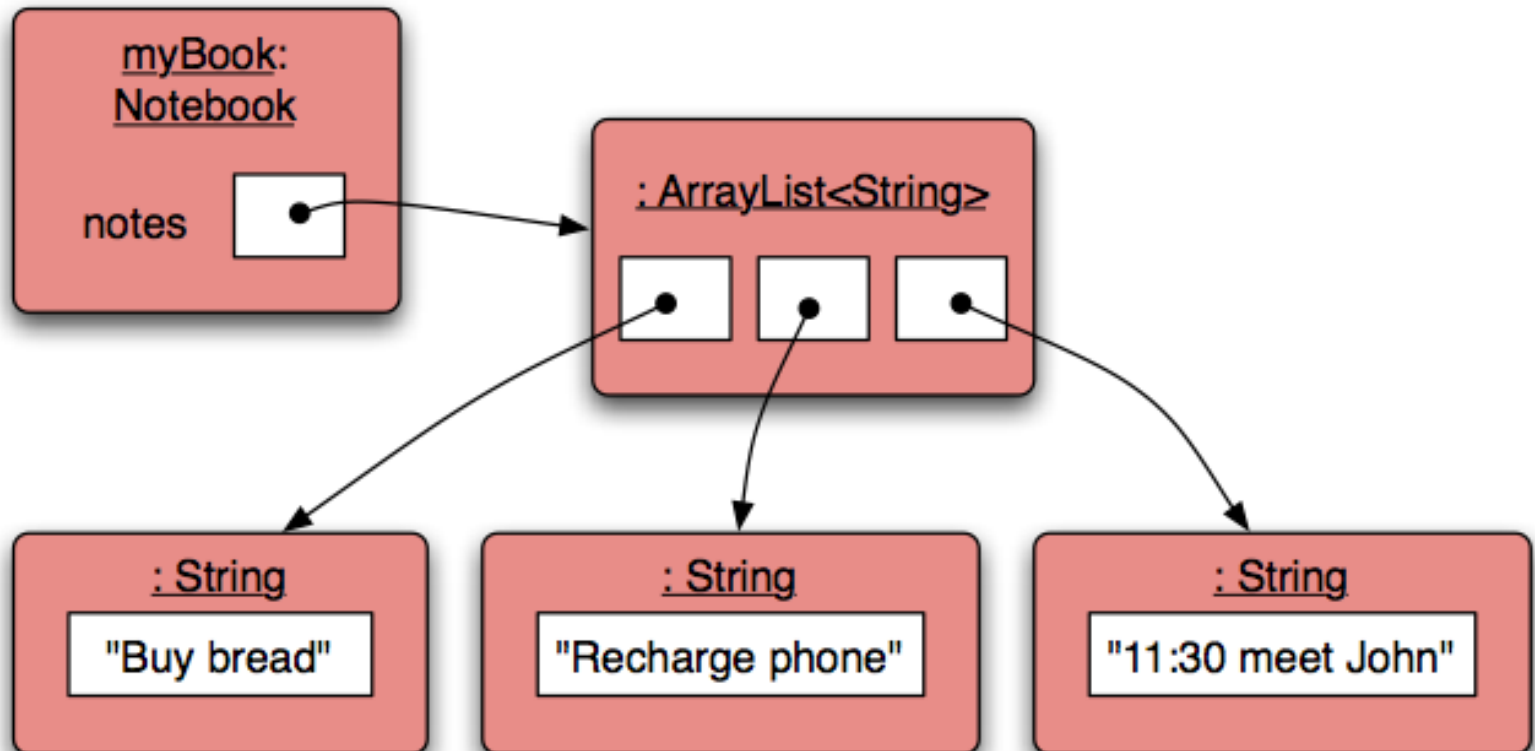
Coleções flexíveis

- Nós especificamos:
 - O tipo de coleção: `ArrayList`
 - O tipo de objetos que ela conterá:
`<String>`
- Nós dizemos, “`ArrayList de String`”.
- Coleções são conhecidas como tipos *parametrizados* or *genéricos*.

Estruturas de objetos com coleções



Adicionando uma terceira nota



Recursos da coleção

- Ela aumenta a capacidade interna conforme necessário.
- Mantém uma contagem privada (método de acesso `size()`).
- Mantém os objetos em ordem.
- Detalhes sobre como tudo isso é feito são ocultados.
 - Isso importa? Não saber como é feito nos impede de utilizá-los?

Utilizando a coleção

```
public class Notebook
{
    private ArrayList<String> notes;
    ...

    public void storeNote(String note)
    {
        notes.add(note);

    }

    public int numberOfNotes()
    {
        return notes.size();

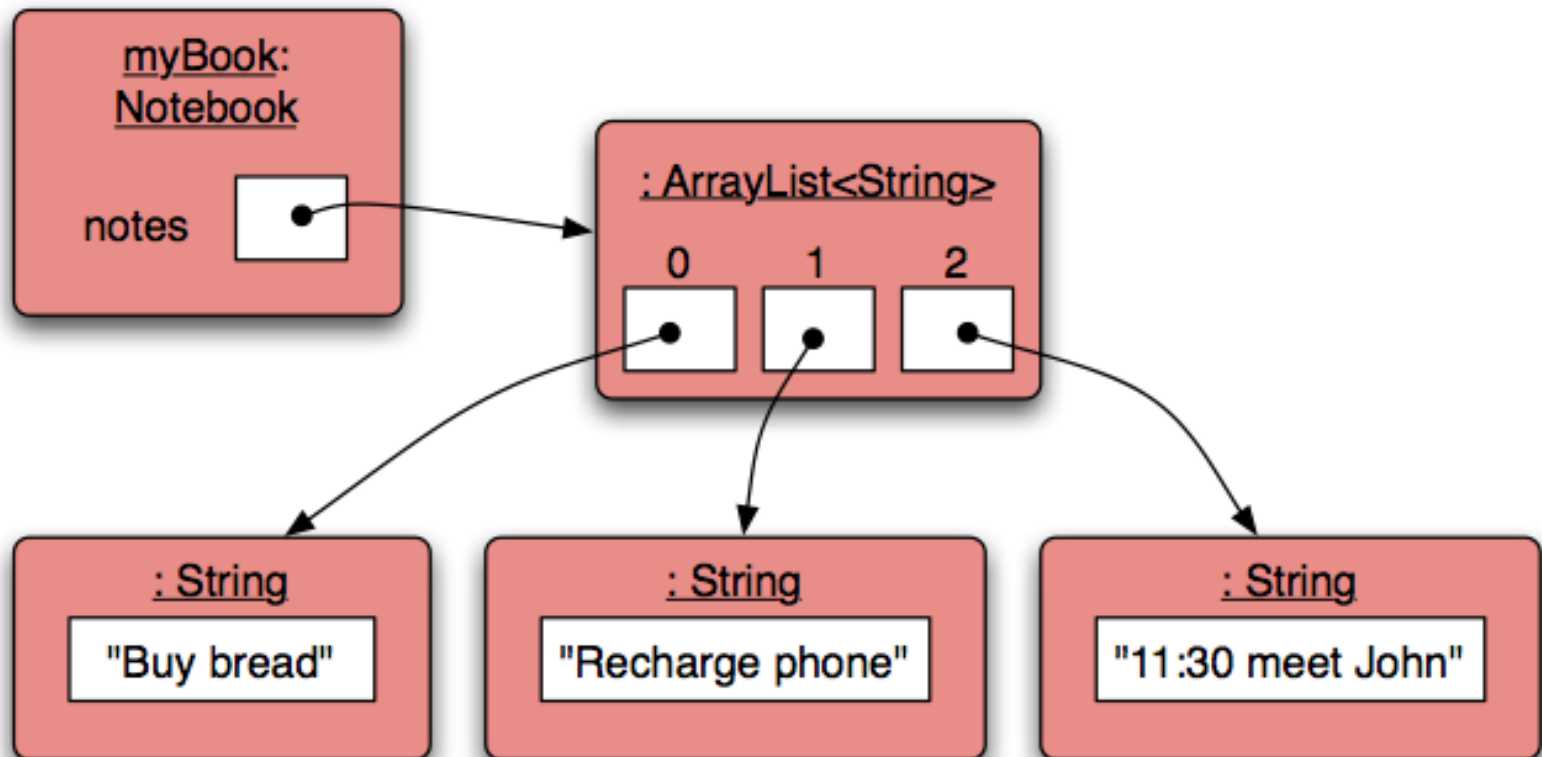
    }

    ...
}
```

