Java para Concursos – Módulo 3

Vitor Almeida
http://www.itnerante.com.br/profile/vitor

Agenda

- 1 Números e Datas
- 2 Interfaces e Classes Abstratas
- 3 Enums
- 4 Coleções
- 5 Generics
- 6 Classes Internas

1 Números e Datas

1.1 Introdução

- Basicamente fazemos três coisas com números e datas
  - Parsing (conversão de uma String em número ou data)
  - Formatação
  - Manipulação
- Veremos três classes
  - java.lang.Math
  - java.util.Date e java.util.Calendar

1.2 Conversão de Números

- Wrappers possuem métodos estáticos para conversão de Strings para números

```
public static int parseInt(String s) throws
NumberFormatException
```

- Exemplo

```
1 int x = Integer.parseInt("123");
```

1.2 Conversão de Números

- Exemplo

```
01 import java.util.Scanner;
02 public class NumTeste {
03 public static void main(String[] args) {
04  Scanner scanner = new Scanner(System.in);
05  String userInput = scanner.next();
06  try {
07   int i = Integer.parseInt(userInput);
08   System.out.println("Digitou: " + i);
09  }
10  catch (NumberFormatException e) {
11   System.out.println("Input inválido");}}}
```

1.2 Formatação de Números

- Para formatar números, Java oferece a classe java.text.NumberFormat
- É uma classe abstrata. Na prática, você utiliza a subclasse java.text.DecimalFormat

1.2 Formatação de Números

- Exemplo

```
1 import java.text.NumberFormat;
2 import java.util.Locale;
3 public class NFTeste {
4   public static void main(String[] args) {
5     NumberFormat nf =
      NumberFormat.getInstance(Locale.US);
6     System.out.println(nf.format(123445));
7   }
8 } //123,445
```

1.2 Formatação de Números

- Você pode usar o método parse de NumberFormat para conversão de Strings

```
public java.lang.Number parse(java.lang.String
s) throws ParseException
```

1.3 java.lang.Math

- Math fornece métodos estáticos para realização de operações matemáticas
- Tem duas constantes: E e PI
- Métodos

```
public static double abs(double a)
public static double acos(double a)
public static double asin(double a)
```

1.3 java.lang.Math

- Métodos

```
public static double atan(double a)
public static double cos(double a)
public static double exp(double a)
public static double log(double a)
public static double log10(double a)
public static double max(double a, double b)
public static double min(double a, double b)
```

1.4 Trabalhando com o Tempo

- São duas classes
  - java.util.Date
  - java.util.Calendar
- Dois construtores para Date

```
public Date()
public Date(long t)
```

- Dois métodos úteis

```
public boolean after(Date d)
public boolean before(Date d)
```

1.4 Trabalhando com o Tempo

- Exemplo

```
1 Date d1 = new Date(1000);
2 Date d2 = new Date(1001);
3 if (d1.before(d2)) {
4   System.out.println("d1 anterior a d2");
5 } else {
6   System.out.println("d1 depois de d2");
7 }
```

1.4 Trabalhando com o Tempo

- Para criar uma instância de Calendar, use os métodos

```
public static Calendar getInstance(Locale l)
public void setTime(Date d)
```

```
1 Calendar c = Calendar.getInstance();
2 c.setTime(minhaData);
```

- O método getTime retorna a data armazenada em uma instância de Calendar

```
public final Date getTime();
```

1.4 Trabalhando com o Tempo

- Para pegar uma parte da data armazenada num objeto Calendar utilize o método get

```
public int get(int campo)
```

- Os campos possíveis
  - Calendar.YEAR, Calendar.MONTH, Calendar.DATE, Calendar.HOUR, Calendar.MINUTE, Calendar.SECOND, Calendar.MILLISECOND

1.4 Trabalhando com o Tempo

- Para formatar datas podemos usar a classe java.text.DateFormat
- Ela aceita 4 estilos
  - DateFormat.SHORT ou 12/2/2011
  - DateFormat.MEDIUM ou Dec 2, 2011
  - DateFormat.LONG ou December 2, 2011
  - DateFormat.FULL ou Friday, December 2, 2011

1.4 Trabalhando com o Tempo

- O método getInstance é utilizado para pegar uma instância de DateFormat, o format é utilizado para formatar e o parse para converter uma String em data

```
public static DateFormat getDateInstance(int
estilo)

public final String format(Date d)

public Date parse(String s) throws
ParseException
```

1.4 Trabalhando com o Tempo

```
01 import java.text.DateFormat;
02 import java.text.ParseException;
03 import java.util.Date;
04 public class DateFormatTest {
05  public static void main(String[] args) {
06   DateFormat l =
DateFormat.getDateInstance(DateFormat.LONG);
07   DateFormat f =
DateFormat.getDateInstance(DateFormat.FULL);
08   System.out.println(l.format(new Date()));
09   System.out.println(f.format(new Date()));
10   try {
11    Date d = l.parse("December 2, 2006");
12   } catch (ParseException e) {}}}
```

1.4 Trabalhando com o Tempo

- Outra coisa para se preocupar é quanto à tolerância de DateFormat. Por padrão, ela permite a conversão de Strings com datas irreais (exemplo: 31/02/2013)
- Dois métodos tratam do problema

```
public boolean isLenient()
public void setLenient(boolean value)
```

1.4 Trabalhando com o Tempo

- A classe SimpleDateFormat permite usar padrões

```
1 import java.text.SimpleDateFormat;
2 import java.util.Date;
3 public class SimpleDateFormatTeste {
4  public static void main(String[] args) {
5    String pad = "dd/MM/yyyy";
6    SimpleDateFormat f = new
     SimpleDateFormat(pad);
7    System.out.println(f.format(new Date()));
8  }}
```

1.5 Questões de Concurso

- 1 Cesgranrio CMB 2012 Qual método da classe Match retorna o menor número inteiro maior ou igual ao valor dado?

(A) valueOf()

(B) random()

(C) floor()

(D) ceil()

(E) min()

1.5 Questões de Concurso

- 1 Cesgranrio CMB 2012 Qual método da classe Match retorna o menor número inteiro maior ou igual ao valor dado?

(A) valueOf()

(B) random()

(C) floor()

(D) ceil()

(E) min()

1.5 Questões de Concurso

- 2 FCC TRE-CE 2012 Considere o fragmento de código Java a seguir:

```
double a, b, c, d, e, f, r1, r2;
boolean r3;
a = 2; b = 3; c = 5; d = 3; e = 10; f = 1;
r1 = Math.pow(b, a) – e / (a+b) * d + a * c + d;
r2 = Math.sqrt(r1) * b + e – c * a;
r3 = r1 + b != r2 + d;
```

Os valores que serão armazenados nas variáveis r1, r2 e r3 são, respectivamente,

a) 15.0, 11.61 e true.

b) 16.0, 12.0 e true.

c) 16.0, 12.0 e false.

d) 16.0, 34 e true.

e) 15.0, 12.0 e false.

1.5 Questões de Concurso

- 2 FCC TRE-CE 2012 Considere o fragmento de código Java a seguir:

```
double a, b, c, d, e, f, r1, r2;
boolean r3;
a = 2; b = 3; c = 5; d = 3; e = 10; f = 1;
r1 = Math.pow(b, a) – e / (a+b) * d + a * c + d;
r2 = Math.sqrt(r1) * b + e – c * a;
r3 = r1 + b != r2 + d;
```

Os valores que serão armazenados nas variáveis r1, r2 e r3 são, respectivamente,

a) 15.0, 11.61 e true.

b) 16.0, 12.0 e true.

c) 16.0, 12.0 e false.

d) 16.0, 34 e true.

e) 15.0, 12.0 e false.

