Programação Orientada a Objetos em

Flávio Miguel Varejão

Departamento de Informática

UFES

Utilidades

- Pacotes com várias classes amplamente utilizadas
- Exemplos
  - Vector, ArrayList, Calendar, Currency, Random, Formatter, Scanner, Comparator, Iterator, Enumeration, Properties, StringTokenizer

Coleções

- Uma coleção representa um grupo de objetos (chamados elementos) em um único objeto
- Também conhecidos como containers
- Infraestrutura de coleções incluem
  - Interfaces
    - Permitem a manipulação de diferentes coleções independentemente dos detalhes de representação
  - Implementações
    - Implementações de estruturas de dados amplamente utilizadas
  - Algoritmos
    - Métodos úteis para várias coleções, como sort e search

Coleções

- Algumas Vantagens
  - Redução no esforço de programação
    - Programador pode se concentrar na funcionalidade desejada em vez de se preocupar com a implementação de estruturas de dados
  - Melhoria do desempenho e qualidade
    - Implementação das coleções têm alto desempenho e qualidade
  - Aumenta interoperabilidade de programas
    - Diferentes implementações de coleções podem ser usadas em um mesmo trecho de código sem necessidade de código de adaptação ou conversão

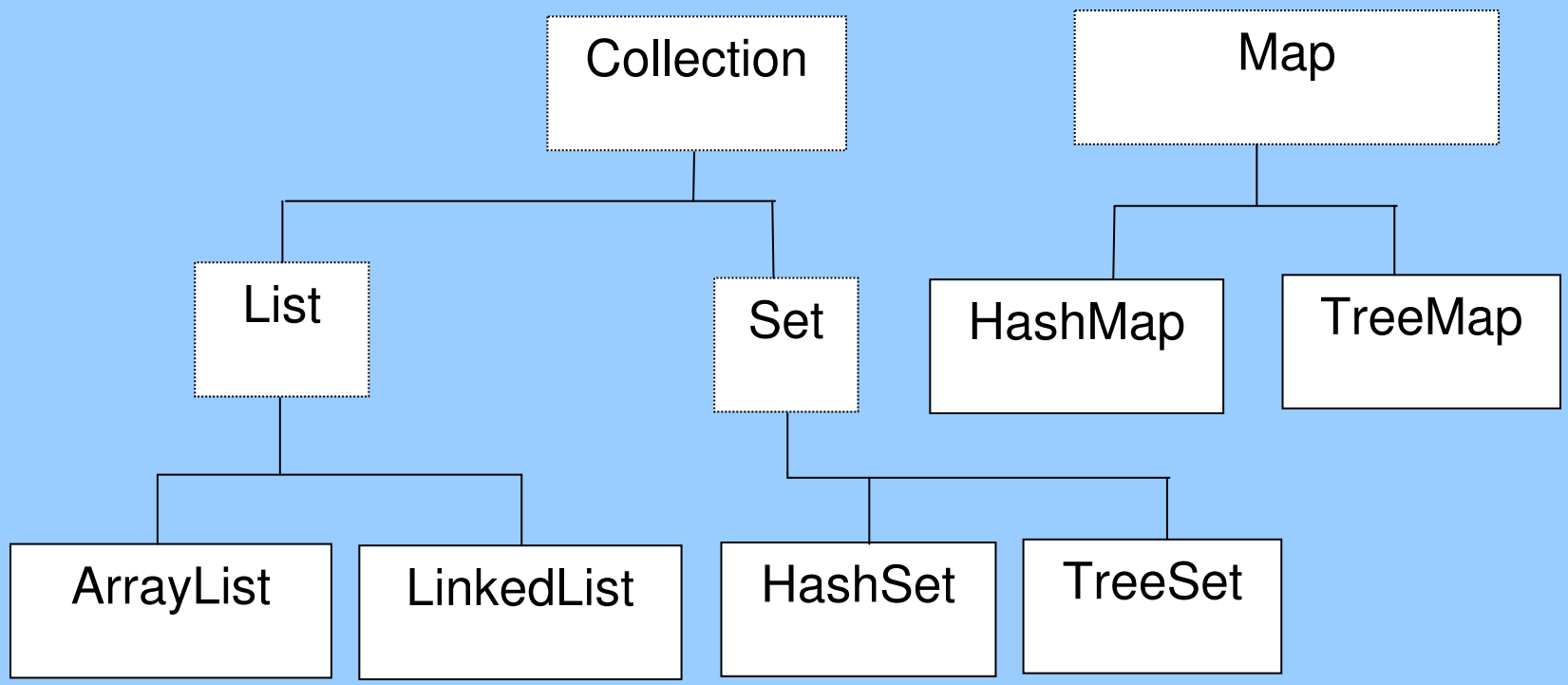
Coleções

- Algumas das interfaces que fazem parte da infraestrutura de coleções
  - Collection
    - Grupo de objetos com ordem não especificada
    - Duplicação permitida
  - List
    - Grupo de objetos ordenados
    - Pode haver duplicação
  - Set
    - Grupo de objetos não ordenados
    - Sem duplicação