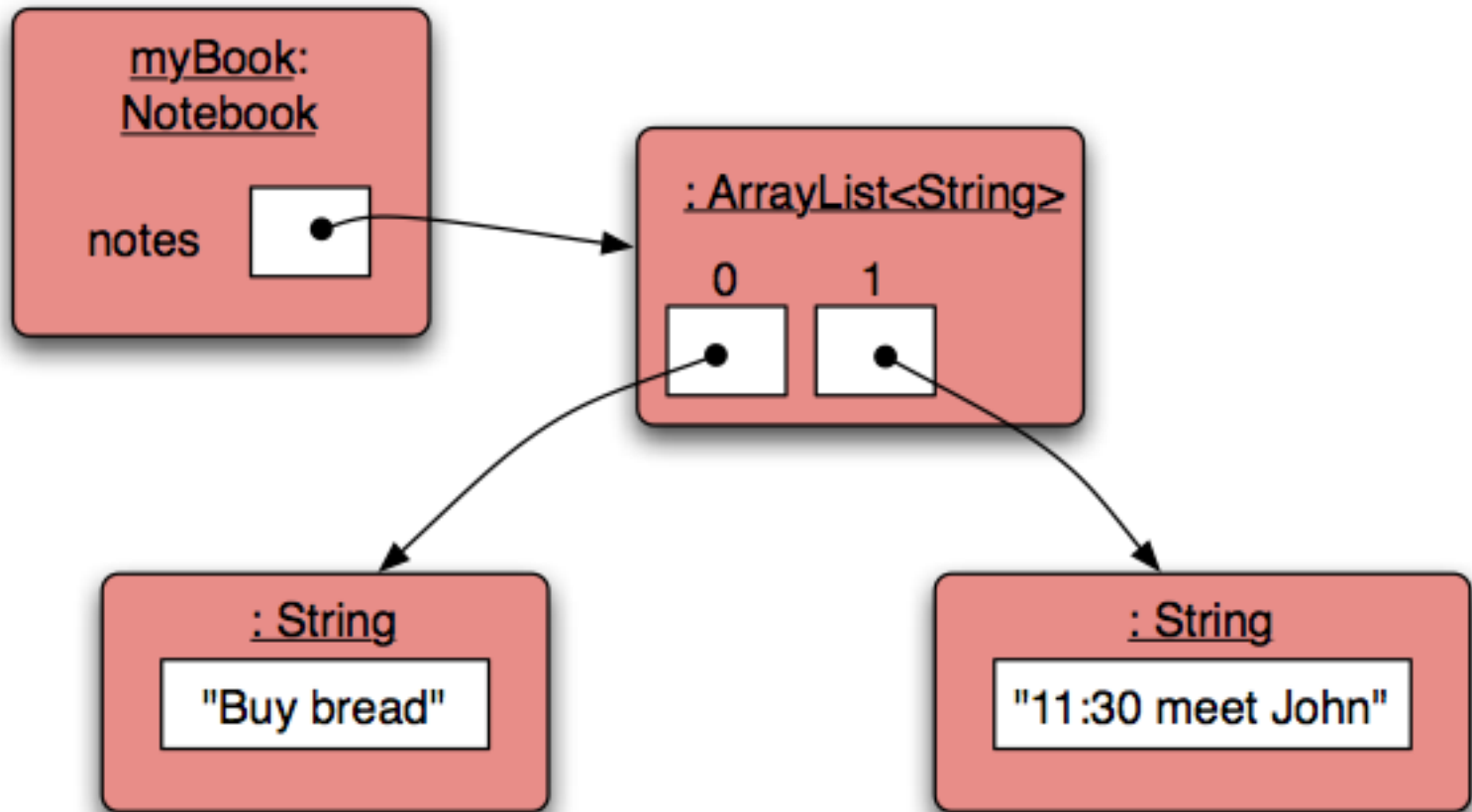
Adicionando uma
nova nota

Retornando o número
de notas (delegação).

Numeração de índice



Remoção pode afetar a numeração



Recuperando um objeto

```
public void showNote(int noteNumber)
{
    if(noteNumber < 0) {
        // Este não é um número de nota válido.
    }
    else if(noteNumber < numberOfNotes()) {
        System.out.println(notes.get(noteNumber));
    }
    else {
        // Este não é um número de nota válido.
    }
}
```

Verificações da validade
do índice

Recupera e imprime a nota

Exercício

- **Bloco de notas pessoal**

- Edite a classe *Notebook* e implemente um método *removeNote* que receba por parâmetro o número de uma nota e a remova da coleção. Dica: use como modelo o método *showNote* e substitua a impressão por uma chamada ao método *remove* de *ArrayList*.
- Modifique *showNote* e *removeNote* para imprimir uma mensagem de erro caso o número informado para a nota não seja válido.

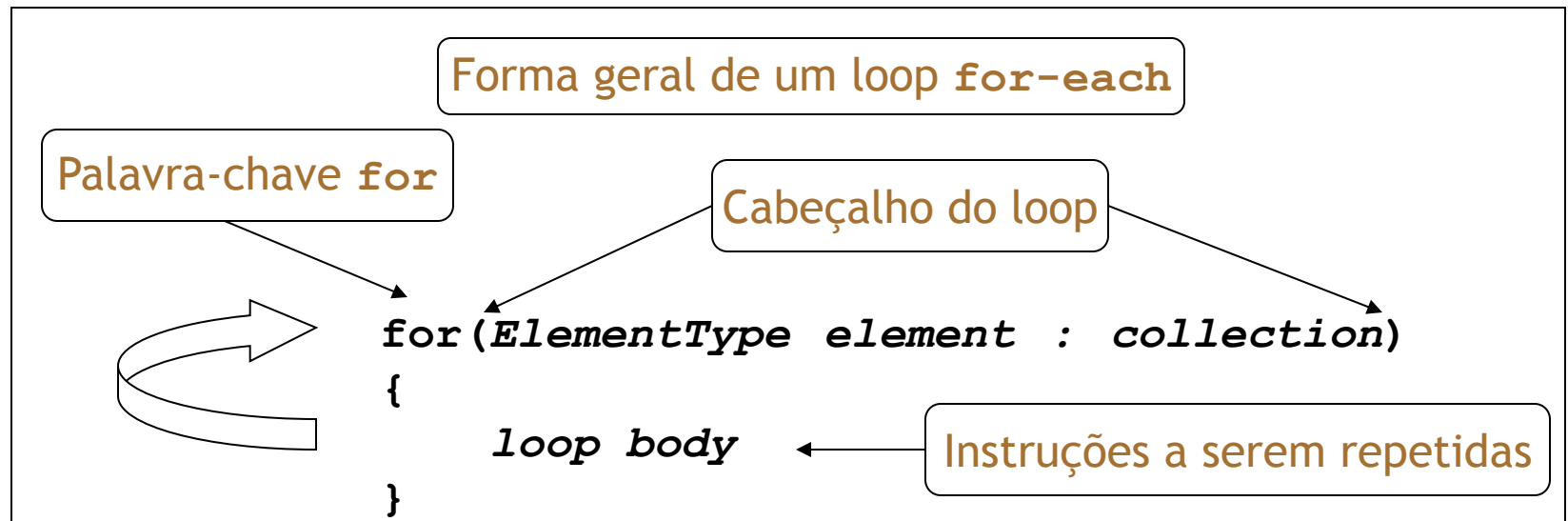
Exercício

- **Bloco de notas pessoal**
 - Como você implementaria um método para imprimir todas as notas, usando o método *get* de *ArrayList* ? Quais seriam os índices válidos para imprimir as notas ?

Iteração

- Frequentemente, queremos realizar algumas ações em um número arbitrário de vezes.
 - Por exemplo, imprimir todas as notas na agenda. Quantas existem?
- A maioria das linguagens de programação possuem instruções de *loop* para tornar isso possível.
- O Java tem diversas instruções de loop.

Pseudocódigo do loop **for-each**




Pseudocódigo para um
loop **for-each**

For each *element* na coleção *collection*,
faça as instruções do corpo do *loop*.

Um exemplo Java

```
/**  
 * Lista todas as notas no bloco de notas.  
 */  
public void listNotes()  
{  
    for(String note : notes) {  
        System.out.println(note);  
    }  
}
```



Não são
usados
índices

Para cada *note* em *notes*, imprima *note*

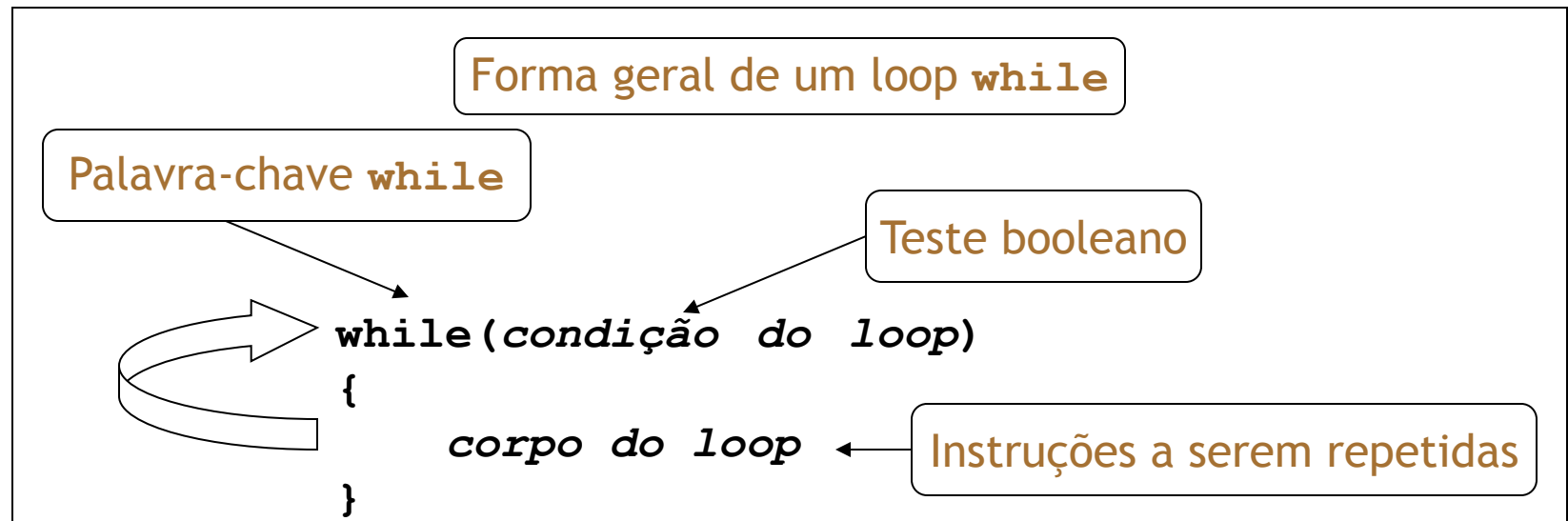
Exercício

- **Bloco de notas pessoal**
 - Edite a classe *Notebook* e implemente um método *listNotes* que imprima todas as notas.

O loop `while`

- O loop `for-each` repete o corpo do loop para cada objeto da coleção.
- Às vezes precisamos de algo diferente disso.
- Podemos usar uma condição booleana para decidir quando seguir ou não.
- O loop `while` provê este controle.

Pseudocódigo do loop `while`



Exemplo de pseudocódigo
para imprimir todas as notas

```
while(há pelo menos mais uma nota a ser impressa) {  
    mostre a próxima nota  
}
```

Um exemplo Java

```
/**
 * Lista todas as notas no bloco de notas.
 */
public void listNotes()
{
    int index = 0;
    while(index < notes.size()) {
        System.out.println(notes.get(index));
        index++;
    }
}
```

Incrementa um na variável



for-each versus while

- for-each:
 - fácil de escrever.
 - seguro: a parada é garantida.
- while:
 - não é preciso processar toda a coleção.
 - não é preciso usar sempre com uma coleção.
 - tenha cuidado: pode ser um loop *infinito*.

Pesquisando em uma coleção

```
int index = 0;
boolean found = false;
while(index < notes.size() && !found) {
    String note = notes.get(index);
    if(note.contains(searchString)) {
        found = true;
    }
    else {
        index++;
    }
}
```

Não é necessário continuar procurando

Exercício

- **Bloco de notas pessoal**

- Edite a classe *Notebook* e implemente um método *searchFirst* que receba um *string* e procure uma nota que o contenha. Se o *string* for encontrado, imprima a nota e interrompa a busca. Dicas: 1) use o método *contains* de *String* e 2) não chame o método *size* de *ArrayList* em cada iteração.
- Implemente um método *listNumberedNotes* que imprima todas as notas prefixadas pelo seu índice no bloco de notas.