1.5 Questões de Concurso

- Gabarito
  - 1 D
  - 2 B

2 Interfaces e Classes Abstratas

2.1 Interfaces

- Uma interface não é uma classe com métodos sem implementação
- Uma interface é um contrato entre um provedor de serviços e um cliente
- Exemplo

```
1 public interface DriveImp {
2  public void imprimir(Documento doc);
3 }
```

2.1 Interfaces

- Uma implementação de DriveImp

```
1 public class HPDriver implements DriveImp {
2   public void imprimir(Documento doc) {
3     //código para fazer HP imprimir
4   }
5 }
```

- Atributos são implicitamente públicos, estáticos e final

```
1 public int STATUS = 1;
2 int STATUS = 1;
3 public static final STATUS = 1;
```

2.1 Interfaces

- Ter que implementar todos os métodos de uma interface é tedioso
- A saída é criar uma classe base

```
1 import java.io.IOException;
2 import javax.servlet.GenericServlet;
3 import javax.servlet.ServletException;
4 import javax.servlet.ServletRequest;
5 import javax.servlet.ServletResponse;
6 public class MeuServ extends GenericServlet{
7  public void service(ServletRequest request,
ServletResponse response)
throws ServletException, IOException {
8   response.getWriter().print("Olá");}}}
```

2.2 Classes Abstratas

- Uma classe possui métodos abstratos, que precisam ser implementados pelas subclasses, mas também podem possuir métodos com implementação
- Os métodos sem implementação devem ser declarados como abstratos

```
1 public abstract class ImpPadrao {
2  public String toString() {
3   return "Uma impressora para imprimir";
4  }
5  public abstract void print(Object doc);}
```

2.2 Classes Abstratas

- Um exemplo de subclasse

```
1 public class HPImpr extends ImpPadrao {
2  public void imprimir(Object doc) {
3    System.out.println("Imprimindo");
4    //mais código da HP
5   }
6 }
```

2.3 Questões de Concurso

- 1 CESPE TRT-ES 2009 A interface é uma coleção de operações que pode especificar serviços de uma classe ou componente.

2.3 Questões de Concurso

- 1 CESPE TRT-ES 2009 A interface é uma coleção de operações que pode especificar serviços de uma classe ou componente.

2.3 Questões de Concurso

- 2 CESPE ANAC 2009 Em Java, se uma subclasse é derivada de uma superclasse com um método abstract sem fornecer uma definição para esse método abstract na subclasse, esse método permanece abstract e os objetos instanciados a partir dessa subclasse não poderão utilizar o método abstract.

2.3 Questões de Concurso

- 2 CESPE ANAC 2009 Em Java, se uma subclasse é derivada de uma superclasse com um método abstract sem fornecer uma definição para esse método abstract na subclasse, esse método permanece abstract e os objetos instanciados a partir dessa subclasse não poderão utilizar o método abstract.

2.3 Questões de Concurso

- 3 Cesgranrio CMB 2012 Uma diferença entre classe e interface é que a classe

a) pode ter um campo de dados, enquanto a interface não.

b) pode ser implementada em uma interface, enquanto a interface não pode ser implementada em uma classe

c) é usada apenas em applets, enquanto a interface é usada apenas em aplicativos dedicados.

d) é catalogada em pacotes, enquanto a interface não.

e) declara e implementa seus métodos, enquanto a interface apenas declara.

2.3 Questões de Concurso

- 3 Cesgranrio CMB 2012 Uma diferença entre classe e interface é que a classe

a) pode ter um campo de dados, enquanto a interface não.

b) pode ser implementada em uma interface, enquanto a interface não pode ser implementada em uma classe

c) é usada apenas em applets, enquanto a interface é usada apenas em aplicativos dedicados.

d) é catalogada em pacotes, enquanto a interface não.

e) declara e implementa seus métodos, enquanto a interface apenas declara.

2.3 Questões de Concurso

- 4 Cesgranrio Petrobrás 2012 Ao escrever o código da Classe PortaDeCofre em Java para que ela atenda a interface Porta, como um programador deve começar a declaração da classe?

a) public class Porta:PortaDeCofre {

b) public class PortaDeCofre :: Porta {

c) public class PortaDeCofre inherits Porta {

d) public class PortaDeCofre extends Porta {

e) public class PortaDeCofre implements Porta {

2.3 Questões de Concurso

- 4 Cesgranrio Petrobrás 2012 Ao escrever o código da Classe PortaDeCofre em Java para que ela atenda a interface Porta, como um programador deve começar a declaração da classe?

a) public class Porta:PortaDeCofre {

b) public class PortaDeCofre :: Porta {

c) public class PortaDeCofre inherits Porta {

d) public class PortaDeCofre extends Porta {

e) public class PortaDeCofre implements Porta {

2.3 Questões de Concurso

- 5 Cesgranrio BNDES 2009 Qual das afirmações a seguir faz uma apreciação correta a respeito da linguagem de programação Java?

a) O conceito de herança múltipla é implementado nativamente.

b) Uma classe pode implementar somente uma interface ao mesmo tempo.

c) Uma classe pode implementar uma interface ou ser subclasse de outra classe qualquer, mas não ambos simultaneamente.

d) A construção de um método que pode levantar uma exceção, cuja instância é uma subclasse de java.lang.RuntimeException, não exige tratamento obrigatório por parte do programador dentro daquele método.

e) Objetos da classe java.lang.String têm comportamento otimizado para permitir que seu valor seja alterado sempre que necessário, liberando imediatamente a memória usada pelo conteúdo anterior.

2.3 Questões de Concurso

- 5 Cesgranrio BNDES 2009 Qual das afirmações a seguir faz uma apreciação correta a respeito da linguagem de programação Java?

a) O conceito de herança múltipla é implementado nativamente.

b) Uma classe pode implementar somente uma interface ao mesmo tempo.

c) Uma classe pode implementar uma interface ou ser subclasse de outra classe qualquer, mas não ambos simultaneamente.

d) A construção de um método que pode levantar uma exceção, cuja instância é uma subclasse de java.lang.RuntimeException, não exige tratamento obrigatório por parte do programador dentro daquele método.

e) Objetos da classe java.lang.String têm comportamento otimizado para permitir que seu valor seja alterado sempre que necessário, liberando imediatamente a memória usada pelo conteúdo anterior.

2.3 Questões de Concurso

- Gabarito
  - 1 C
  - 2 E
  - 3 E
  - 4 E
  - 5 D

3 Enums

3.1 Introdução

- Você utiliza enums para criar um conjunto válido de valores para um atributo ou método
- Com Enum você garante que seu programa terá apenas valores válidos

3.1 Introdução

- Exemplo de Enum

```
1 public enum TipoCliente {
2     INDIVÍDUO,
3     ORGANIZAÇÃO
4 }
```

3.1 Introdução

- Uso de uma Enum

```
1 public class Cliente {
2     public String nomeCliente;
3     public TipoCliente tipoCliente;
4     public String endereco;
5 }
```

- Atribuição de valor a uma Enum

```
1 Cliente cliente = new Cliente();
2 Cliente.tipoCliente = TipoCliente.INDIVÍDUO;
```

3.2 Enums em uma Classe

- Enum dentro de uma classe

```
1 public class Forma {
2  private enum TipoForma {
3     RETÂNGULO, TRIÂNGULO, CÍRCULO};
4  private TipoForma tipo = TipoForma.CÍRCULO;
5  public String toString() {
6     if (this.tipo == tipoForma.RETÂNGULO) {
7        return "Forma de retângulo";}
8     if (this.tipo == tipoForma.TRIÂNGULO) {
9        return "Forma de triângulo";}
0     return "Forma de círculo";}}
```

3.2 java.lang.Enum

- Quando você define uma Enum, o javac cria uma definição de classe que extende de java.lang.Enum
- Particulariedades da classe
  - Não existe construtor público nem a possibilidade de instanciação
  - Existe apenas uma instância para cada valor da Enum