Iterator e iterator()

- Coleções possuem um método `iterator()`.
- Ele retorna um objeto `Iterator`.
- `Iterator<E>` tem três métodos:

– `boolean hasNext()`

testa se existe
o próximo

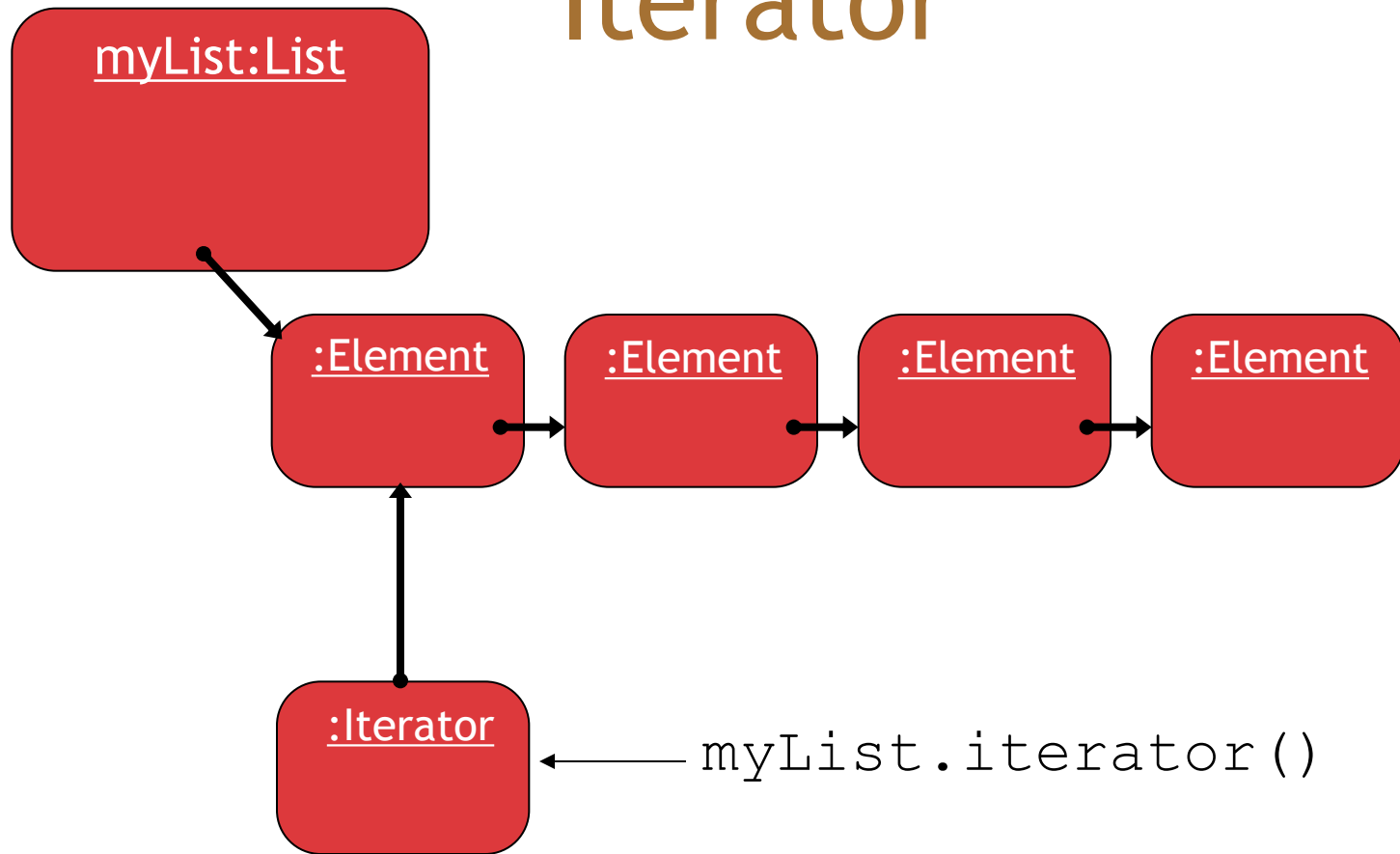
– `E next()`

obtem o próximo

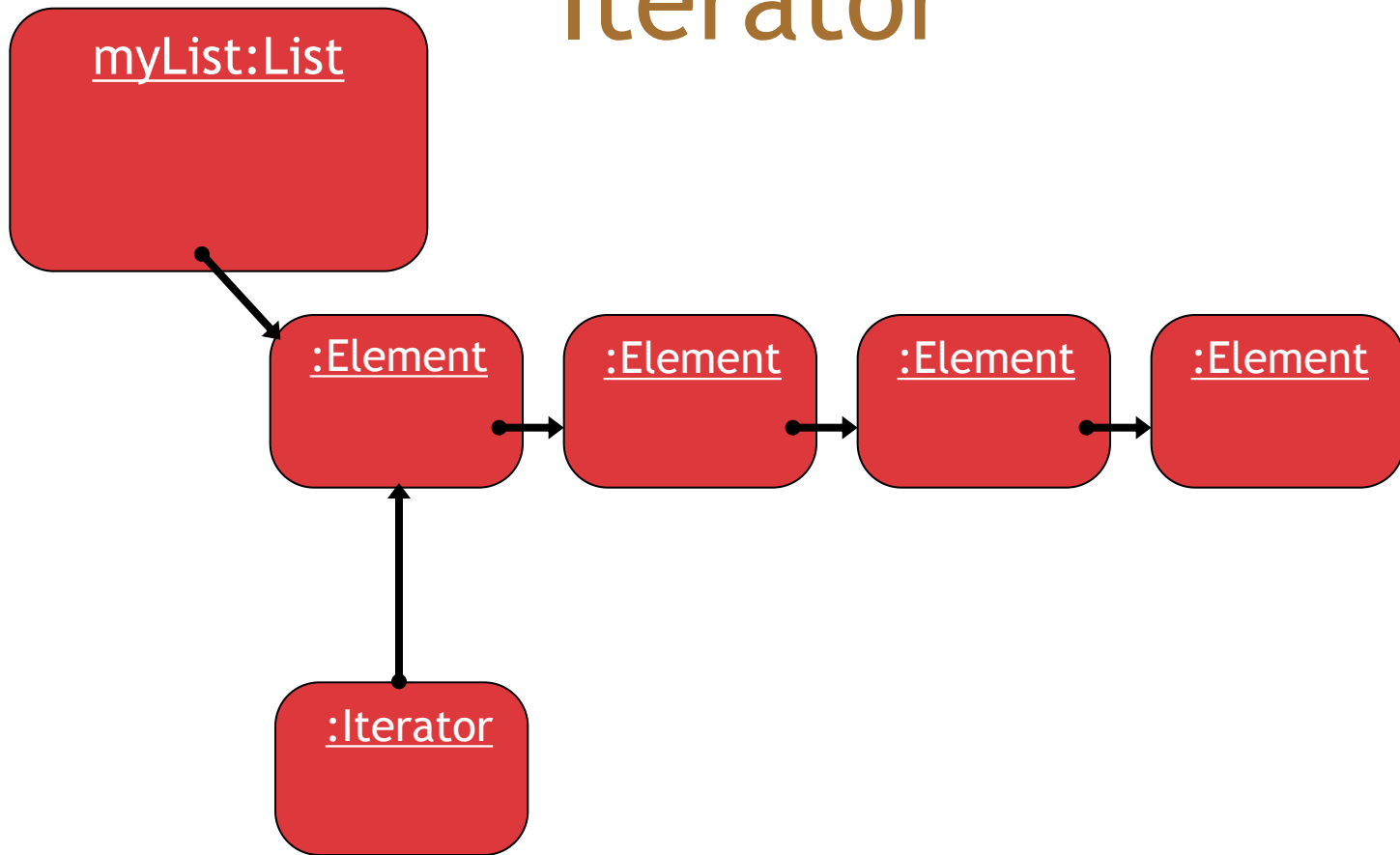
– `void remove()`

remove o último
obtido por `next()`

Iterator

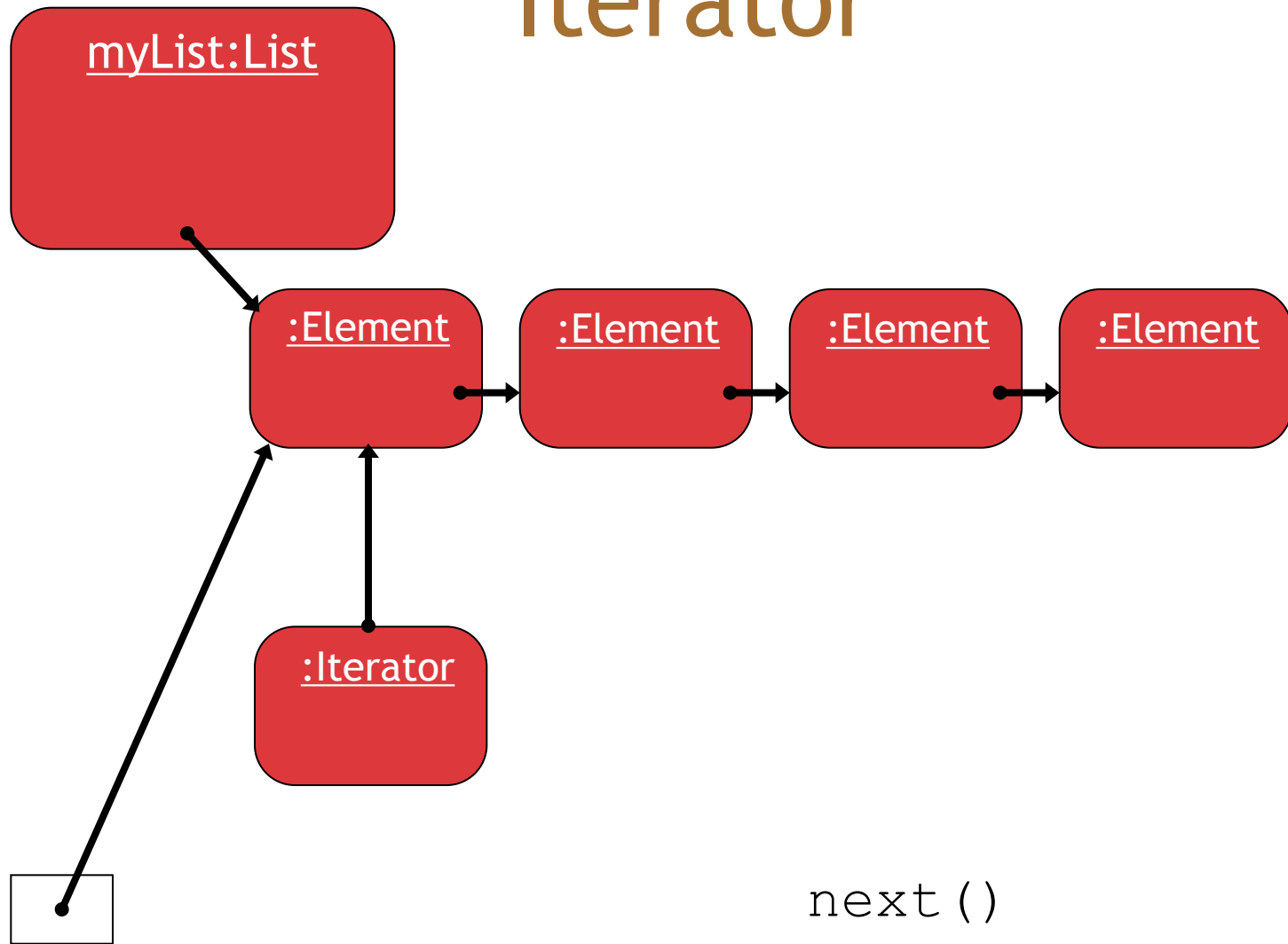


Iterator



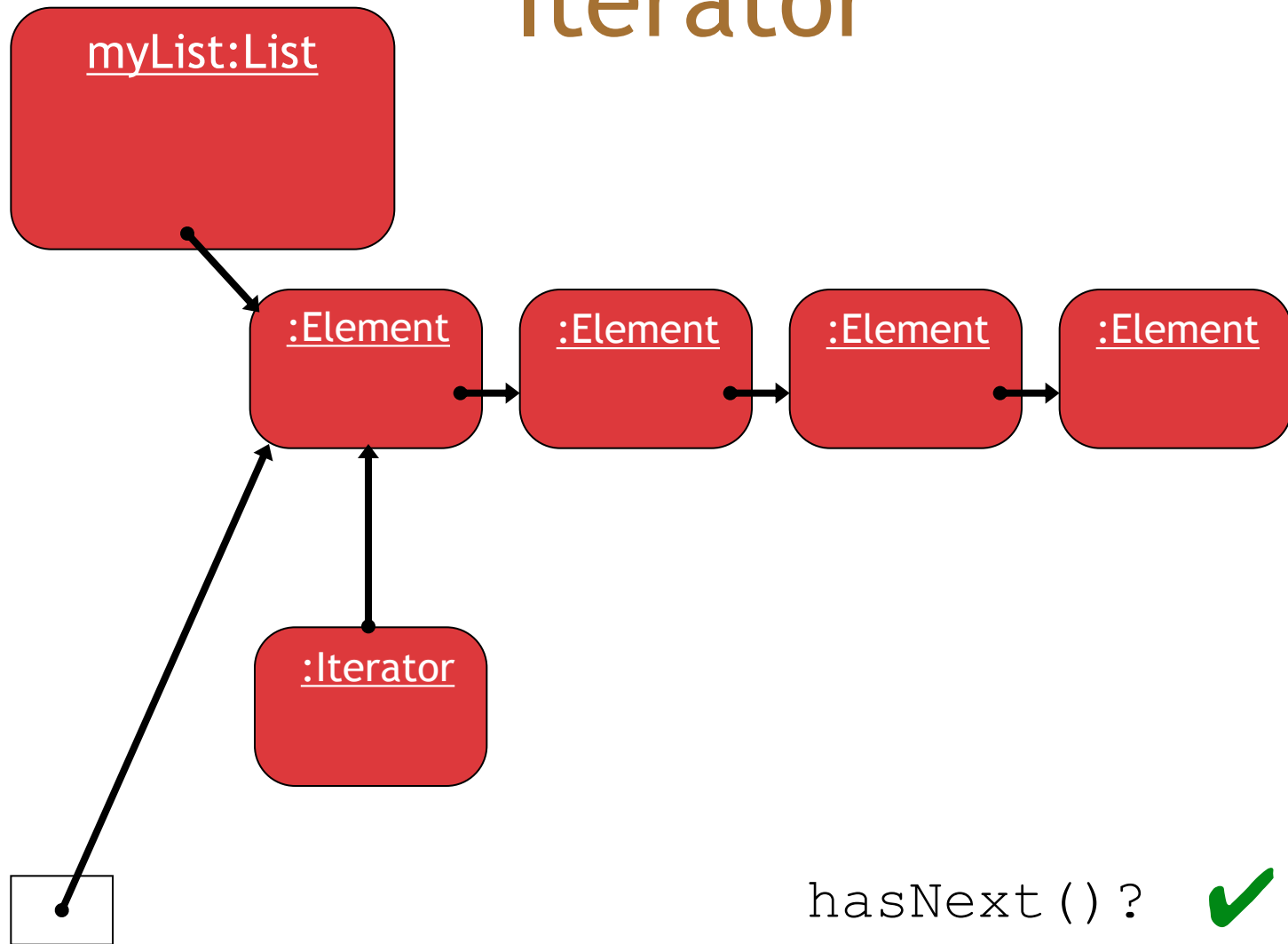
hasNext () ? ✓

Iterator



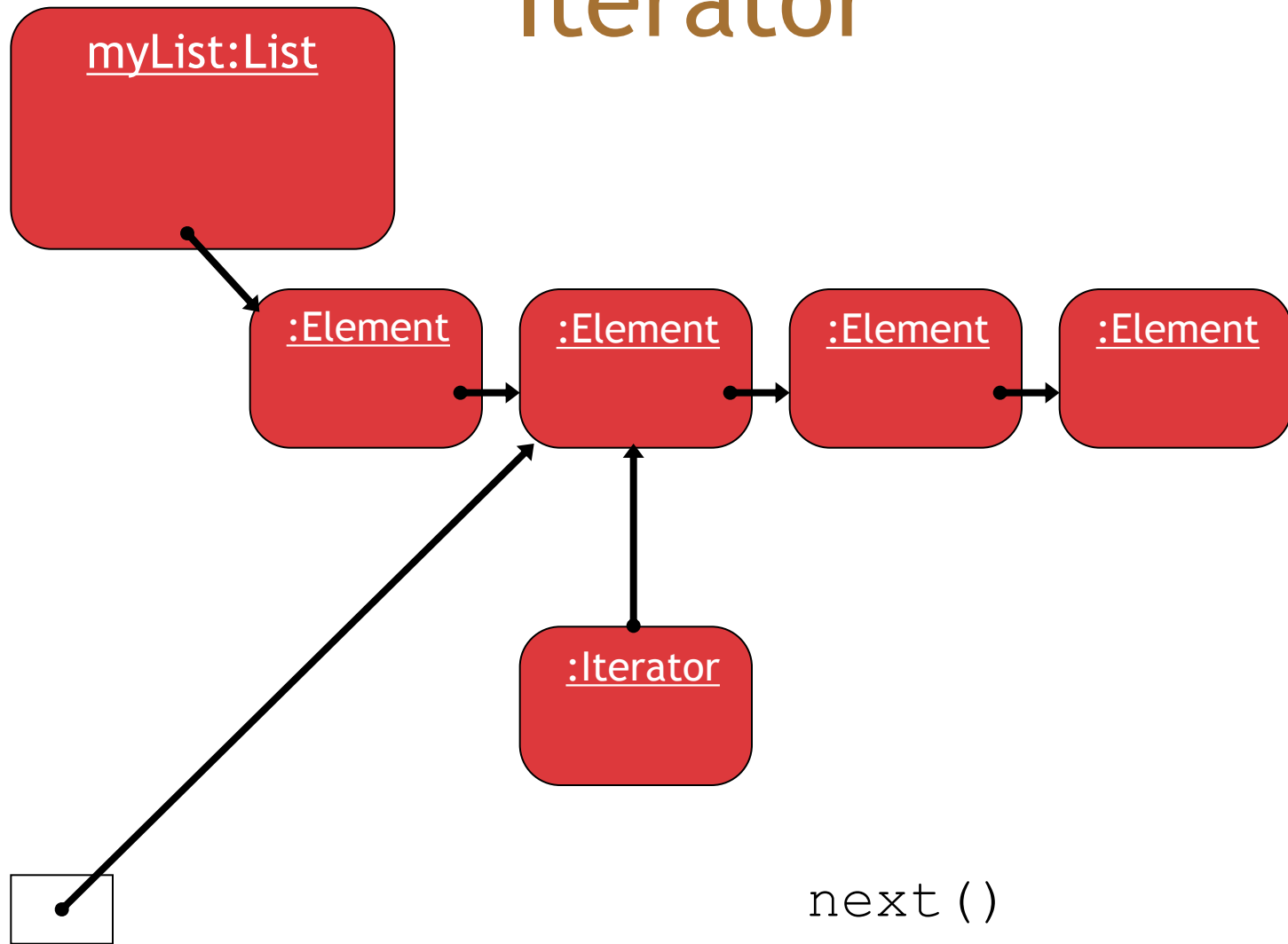
```
Element e = iterator.next();
```

Iterator



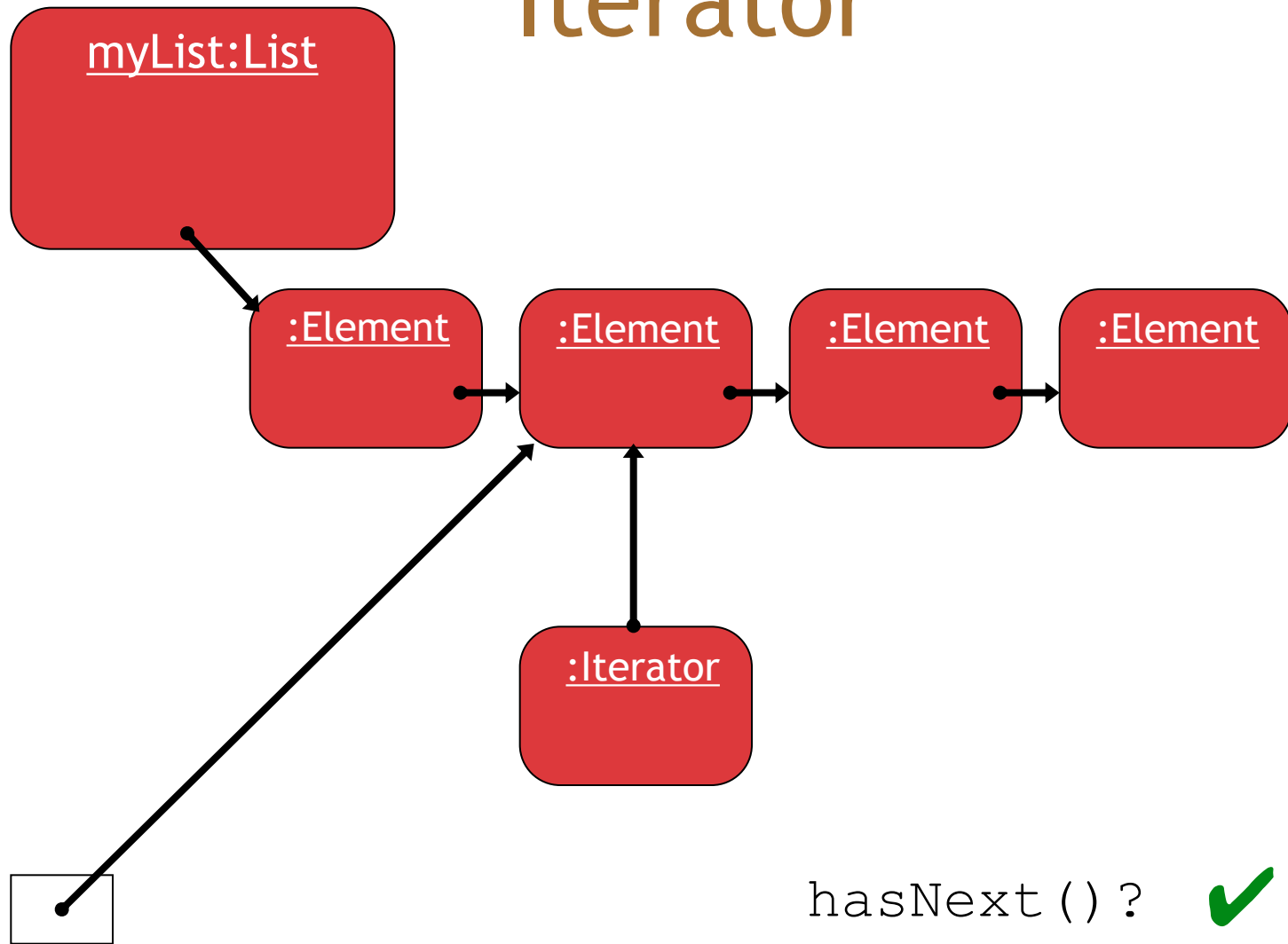
```
Element e = iterator.next();
```

Iterator



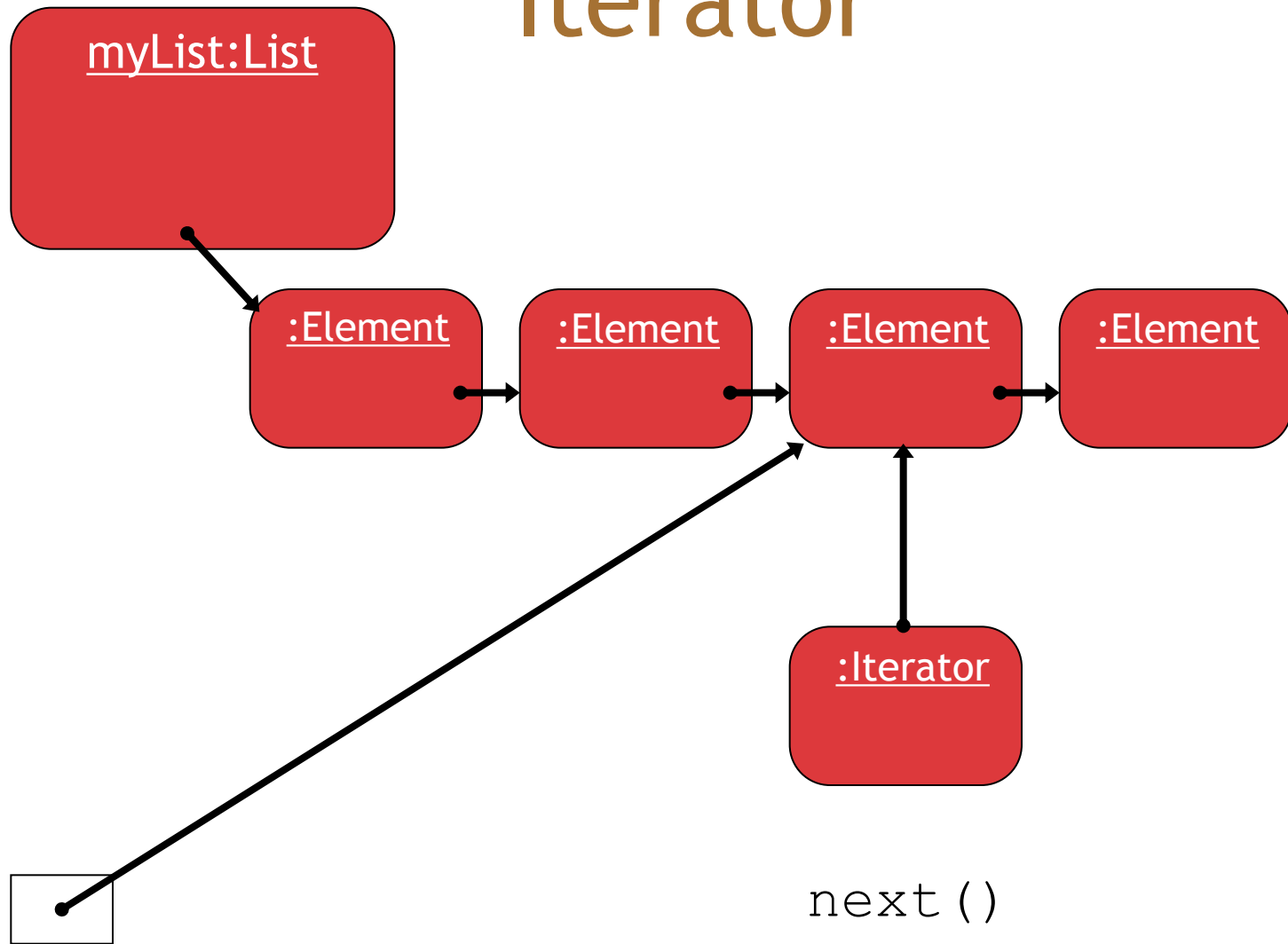
```
Element e = iterator.next();
```

Iterator



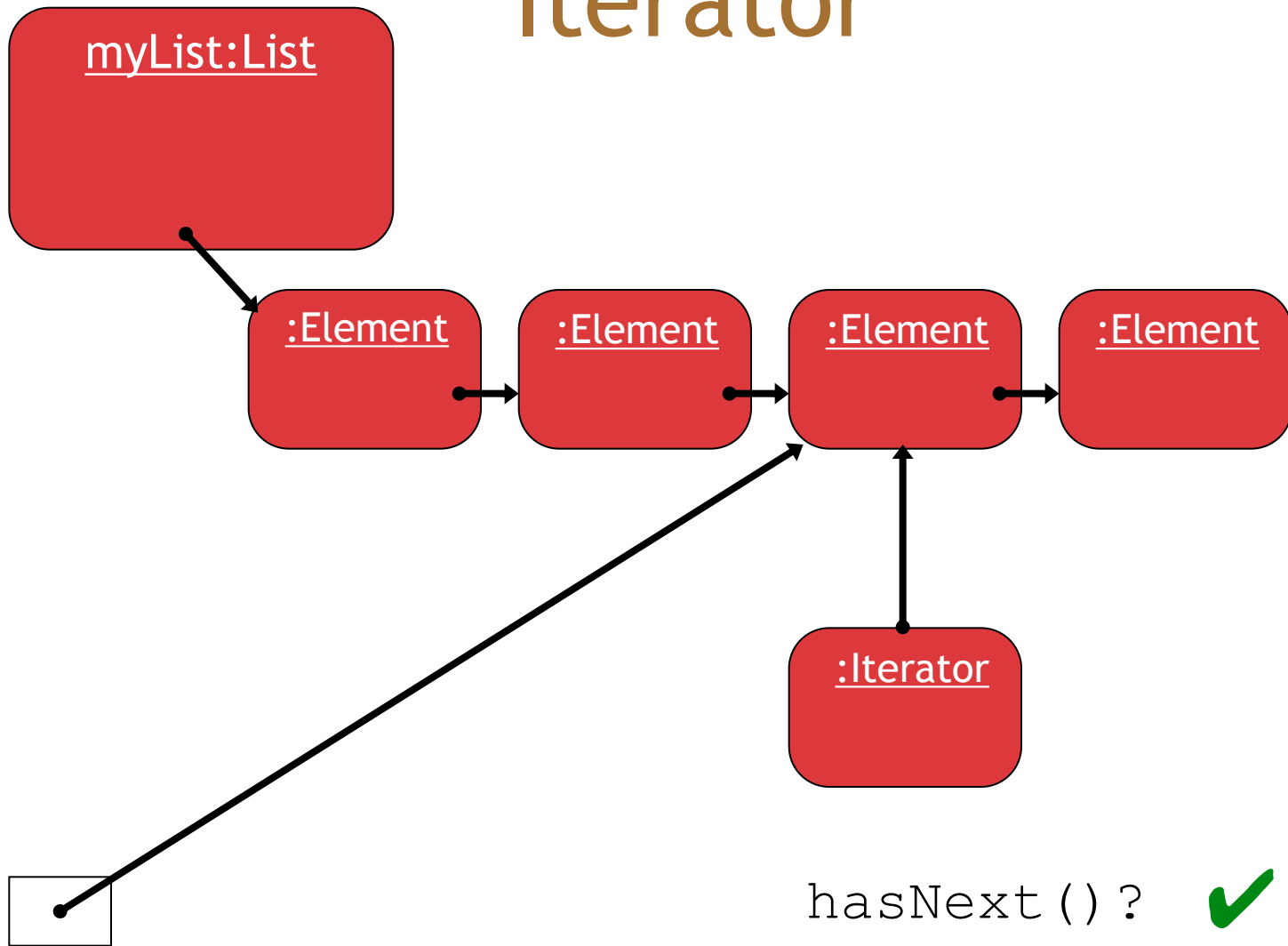
```
Element e = iterator.next();
```

Iterator



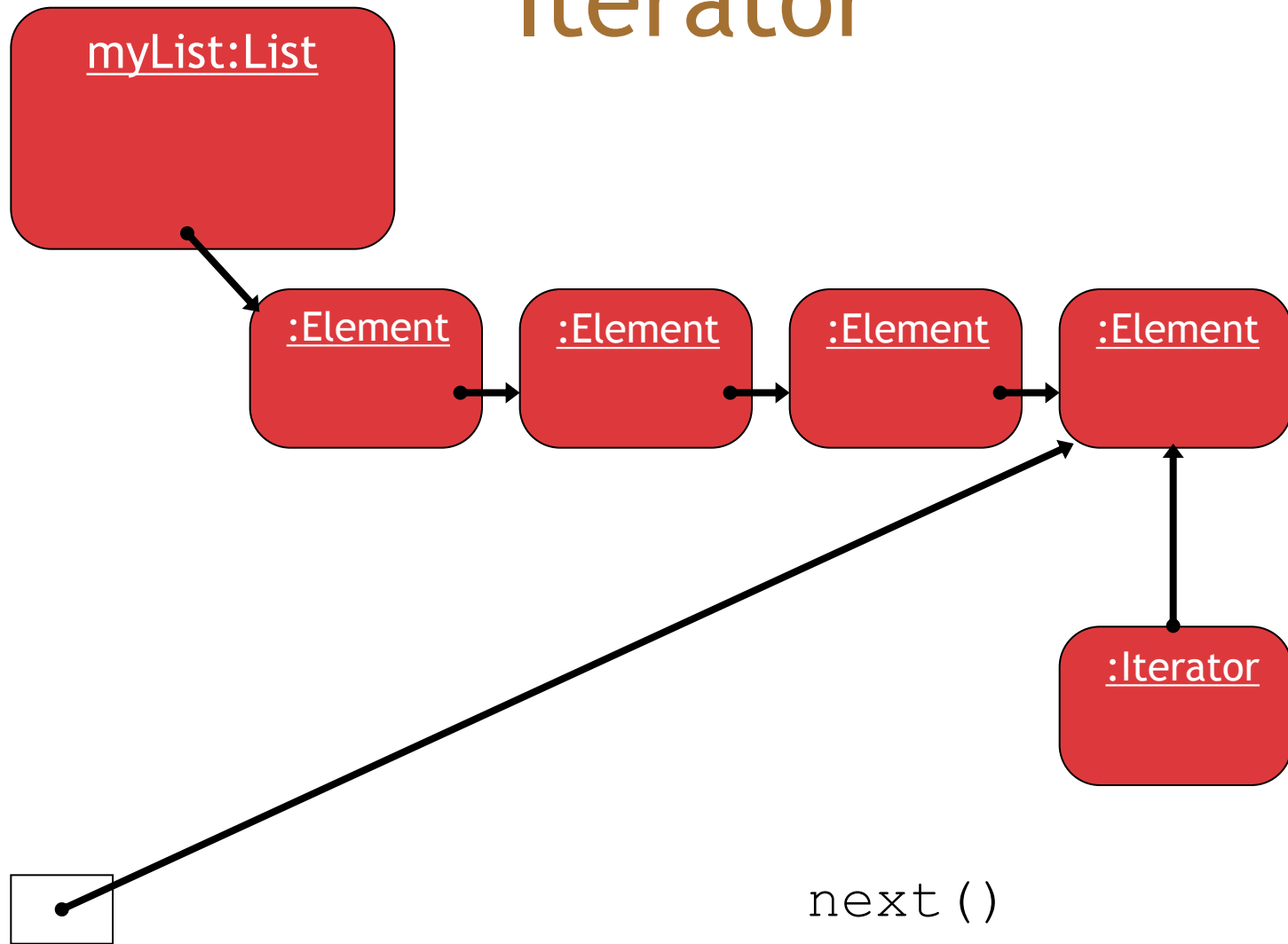
```
Element e = iterator.next();
```

Iterator



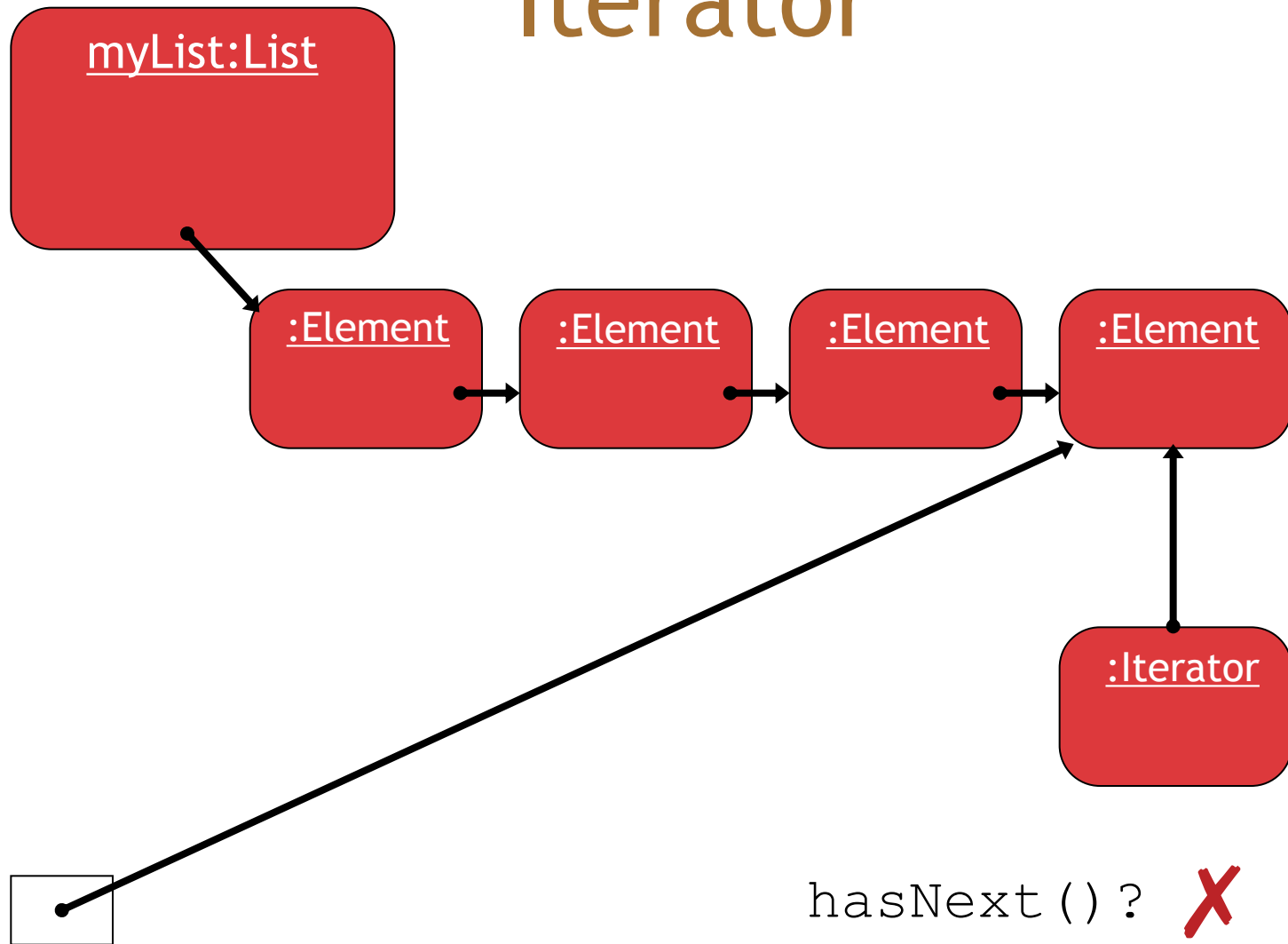
```
Element e = iterator.next();
```

Iterator



```
Element e = iterator.next();
```


Iterator



```
Element e = iterator.next();
```

Usando um objeto Iterator

`java.util.Iterator`

Retorna um objeto Iterator

```
Iterator<ElementType> it = myCollection.iterator();  
while(it.hasNext()) {  
    call it.next() to get the next object  
    do something with that object  
}
```

```
public void listNotes()  
{  
    Iterator<String> it = notes.iterator();  
    while(it.hasNext()) {  
        System.out.println(it.next());  
    }  
}
```

`next()` incrementa o Iterator

Um exemplo Java

Versão for-each

```
for(String note : notes) {  
    System.out.println(note);  
}
```

Versão while

```
int index = 0;  
while(index < notes.size()) {  