3.3 Iterando Enum

- Você utiliza um for e o método values da classe Enum

```
1 for (TipoCliente tipoCliente :
TipoCliente.values())
{
2     System.out.println(tipoCliente);
3 }
```

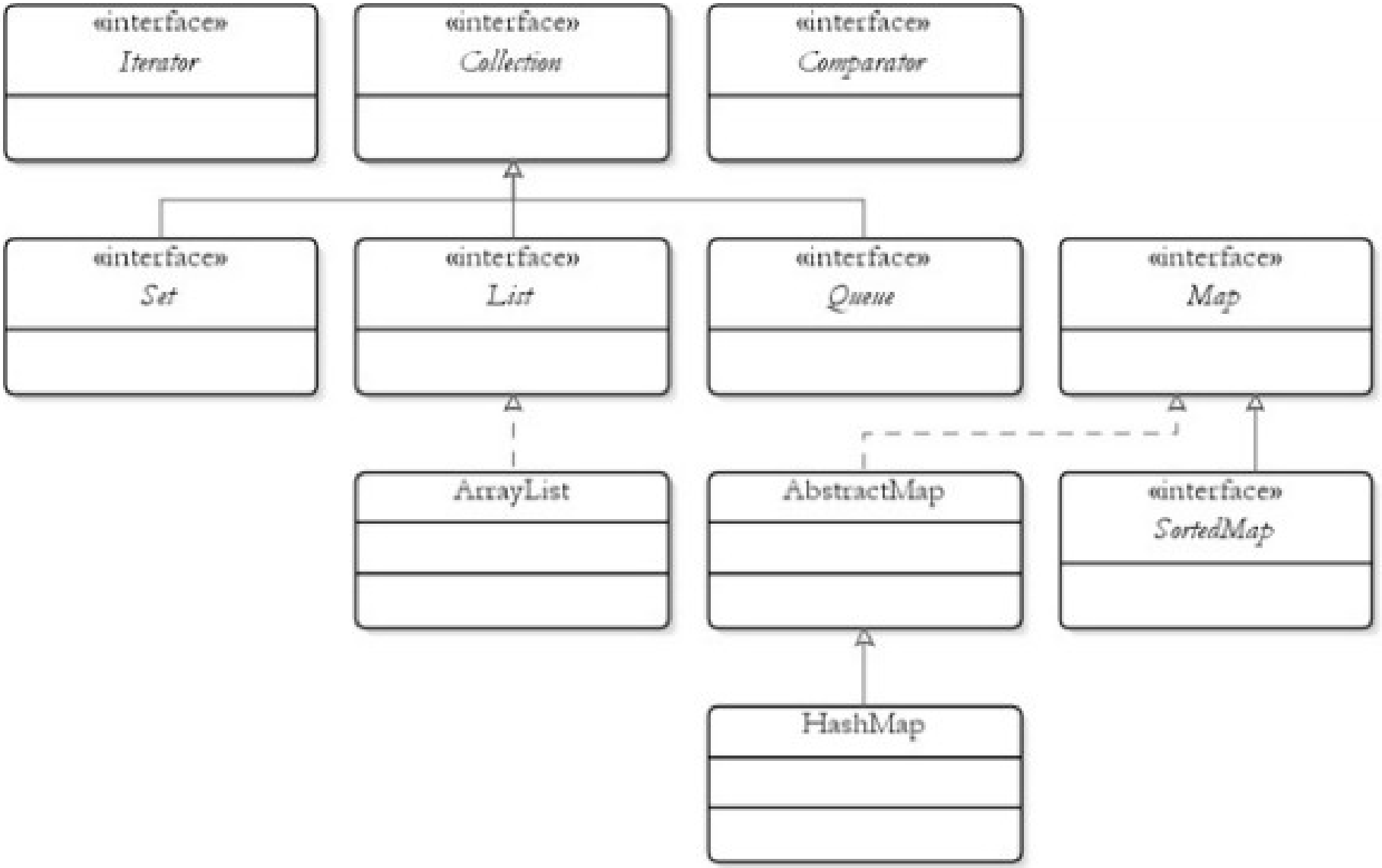
3.3 Enum e Switch

- Exemplo

```
1 Cliente c = new Cliente();
2 c.tipoCliente = TipoCliente.INDIVÍDUO;
3 switch (c.tipoCliente) {
4  case INDIVÍDUO:
5   System.out.println("Tipo: Indivíduo");
6   break;
7  case ORGANIZAÇÃO:
8   System.out.println("Tipo: Organização");
9   break;}
```

4 Coleções

4.1 Introdução



4.1 Introdução

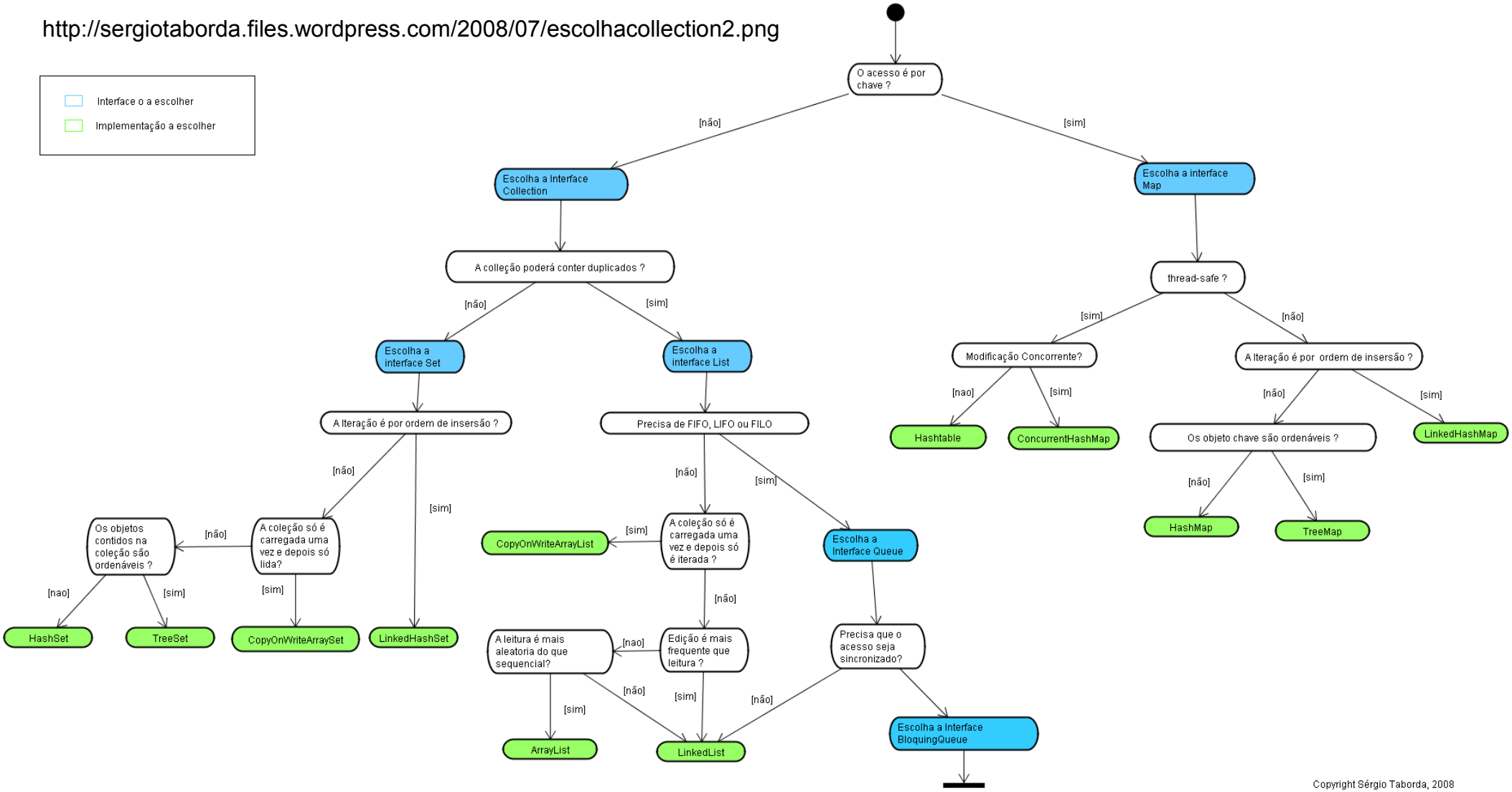
Interface	Descrição
`Collection`	A interface-raiz na hierarquia de coleções a partir da qual as interfaces `Set`, `Queue` e `List` são derivadas.
`Set`	Uma coleção que não contém duplicatas.
`List`	Uma coleção ordenada que pode conter elementos duplicados.
`Map`	Associa chaves a valores e não pode conter chaves duplicadas.
`Queue`	Em geral, uma coleção primeiro a entrar, primeiro a sair que modela uma fila de espera; outras ordens podem ser especificadas.