    System.out.println(notes.get(index));  
    index++;  
}
```

Versão Iterator

```
Iterator<String> it = notes.iterator();  
while(it.hasNext()) {  
    System.out.println(it.next());  
}
```

Index versus Iterator

- Modos de iterar sobre uma coleção:
 - Loop **for-each**
 - Use se precisar processar todos os elementos.
 - Loop **while**
 - Use se pode querer parar antes do fim.
 - Use para repetições que não usem coleções.
 - Objeto **Iterator**
 - Use se pode querer parar antes do fim.
 - Frequentemente usado com coleções onde o acesso indexado não é muito eficiente ou é impossível.

Removendo elementos

- Não podemos usar o método **remove()** de uma coleção durante uma iteração.
- Devemos usar um objeto **Iterator**, o qual fornece um método **remove()** que pode ser usado em uma iteração.

Removendo elementos

```
import java.util.Iterator;  ← Instrução de importação  
...
```

```
Iterator<String> it = notes.iterator();  
while(it.hasNext()) {  
    String note = it.next();  
    if(note.equals(noteToRemove)) {  
        it.remove();  
    }  
}
```

Use o método remove() de Iterator

Exercício

- **Bloco de notas pessoal**

- Edite a classe *Notebook* e implemente um método *searchAndRemove* que receba um string e remova do bloco todas as notas que contenham este string.
- Altere o método *searchAndRemove* para que imprima a quantidade de notas removidas.
- Compare o estágio atual do projeto em uso com *notebook2*.

Revisão (1)

- Coleções permitem que um número arbitrário de objetos seja armazenado.
- Bibliotecas de classes normalmente contêm classes experimentadas e testadas.
- Bibliotecas de classe Java são chamadas *pacotes*.
- Utilizamos a classe `ArrayList` do pacote `java.util`.

Revisão (2)

- Na classe **ArrayList** :
 - Itens podem ser adicionados e removidos.
 - Todo item tem um índice.
 - Valores de índice podem mudar se os itens forem removidos ou, se outros itens forem adicionados.
- Os principais métodos de **ArrayList** são:
 - **size()**
 - **get()**
 - **add()**
 - **remove()**

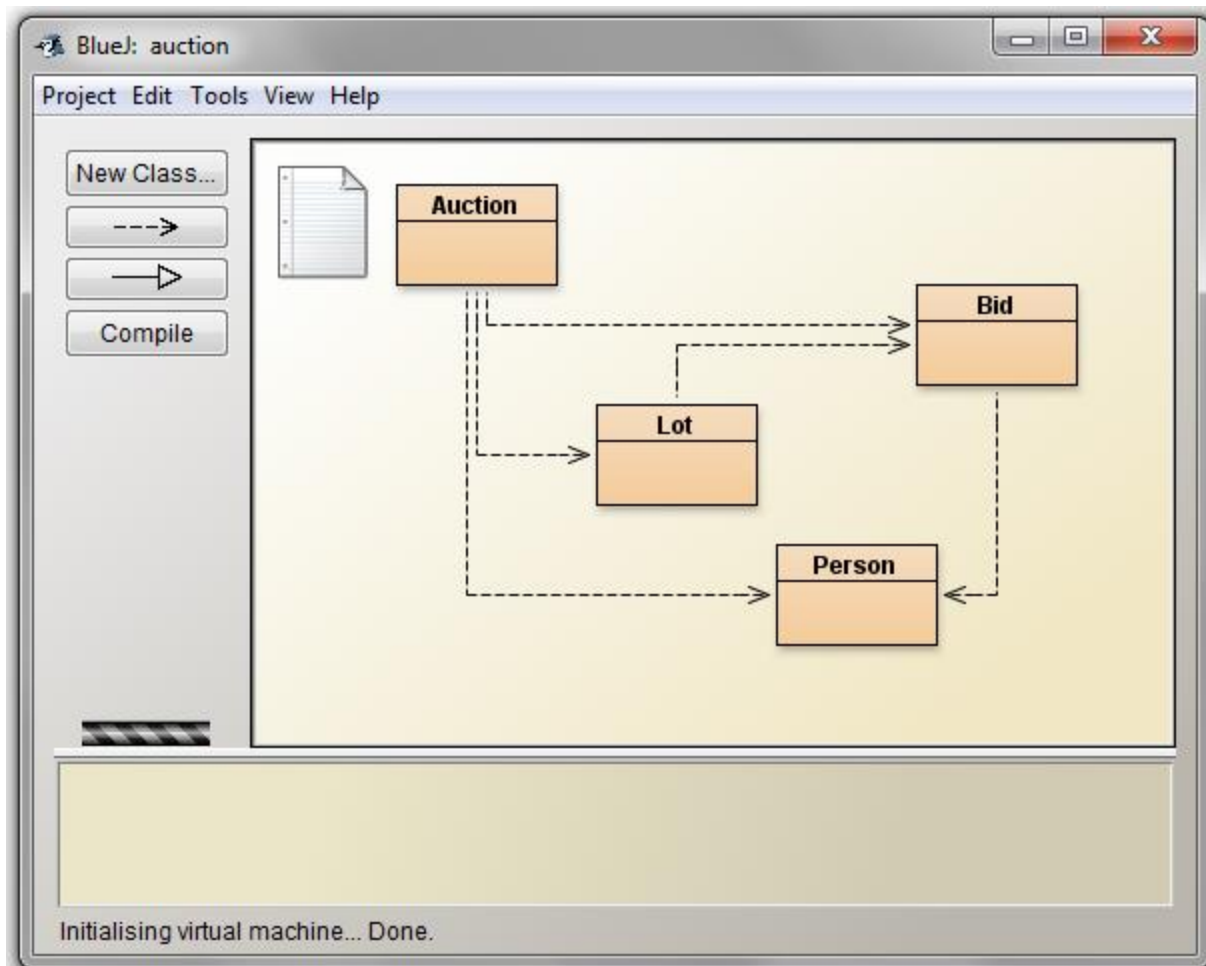
Revisão (3)

- Coleções possuem um método `iterator()` que retorna um um objeto `Iterator`.
- Os métodos de `Iterator` são:
 - `hasNext()`
 - `next()`
 - `remove()`

Um sistema de leilão

- Modelaremos parte da operação de um sistema de leilão online:
 - Leilão é um conjunto de itens oferecidos para a venda.
 - Os itens são chamados lotes e recebem um número identificador único no leilão.
 - Uma pessoa pode oferecer um lance para um lote.
 - No fechamento do leilão, a pessoa que ofereceu o maior lance pode comprar o lote.

Um sistema de leilão



Exercício

- **Sistema de leilão**

- Feche o projeto anterior, abra o projeto *auction* e crie uma instância de *Auction* e duas de *Person*.
- Crie um lote com o método *enterLot* e o visualize com o método *showLots*.
- Faça alguns lances usando o método *bidFor* e visualize o lote. Primeiro faça um lance em nome de uma pessoa. Depois use a outra pessoa para fazer dois lances, um inferior e depois um superior.

Código-fonte: Lot

```
public class Lot
{
    private final int number;
    private String description;
    private Bid highestBid;

    public Lot(int number, String description)
    {
        this.number = number;
        this.description = description;
    }
}
```

Código-fonte: Lot

```
public boolean bidFor(Bid bid)
{
    if((highestBid == null) ||
        (bid.getValue() > highestBid.getValue())) {
        // Este é o melhor lance até agora.
        highestBid = bid;
        return true;
    }
    else {
        return false;
    }
}
```

Testa se highestBid está referenciando algum objeto

Ou curto-circuito

Palavra-chave *null*

- Podemos testar se uma variável de tipo objeto contém o valor null:
`if (highestBid == null) ...`
- No caso sistema de leilão significa ‘sem lance ainda’.

Exercício

- **Sistema de leilão**

- Edite o método *bidFor* da classe *Lot*, substituindo o **ou curto-circuito** (`||`) pelo **ou** (`|`) e compile as classes afetadas.
- Crie uma instância de *Auction* e uma de *Person*. Crie um lote para este leilão.
- Faça um lance para este lote. O que acontece ? Porque ?
- Desfaça a alteração no método *bidFor*.

Código-fonte: Auction

```
public class Auction
{
    private ArrayList<Lot> lots;
    private int nextLotNumber;

    /**
     * Create a new auction.
     */
    public Auction()
    {
        lots = new ArrayList<Lot>();
        nextLotNumber = 1;
    }
}
```

Código-fonte: Auction

```
public void enterLot(String description)
{
    lots.add(new Lot(nextLotNumber, description));
    nextLotNumber++;
}
```

Objeto anônimo



Objeto anônimo

- Frequentemente objetos são criados e usados imediatamente:

```
Lot furtherLot = new Lot(...);  
lots.add(furtherLot);
```

- Nós realmente não precisamos da variável `furtherLot`:

```
lots.add(new Lot(...));
```

Encadeando chamadas de métodos

- Métodos geralmente retornam objetos
- Frequentemente chamamos um método no objeto retornado:

```
Bid bid = lot.getHighestBid();  
Person bidder = bid.getBidder();
```

- Podemos usar o conceito de objeto anônimo e encadear chamadas de métodos:

```
lot.getHighestBid().getBidder()
```

Encadeando chamadas de métodos

- Cada método no encadeamento é chamado no objeto retornado pelo método anteriormente chamado.

```
String name =  
    lot.getHighestBid().getBidder().getName();
```

Retorna um objeto Bid a partir do Lot

Retorna um objeto Person a partir do Bid

Retorna um objeto String a partir do Person

Revisão (4)

- Podemos criar objetos anônimos quando não é necessário guardar uma referência para uso posterior.
- Podemos encadear chamadas de métodos criando diversos objetos anônimos.

Coleções de tamanho fixo

- Às vezes, o tamanho máximo de uma coleção pode ser predeterminado.
- Linguagens de programação freqüentemente oferecem um tipo de coleção de tamanho fixo: um *array*.
- Arrays Java podem armazenar valores de objetos ou de tipo primitivo.
- Arrays utilizam uma sintaxe especial.

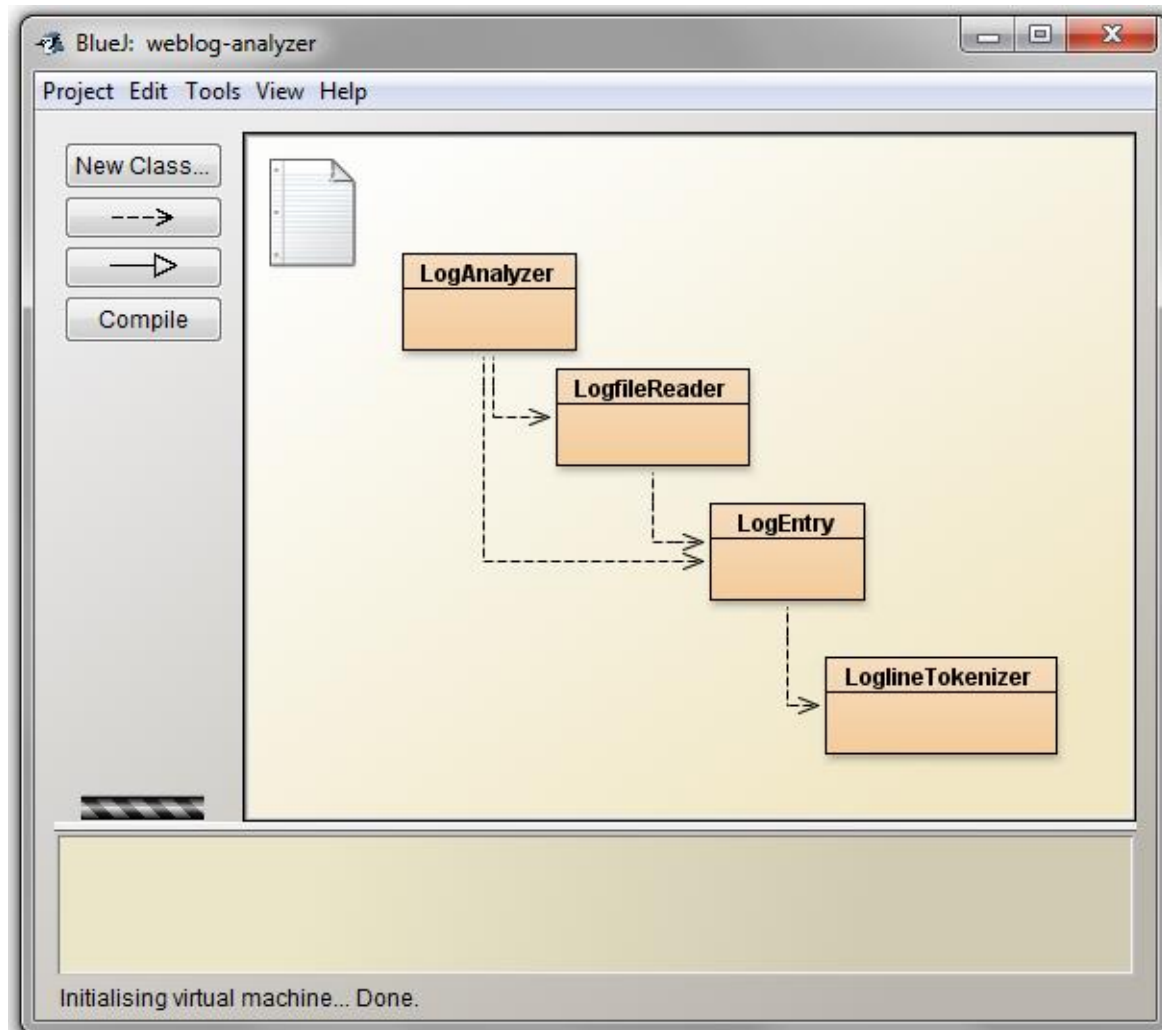
Um analisador de arquivos de log

- Um servidor Web registra os detalhes de cada acesso em arquivos de log:
 - Suporte a tarefas do webmaster.
 - Páginas mais populares.
 - Períodos mais ocupados.
 - Quantos dados são entregues.
 - Referências quebradas.
 - Análise dos acessos por hora.