4.1 Introdução

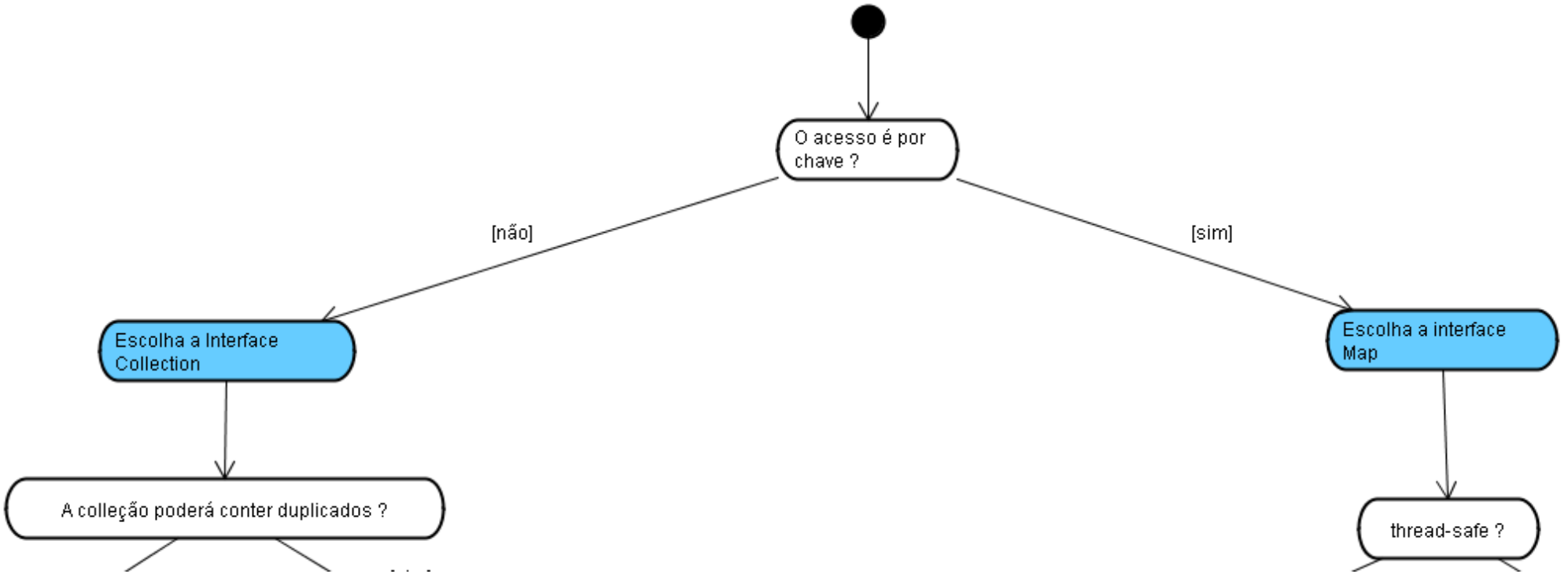
- A maioria das interfaces possuem uma implementação completa
- Muitas vezes, temos duas versões: uma sincronizada e outra não
- Exemplo: Para a interface List, temos java.util.Vector e ArrayList

4.1 Introdução

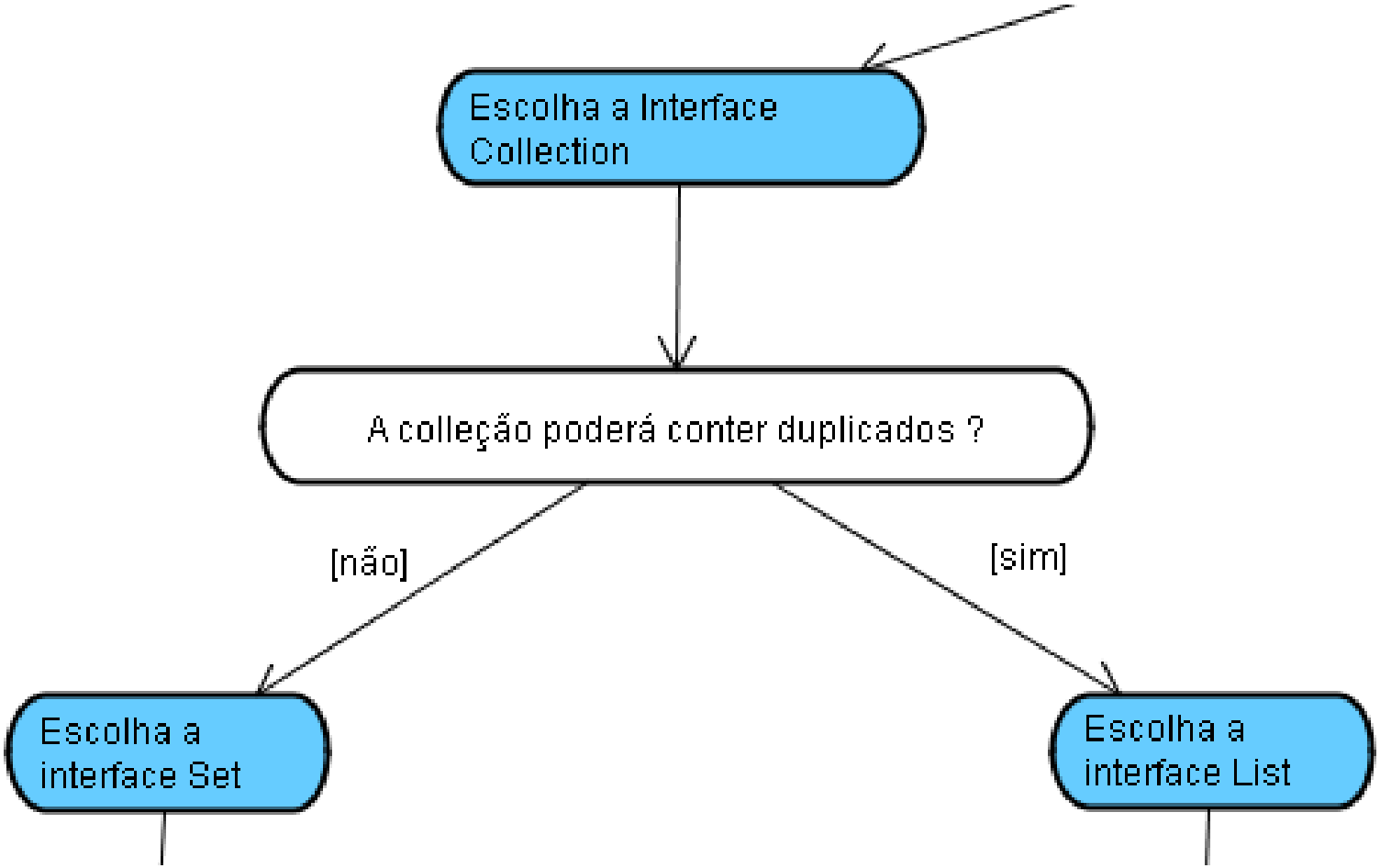
http://sergiotaborda.files.wordpress.com/2008/07/escolhacollection2.png



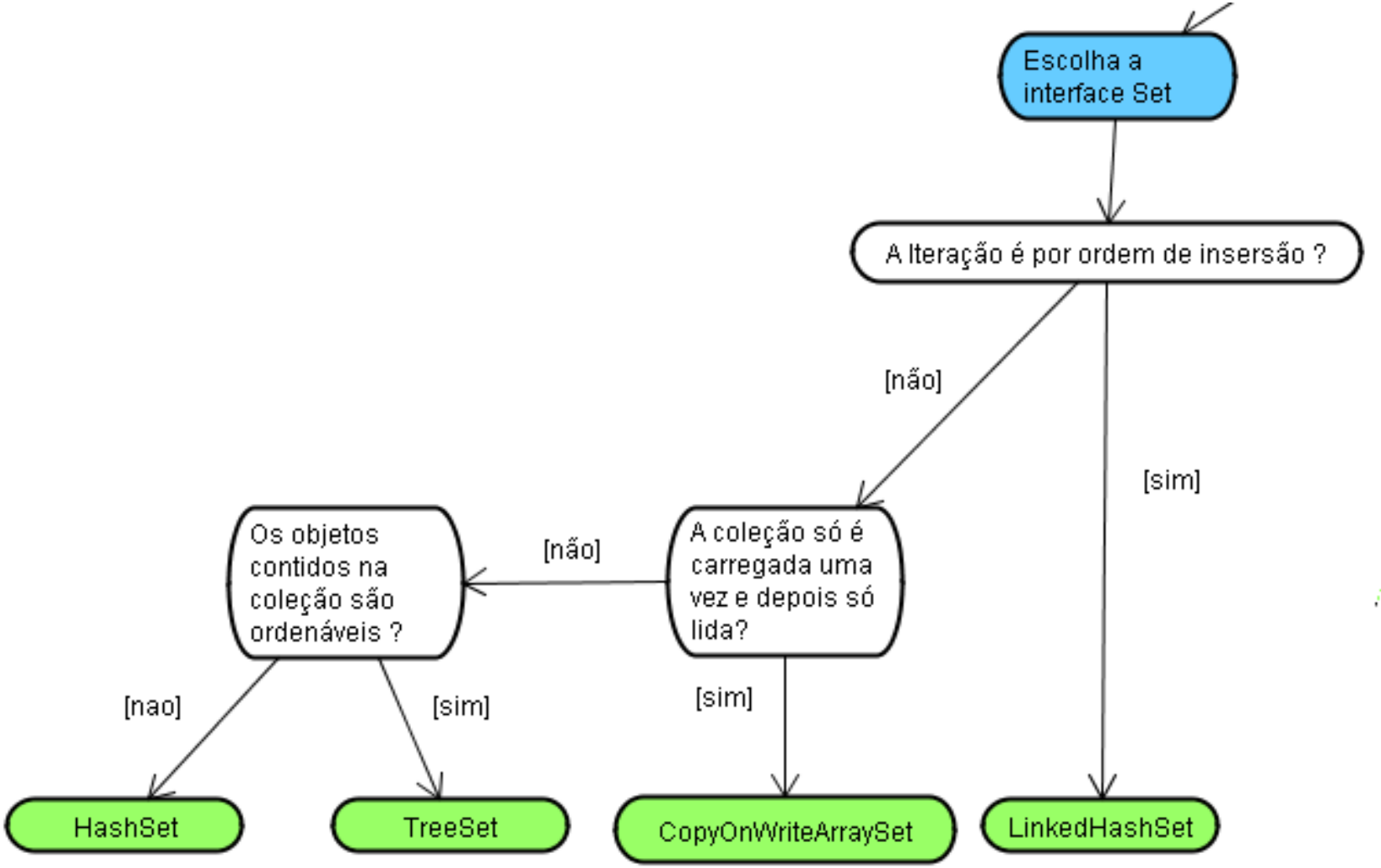
4.1 Introdução

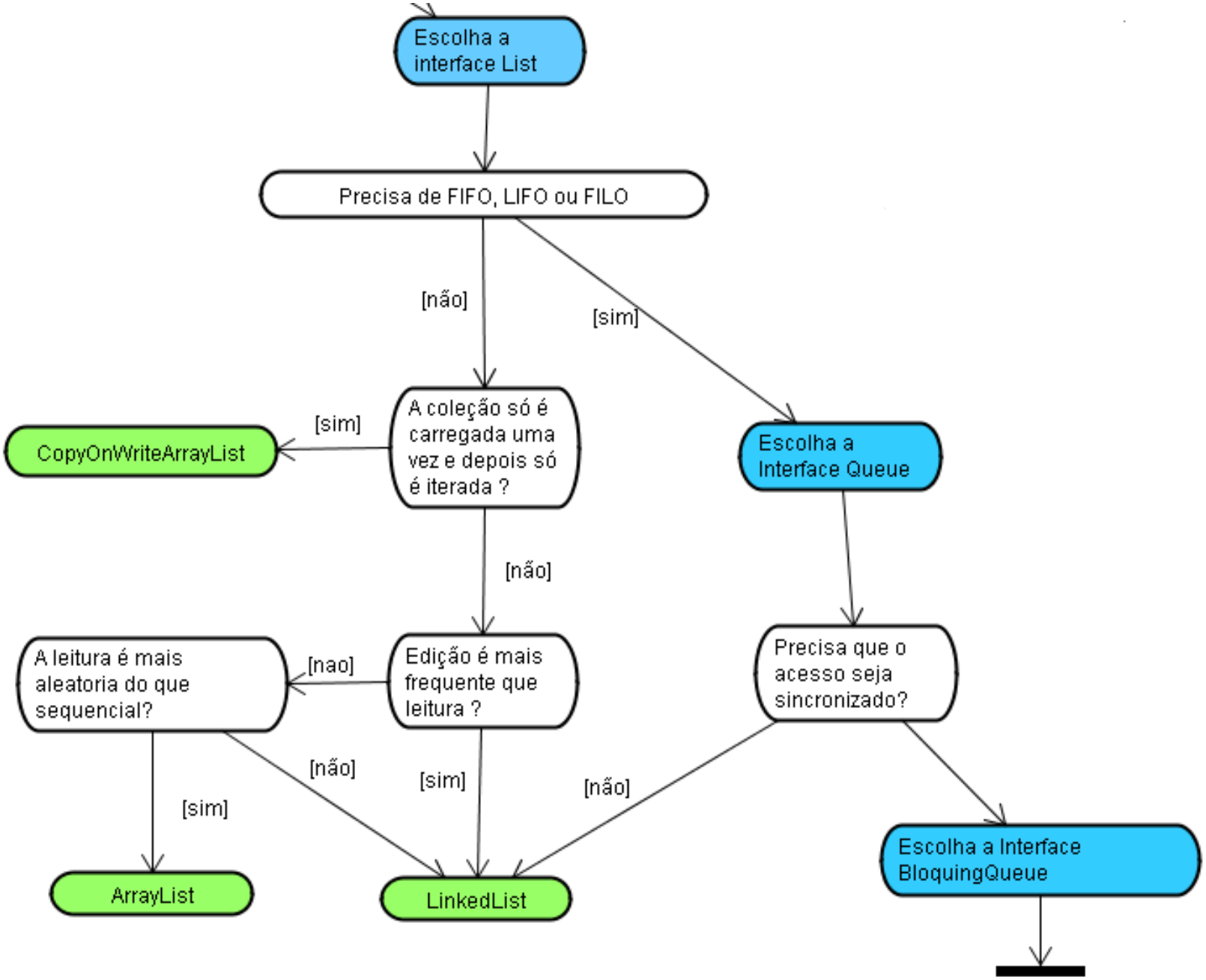


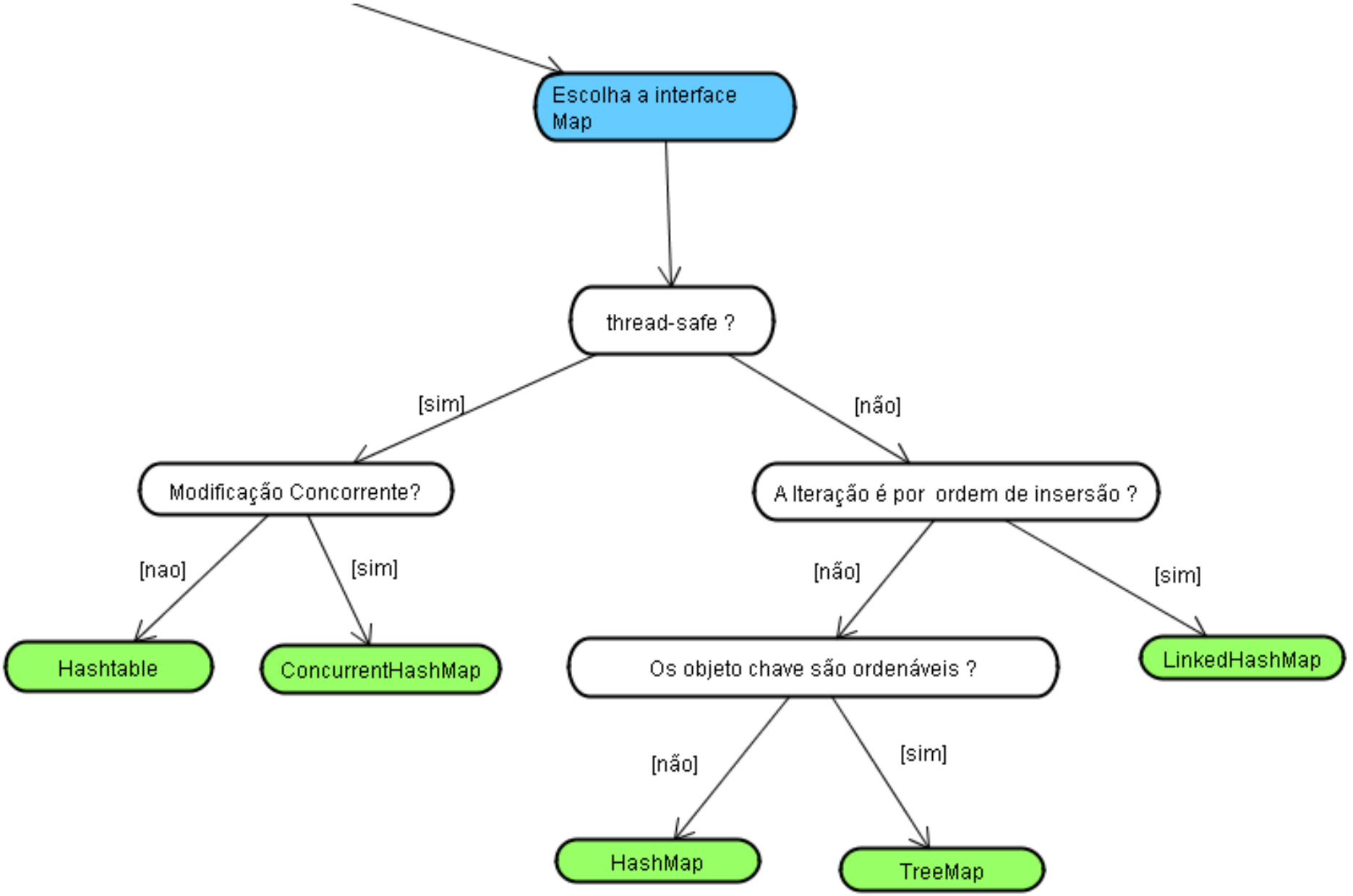
4.1 Introdução



4.1 Introdução




Escolha a interface List
Precisa de FIFO, LIFO ou FILO
[não]
[sim]
A coleção só é carregada uma vez e depois só é iterada ?
[sim]
CopyOnWriteArrayList
Escolha a Interface Queue
[não]
A leitura é mais aleatoria do que sequencial?
[nao]
Edição é mais frequente que leitura ?
Precisa que o acesso seja sincronizado?
[não]
[sim]
[não]
[sim]
[não]
ArrayList
LinkedList
Escolha a Interface BloquingQueue


Escolha a interface Map
thread-safe ?
[sim]
[não]
Modificação Concorrente?
A Iteração é por ordem de insersão ?
[nao]
[sim]
[não]
[sim]
Hashtable
ConcurrentHashMap
Os objeto chave são ordenáveis ?
LinkedHashMap
[não]
[sim]
HashMap
TreeMap

4.1 Introdução

- As interfaces possuem uma implementação concreta
- Muitas vezes, temos duas versões: uma sincronizada e outra não
- Exemplo: Para a interface List, temos java.util.Vector e ArrayList

4.2 A Interface Collection

- Uma Collection é um conjunto de objetos de tamanho variável e que pode ter objetos diferentes
- Métodos
  - add insere um elemento na coleção
  - clear tira todos os elementos
  - size retorna o número de elementos
  - isEmpty testa se a coleção é vazia
  - toArray move os elementos para um Array

4.2 A Interface Collection

- Mais métodos

Método	Descrição
`min`	Retorna o menor elemento em uma `Collection`.
`max`	Retorna o maior elemento em uma `Collection`.
`addAll`	Acrescenta todos os elementos em um array a uma `Collection`.
`frequency`	Calcula quantos elementos da coleção são iguais ao elemento especificado.