Um analisador de arquivos de log

- Modelaremos uma aplicação que realiza análise dos acessos a servidor web a partir do seu arquivo de log:
 - O servidor web escreve uma linha no arquivo de log para cada acesso feito.
 - Cada linha registra marca de tempo do acesso no formato:
 - ano mês dia hora minuto
 - 2012 04 20 10 30

Um analisador de arquivos de log



Exercício

- **Analizador de arquivo de log**
 - Feche o projeto anterior, abra o projeto *weblog-analyser1* e crie uma instância de *LogAnalyser*.
 - Chame o método *printData* e analise os registros.
 - Chame o método *analyzeHourlyData* e depois o método *printHourlyCounts*. Quais são as horas mais ocupadas do dia ?

Código-fonte: LogAnalyser

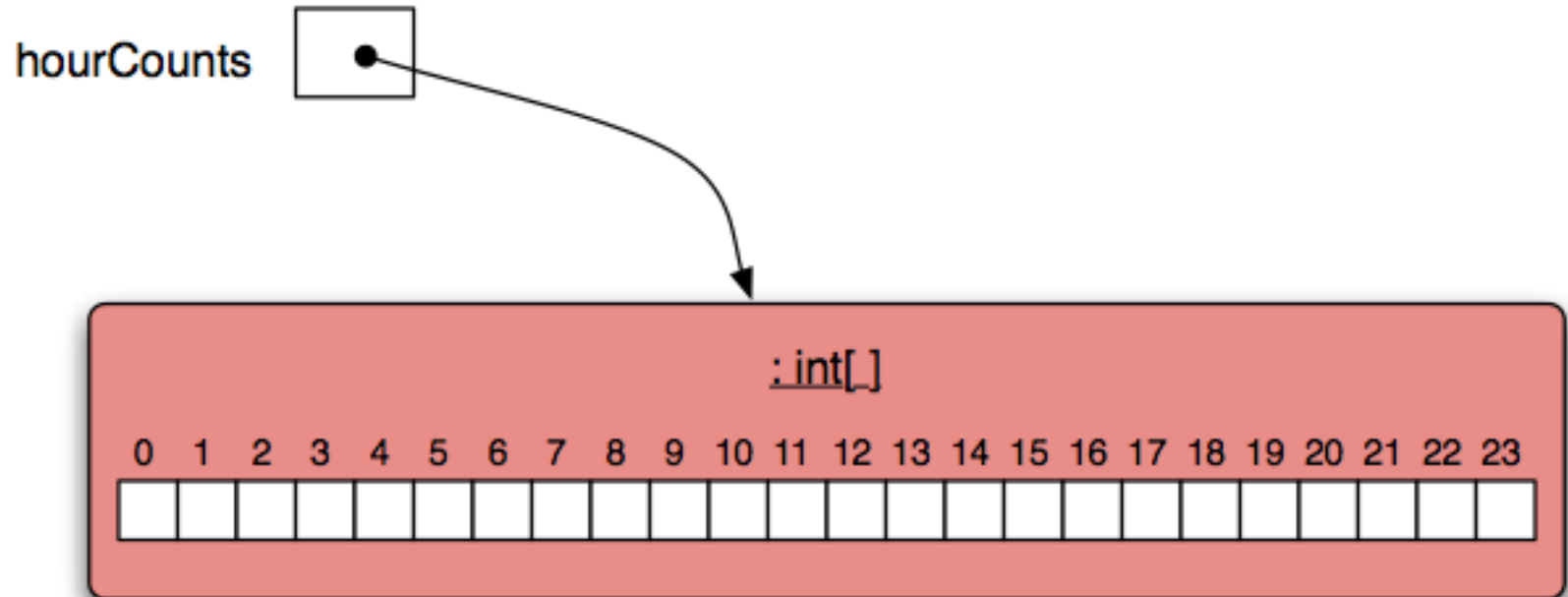
```
public class LogAnalyzer
{
    private int[] hourCounts;
    private LogfileReader reader;

    public LogAnalyzer()
    {
        hourCounts = new int[24];
        reader = new LogfileReader();
    }
    ...
}
```

← Declaração da variável
para referenciar o array

← Criação do objeto array

0 array hourCounts



Utilizando um array

- A notação entre colchetes é utilizada para acessar um elemento do array:

`hourCounts[...]`

- Elementos são utilizados como variáveis convencionais.

- À esquerda de uma atribuição:

- `hourCounts[hour] = ...;`

- Em uma expressão:

- `adjusted = hourCounts[hour] - 3;`

- `hourCounts[hour]++;`

Uso padrão de array

```
private int[] hourCounts;
```

← declaração

```
...
```

```
hourCounts = new int[24];
```

← criação

```
...
```

```
hourcounts[i] = 0;
```

```
hourcounts[i]++;
```

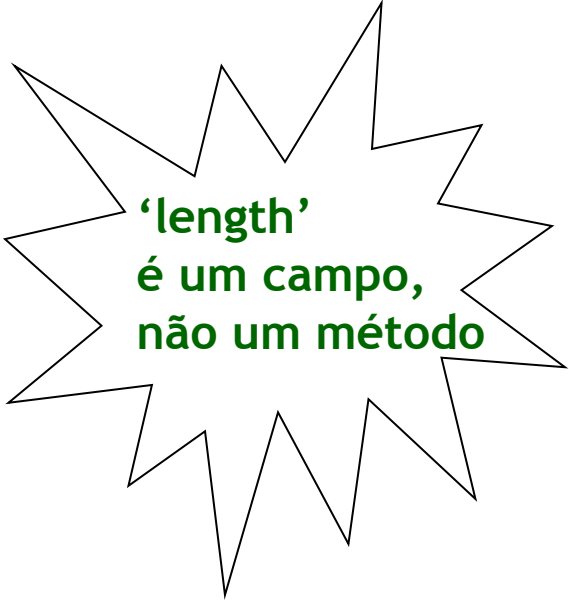
```
System.out.println(hourcounts[i]);
```

← uso

Array length

```
private int[] hourCounts;  
...  
hourCounts = new int[24];  
...  
int n = hourCounts.length;
```

Sem parêntesis !!!



'length'
é um campo,
não um método

Array de literais

```
private int[] numbers = {3,15};
```

Declaração e
inicialização
(tamanho
definido
pelos dados)

```
System.out.println(numbers[i]);
```

uso

Código-fonte: LogAnalyser

```
public void analyzeHourlyData()  
{  
    while(reader.hasMoreEntries()) {  
        LogEntry entry = reader.nextEntry();  
        int hour = entry.getHour();  
        hourCounts[hour]++;  
    }  
}
```

Uso do objeto array

O loop `for`

- Há duas variações do loop `for`: *for-each* e *for*.
- O loop `for-each` é usado somente com coleções.
- O loop `for` é frequentemente usado para iterar um número fixo de vezes.
- O loop `for` é frequentemente usado com uma variável que sofre uma variação fixa em cada iteração.

Pseudocódigo do loop `for`

Forma geral de um loop `for`

```
for(inicialização; condição; ação após o corpo) {  
    instruções a serem repetidas  
}
```

Forma equivalente com loop `while`

```
inicialização;  
while(condição) {  
    instruções a serem repetidas  
    ação após o corpo  
}
```

Um exemplo Java

Versão com loop `for`

```
int index;  
for (index = 0; index < notes.size(); index++) {  
    System.out.println(notes.get(index));  
}
```

Versão com loop `while`

```
int index = 0;  
while(index < notes.size()) {  
    System.out.println(notes.get(index));  
    index++;  
}
```

Código-fonte: LogAnalyser

```
public void printHourlyCounts()  
{  
    System.out.println("Hr: Count");  
    for(int hour = 0; hour < hourCounts.length; hour++) {  
        System.out.println(hour + ": " + hourCounts[hour]);  
    }  
}
```

Forma equivalente com loop **for-each**

```
for(int value : hourCounts) {  
    System.out.println(": " + value);  
}
```

↑
Sem apresentar a hora
(não há contador do loop)

Loops sem usar coleções

Versão com loop `for`

```
// Imprimir os números de 0 a 30.  
int index;  
for (index = 0; index <= 30; index++) {  
    System.out.println(index);  
}
```

Versão com loop `while`

```
// Imprimir os números de 0 a 30.  
int index = 0;  
while (index <= 30) {  
    System.out.println(index);  
    index++;  
}
```


Removendo elementos

```
Iterator<String> it = notes.iterator();  
while(it.hasNext()) {  
    String note = it.next();  
    if(note.equals(noteToRemove)) {  
        it.remove();  
    }  
}
```

```
for (Iterator<String> it = notes.iterator(); it.hasNext(); ) {  
    String note = it.next();  
    if(note.equals(noteToRemove)) {  
        it.remove();  
    }  
}
```

O loop *do-while*

- O Java tem outra estrutura de loop chamada loop *do-while*.
- O loop *do-while* é muito adequado quando é necessário executar ao menos uma vez o corpo do loop antes de testar a condição.
- Pode ser sempre substituído por um loop *while*.

Pseudocódigo do loop do-while

inicialização;

do {

 instruções a serem repetidas

 ação após o corpo

} while (condição);

Forma geral de um loop do-while

inicialização;

boolean ok = false;

while(!ok) {

 instruções a serem repetidas

 ação após o corpo

 if(condição) ok = true;

}

Forma equivalente com loop while

Exercício

- **Analizador de arquivo de log**
 - Edite a classe *LogAnalyser* e implemente um método *busiestHour* que retorna a hora com mais acessos (no caso de empate, escolha a primeira) e a respectiva quantidade de acessos.
 - Altere o método *printHourlyCounts* de modo que as horas sejam apresentadas com dois dígitos.
 - Compare o estágio atual do projeto em uso com *weblog-analyser2*.

Revisão (5)

- Arrays são apropriados quando é preciso uma coleção de tamanho fixo.
- Arrays usam uma notação especial.
- Loops *for* são usados quando a quantidade de repetições é conhecida.
- Loops *for* são usados quando uma variável index é necessária.



Contatos

Câmara dos Deputados
CENIN - Centro de Informática

Carlos Renato S. Ramos

carlosrenato.ramos@camara.gov.br