`disjoint`	Determina se duas coleções não têm nenhum elemento em comum.

4.1 List e ArrayList

- Coleção ordenada, cujo primeiro elemento está na posição 0.
- Métodos de list:

```
public boolean add(java.lang.Object element)
public void add(int ind, java.lang.Object e)
public java.lang.Object set(int ind,
java.lang.Object e)
public java.lang.Object remove(int ind)
```

4.1 List e ArrayList

- Para criar um List, você atribui um objeto ArrayList para uma variável de referência List

```
1 List myList = new ArrayList(20);
```

- Listas permitem objetos duplicados (ou duas referências apontando para o mesmo objeto)

```
01 import java.util.ArrayList;
02 import java.util.List;
03 public class ListTeste {
04  public static void main(String[] args) {
05   List myList = new ArrayList();
06   String s1 = "Hello";
07   String s2 = "Hello";
08   myList.add(100);
09   myList.add(s1);
10   myList.add(s2);
11   myList.add(s1);
12   myList.add(1);
13   myList.add(2, "World");
14   myList.set(3, "Opa");
15   myList.add(null);
16   System.out.println(myList.size());
17   for (Object object : myList) {
18      System.out.println(object);}}}
```

```
6
100
Hello
World
Opa
Hello
1
null
```

4.1 List e ArrayList

- Em java.util.Arrays tem o método asList que permite adicionar vários elementos em uma lista de uma vez

```
1 List l = Arrays.asList("Chuck", "Norris");
```

Métodos para pesquisar determinado objeto

```
public int indexOf(java.lang.Object o)
public int lastIndexOf(java.lang.Object o)
```

4.2 Iterando coleções

- Existem duas formas, a mais fácil e prática é com o for

```
1 for (Object object : myList) {
2     System.out.println(object);
3 }
```

4.2 Iterando coleções

- Collection é subclasse de Iterable que tem um método iterator
- Este método retorna um objeto java.util.Iterator
- Iterator tem os métodos:
  - hasNext (retorna true se existe o próximo elemento)
  - next (move para o próximo elemento e o retorna)
  - Remove (retira o elemento)

4.2 Iterando coleções

- Na prática

```
1 Iterator iterator = myList.iterator();
2 while (iterator.hasNext()) {
3  String s = (String) iterator.next();
4  System.out.println(s);
5 }
```

4.3 Set e HashSet

- Set é um conjunto matemático que não permite objetos duplicados

```
1 Set set = new HashSet();
2 set.add("Hello");
3 if (set.add("Hello")) {
4       System.out.println("sucesso");
5 } else {
6       System.out.println("xiii");
7 }
```

4.3 Set e HashSet

- HashSet é a implementação de Set mais utilizada porque é mais rápida
- Permite um único null entre seus objetos
- Não a garantia de que os elementos estejam ordenados
- Outras implementações: TreeSet e LinkedHashSet

4.4 Queue e LinkedList

- Queue é uma Collection FIFO
- Métodos
  - offer – funciona como o add, mas sem lançar exceção
  - remove – remove um elemento, lança exceção se vazia
  - poll – remove um elemento, retorna null se vazia
  - element – retorna o elemento, mas não remove (exceção)
  - peek, retorna o elemento, mas não remove (null)

4.4 Queue e LinkedList

- Exemplo:

```
1 Queue queue = new LinkedList();
2 queue.add("a");
3 queue.add("b");
4 queue.add("c");
5 System.out.println(queue.remove());
6 System.out.println(queue.remove());
7 System.out.println(queue.remove());
8 //a b c
```

4.5 Conversão de Coleções

- Coleções possuem um construtor que aceitam uma Collection

```
public ArrayList(Collection c)
public HashSet(Collection c)
public LinkedList(Collection c)
```

```
1 List l = new ArrayList();
2 myList.add("Hello");
3 myList.add("World");
4 myList.add("World");
5 Set set = new HashSet(l);
```

4.6 Map e HashMap

- Um Map constitui uma série de pares de chaves/valores
- Métodos
  - put – coloca uma par chave/valor
  - putAll – coloca um Map em outro
  - remove – passa a chave e remove o par
  - clear, size, isEmpty
  - keySet – retorna um set com todas as chaves
  - values – retorna uma Collection com todos valores

4.6 Map e HashMap

- Exemplo:

```
1 Map map = new HashMap();
2 map.put("1", "hum");
3 map.put("2", "dois");
4 System.out.println(map.size());
5 System.out.println(map.get("1"));
6 Set keys = map.keySet();
7 for (Object object : keys) {
8  System.out.println(object);
9 }
```

4.7 Questões de Concurso

- 1 CESPE 2012 BASA O código de classe a seguir é adequado para converter, em Java, um array de string para um arraylist.

```
import java.util.*;
public class prova {
        public static void main(String[] args) {
            List minhaLista = new ArrayList();
            String[] palavras = new String[] {"Java", "é", "Legal"};
            Collections.add(minhaLista, palavras);
            System.out.println(minhaLista);}}
```

4.7 Questões de Concurso

- 1 CESPE 2012 BASA O código de classe a seguir é adequado para converter, em Java, um array de string para um arraylist.

```
import java.util.*;
public class prova {
        public static void main(String[] args) {
            List minhaLista = new ArrayList();
            String[] palavras = new String[] {"Java", "é", "Legal"};
            Collections.add(minhaLista, palavras);
            System.out.println(minhaLista);}}
```

4.7 Questões de Concurso

- 2 FCC TRF 2012 No Java, é uma interface que não permite elementos duplicados e modela a abstração matemática de conjunto. Contém apenas métodos herdados da interface Collection e adiciona a restrição de que elementos duplicados são proibidos.

A interface citada é:

a) List.

b) Set.

c) ArrayList.

d) Map.

e) HashMap.

4.7 Questões de Concurso

- 2 FCC TRF 2012 No Java, é uma interface que não permite elementos duplicados e modela a abstração matemática de conjunto. Contém apenas métodos herdados da interface Collection e adiciona a restrição de que elementos duplicados são proibidos.

A interface citada é:

a) List.

b) Set.

c) ArrayList.

d) Map.

e) HashMap.

4.7 Questões de Concurso

- 3 FCC 2009 TJ-SE Uma lista Java é uma coleção ordenada de elementos do mesmo tipo, conhecida por sequência. Os elementos de uma lista podem ser acessados pela sua posição, isto é, seu índice e são derivados da interface

a) java.util.LinkedList, que estende a interface Collection.

b) java.util.Collection, que estende a interface Set.

c) java.util.Set, que estende a interface Collection.

d) java.util.Collection, que estende a interface List.

e) java.util.List, que estende a interface Collection.

4.7 Questões de Concurso

- 3 FCC 2009 TJ-SE Uma lista Java é uma coleção ordenada de elementos do mesmo tipo, conhecida por sequência. Os elementos de uma lista podem ser acessados pela sua posição, isto é, seu índice e são derivados da interface

a) java.util.LinkedList, que estende a interface Collection.

b) java.util.Collection, que estende a interface Set.

c) java.util.Set, que estende a interface Collection.

d) java.util.Collection, que estende a interface List.

e) java.util.List, que estende a interface Collection.

4.7 Questões de Concurso

- 4 FCC 2009 PGE-RJ A interface Map do fremework de coleções da linguagem Java retorna o valor associado a uma chave especificada por meio do método

a) Object get(Object key).

b) Object put(Object key, Object value).

c) int size( ).

d) Object firstKey( ).

e) boolean containsKey(Objet key).

4.7 Questões de Concurso

- 4 FCC 2009 PGE-RJ A interface Map do fremework de coleções da linguagem Java retorna o valor associado a uma chave especificada por meio do método

a) Object get(Object key).

b) Object put(Object key, Object value).

c) int size( ).

d) Object firstKey( ).

e) boolean containsKey(Objet key).

4.7 Questões de Concurso

- 5 FCC 2009 TRT Em Java, uma Collection que não contém elementos duplicados é a interface

a) SET.

b) MAP.

c) LIST.

d) ITERATOR.

e) ENUMERATION.

4.7 Questões de Concurso

- 5 FCC 2009 TRT Em Java, uma Collection que não contém elementos duplicados é a interface

a) SET.

b) MAP.

c) LIST.

d) ITERATOR.

e) ENUMERATION.

4.7 Questões de Concurso

- Gabarito
  - 1 E
  - 2 B
  - 3 E
  - 4 A
  - 5 A

5 Generics

5.1 Introdução

- Com Generics você cria tipos e métodos parametrizados
- Benefícios
  - Checagem de tipo em tempo de compilação
  - Eliminação da necessidade de Casting

5.1 Introdução

- A vida antes da JDK 5

```
1 List l = new ArrayList();
2 l.add("Provas de TI");
3 l.add("Bom demais");
4 String s1 = (String) l.get(0);
```

5.2 Como funciona?

- Exemplo

```
1 import java.util.List;
2 import java.util.ArrayList;
3 public class GenericsListTeste {
4  public static void main(String[] args) {
5    List<String> l = new ArrayList<>();
6    l.add("Java");
7    l.add("com generics");
8    String s2 = l.get(0);
9    System.out.println(s2.toUpperCase());}}
```

5.2 Como funciona?

```
1 List<List<String>> minhaListadeListas;
(…)
2 String s = minhaListadeListas.get(0).get(0);
```

5.2 Como funciona?

- Maps Generics

```
1 import java.util.HashMap;
2 import java.util.Map;
3 public class MapTeste {
4  public static void main(String[] args) {
5     Map<String, String> m = new HashMap<>();
6     m.put("key1", "Provas de TI");
7     m.put("key2", "Salvação");
8     String s = m.get("key1");}}
```

- Uso do ?

```
01 import java.util.ArrayList;
02 import java.util.List;
03 public class CoringaTeste {
04  public static void printList(List<?> l) {
05     for (Object element : l) {
06        System.out.println(element);}}
07  public static void main(String[] args) {
08     List<String> l1 = new ArrayList<>();
09     l1.add("Hello");
10     l1.add("World");
11     printList(l1);
12     List<Integer> l2 = new ArrayList<>();
13     l2.add(100);
14     list2.add(200);
15     printList(l2);}}
```

- ? delimitado

```
01 import java.util.ArrayList;
02 import java.util.List;
03 public class CoringaTeste {
04  public static double getMedia(List<?
   extends Number> l) {
05    double total = 0.0;
06    for (Number n : l) {
07     total += n.doubleValue();}
08    return total/l.size();}
09  public static void main(String[] args) {
10   List<Integer> l1 = new ArrayList<>();
11   l1.add(3);
12   l1.add(30);
13   l1.add(300);
14   System.out.println(getMedia(l1)); // 111
15 }}
```

5.3 Escrevendo Tipos Genéricos

- Você pode criar classes Generics

```
01 public class Ponto<T> {
02     T x;
03     T y;
04     public Ponto(T x, T y) {
05         this.x = x;
06         this.y = y;}
07     public T getX() {
08         return x;}
09     public T getY() {
10         return y;}
11     public void setX(T x) {
12         this.x = x;}
13     public void setY(T y) {
14         this.y = y;}}
```

5.3 Escrevendo Tipos Genéricos

- Agora é só usar

```
1 Ponto<Integer> p1 = new Ponto<>(4, 2);
2 p1.setX(7);
3 Ponto<Double> p2 = new Ponto<>(1.3, 2.6);
4 p2.setX(109.91)
```

5.4 Questões de Concurso

- 1 CESPE 2012 ANAC O código abaixo irá compilar e retornar o número 70.
  import java.util.*
  public class OutTeste {
  - public static void main(String[] args) {
    - List<Integer> list = new ArrayList<Integer>();
    - list.add(0, 70);
    - int total = list.get(1);
    - System.out.println(total);
  - }
- }

5.4 Questões de Concurso

- 1 CESPE 2012 ANAC O código abaixo irá compilar e retornar o número 70.
  import java.util.*
  public class OutTeste {
  - public static void main(String[] args) {
    - List<Integer> list = new ArrayList<Integer>();
    - list.add(0, 70);
    - int total = list.get(1);
    - System.out.println(total);
  - }
- }

5.4 Questões de Concurso

- 2 Cesgranrio Petrobrás 2012 Uma aplicação Java precisa manter na memória principal do computador uma coleção de objetos com as seguintes características:
- poderá conter dezenas de milhares de objetos;
- seus objetos não estarão ordenados;
- um número considerável de objetos poderá ser inserido em tempo de execução;
- a operação mais executada será o percurso sequencial na ordem inversa de inserção dos objetos na coleção.

Diante dessas características, qual das classes irá proporcionar à aplicação a melhor performance em relação à manipulação dessa coleção?

a) LinkedList <E> b) ArrayList <E> c) HashSet <E> d) HashMap <K,V> e) TreeSet <E>

5.4 Questões de Concurso

- 2 Cesgranrio Petrobrás 2012 Uma aplicação Java precisa manter na memória principal do computador uma coleção de objetos com as seguintes características:
- poderá conter dezenas de milhares de objetos;
- seus objetos não estarão ordenados;
- um número considerável de objetos poderá ser inserido em tempo de execução;
- a operação mais executada será o percurso sequencial na ordem inversa de inserção dos objetos na coleção.

Diante dessas características, qual das classes irá proporcionar à aplicação a melhor performance em relação à manipulação dessa coleção?

a) LinkedList <E> b) ArrayList <E> c) HashSet <E> d) HashMap <K,V> e) TreeSet <E>

5.4 Questões de Concurso

- Gabarito
  - 1 E
  - 2 A

6 Classes Internas

6.1 Introdução

- Temos 04 tipos de classes internas:
  - Classe estática aninhada (static nested class)
  - Classe não estática aninhada
    - Classe interna comum (member inner class)
    - Classe interna local (local inner class)
    - Classe interna anônima (anonymous inner class)

6.1 Introdução

- Exemplo básico:

```
1 public class Externa {
2     class Interna {
3     }
4 }
```

6.2 Classes Estáticas Aninhadas

```
1 class Externa {
2  private static int valor = 9;
3  static class Interna {
4  int calcular() {
5     return valor;}}}
6
7 public class staticTeste {
8  public static void main(String[] args) {
9   Externa.Interna i = new Externa.Interna();
0     System.out.println(i.calcular());}}
```

6.3 Classes Internas Comuns

```
1 class TopLevel {
2  private int valor = 9;
3  class Inner {
4   int calcular() {
5    return valor;}}}

6 public class MemberInnerTeste {
7  public static void main(String[] args) {
8   TopLevel t = new TopLevel ();
9   TopLevel.Inner i = t.new Inner();
0   System.out.println(i.calcular());}}
```

6.4 Classes Internas Locais

```
01 import java.util.Date;
02 interface Log {
03  public void log(String m);}
04 public class LocalClassTeste {
05  String a = (new Date()).toString();
06  public Log getLog() {
07   class LogImpl implements Log {
08    public void log(String m) {
09     System.out.println(a + " : " + m);}}
10   return new LogImpl();}
11 public static void main(String[] args) {
12  LocalClassTeste t = new LocalClassTeste();
13  Log log = t.getLog();
14  log.log("Exemplo de classe local");}}
```

6.5 Classes Internas Anônimas

```
1 interface Printable {
2  void print(String m);}

3 public class AnonymousInnerClassTeste{
4  public static void main(String[] args) {
5   Printable p = new Printable() {
6    public void print(String m) {
7     System.out.println(m);}};
8   p.print("Java");}}
```

6.6 Questões de Concurso

- 1 VUNESP 2010 CEAGESP Considere o código a seguir, escrito na linguagem de programação JAVA.

```
public class Testing {
   private static int i = 0;
   public static void go(){System.out.print(i++);}
   public static void main(String args[]){
     new Testing().go(); new Testing().go(); new Testing().go(); new Testing().go();
     new Testing().go2();
   public void go2(){new TestingIntern().run();}
   public class TestingIntern {
     public void run(){System.out.print(i++);}}}
```

- Após a sua execução, o resultado apresentado será:

a) 01234 b) 23456 c) Sintax Error( )

d) ArrayOutOffBound( ) e) NullPointerException( )

6.6 Questões de Concurso

- 1 VUNESP 2010 CEAGESP Considere o código a seguir, escrito na linguagem de programação JAVA.

```
public class Testing {
   private static int i = 0;
   public static void go(){System.out.print(i++);}
   public static void main(String args[]){
     new Testing().go(); new Testing().go(); new Testing().go(); new Testing().go();
     new Testing().go2();
   public void go2(){new TestingIntern().run();}
   public class TestingIntern {
     public void run(){System.out.print(i++);}}}
```

- Após a sua execução, o resultado apresentado será:

a) 01234 b) 23456 c) Sintax Error( )

d) ArrayOutOffBound( ) e) NullPointerException( )

6.6 Questões de Concurso

- 2 FCC 2010 MPE-RN A linguagem Java 6 permite a declaração de uma classe aninhada que se trata de uma classe
- a) genérica.
- b) abstrata.
- c) agregada.
- d) membro de outra.
- e) anônima.

6.6 Questões de Concurso

- 2 FCC 2010 MPE-RN A linguagem Java 6 permite a declaração de uma classe aninhada que se trata de uma classe
- a) genérica.
- b) abstrata.
- c) agregada.
- d) membro de outra.
- e) anônima.

5.4 Questões de Concurso

- Gabarito
  - 1 A
  - 2 D

Fim