Coleções

- Algumas das interfaces que fazem parte da infraestrutura de coleções
  - Map
    - Grupo de pares (chave, valor) de objetos
    - Mapeamentos podem ser heterogêneos
      - Tanto chave quanto valor podem ser de tipo definido arbitrariamente
    - Sem duplicação de chave
    - Uma chave só pode ter um valor
    - Mesmo valor pode ser associado a várias chaves

Coleções


Collection
Map
List
Set
HashMap
TreeMap
ArrayList
LinkedList
HashSet
TreeSet

Listas

```
public class Listas {
  public static void main(String[] args) {
    ArrayList pares = new ArrayList();
    LinkedList impares = new LinkedList();
    for(int i = 0; i < 10; i = i + 2)
      pares.add(new Integer(i));
    pares.add(new String("pares"));
    for(int i = 1; i < 10; i = i + 2)
      impares.add(new Integer(i));
    impares.add(new Float(3.14));
    for(int i = 0; i < pares.size(); i++) {
      System.out.println(pares.get(i));
    }
```

Listas

```
    for(int i = 0; i < impares.size(); i++) {
      System.out.println(impares.get(i));
    }
    Integer j = (Integer) pares.get(0);
    //! Integer j = (Integer) pares.get(5);
    Integer k = (Integer) impares.get(0);
    //! Integer k = (Integer) impares.get(5);

  }
}
```

Listas

```
class ListaInteiros {
  private ArrayList v = new ArrayList();
  public void add(Integer i) {
    v.add(i);
  }
  public Integer get(int ind) {
    return (Integer)v.get(ind);
  }
  public int size() { return v.size(); }
  public static void main(String[] args) {
    ListaInteiros inteiros = new ListaInteiros ();
    for(int i = 0; i < 3; i++)
      inteiros.add(new Integer(i));
    for(int i = 0; i < inteiros.size(); i++)
      System.out.println(inteiros.get(i));
  }
}
```

Iteradores

- Iteração é o processo pelo qual uma coleção é percorrida e todos os seus elementos são acessados
- Iteração sobre conjuntos é não determinística
- Iteração sobre listas pode ser na sequência ou na sequência invertida

Iteradores

```
// Iterador.java
import java.util.*;
public class Iterador {
  static void imprimeTodos(Iterator e) {
    while(e.hasNext())
      System.out.println(e.next());
  }
  public static void main(String[] args) {
    List inteiros = new ArrayList();
    for(int i = 0; i < 7; i++)
      inteiros.add(new Integer(i));
    Iterator iterador = inteiros.iterator();
    imprimeTodos(iterador);
  }
} ///:~
```

Novo for no J2SE 5.0

- Uso de iteradores se tornou amplo e disseminado
- Fazer iteração sobre uma coleção é pouco redigível
- Problema se agravaria com a inclusão de generics
- Solução foi criar uma nova alternativa de comando for

Novo for no J2SE 5.0

```
ArrayList lista = new ArrayList();
Lista.add(new Integer(1));
Lista.add(new Integer(2));
Lista.add(new Integer(3));
for (Iterator i = lista.iterator(); i.hasNext();) {
   Integer value =(Integer)i.next();
}
for (Object i: lista) {
   Integer value = (Integer) i;
}
```

Novo for no J2SE 5.0

```
int soma(int[] a) {
    int resultado = 0;
    for (int i : a){
        resultado += i;
    }
    return resultado;
}
```

Conjuntos

```
import java.util.*;
class Colecao {
  public static Collection preenche(Collection c) {
    for(int i = 0; i < 10; i++)
       c.add(Integer.toString(i));
    return c;
  }
  public static void imprime(Collection c) {
    for(Iterator x = c.iterator(); x.hasNext();)
      System.out.print(x.next() + " ");
    System.out.println();
  }
```

Conjuntos

```
public class Conjuntos {
  public static void testa(Set a) {
    Colecao.preenche(a); Colecao.preenche(a);
    Colecao.preenche(a); Colecao.imprime(a);
    // Nao ha' duplicidade
    a.addAll(a);
    a.add("um"); a.add("um"); a.add("um");
    Colecao.imprime(a);
    System.out.println("a contém \"um\": " +
      a.contains("um"));
  }
  public static void main(String[] args) {
    testa(new HashSet());
    testa(new TreeSet());
  }
}
```

Mapeamentos

```
import java.util.*;
public class Mapas {
  public static void main(String[] args) {
     Map conta_pal_mapa = new HashMap();
     ArquivoTexto leitor = new Arquivotexto(args[0]);
     Iterator palavras =
       new IteradorPalavras(leitor);
     while ( palavras.hasNext() ) {
       String pal = (String) palavras.next();
       String pal_min = pal.toLowerCase();
       // esta é a chave
       Integer freq = (Integer)
         cont_pal_mapa.get(pal_min);
```

Mapeamentos

```
            if (freq == null) {
              freq = new Integer(1);
            } else {
              int valor = freq.intValue();
              freq = new Integer(valor + 1);
            }
            cont_pal_mapa.put(pal_min, freq);
          }
        System.out.println(conta_pal_mapa);
    }
}
```

Propriedades

- É um mapeamento no qual chave e valor são Strings
- Muito usada com aplicações baseadas em XML
- Uso da classe Properties com um arquivo de configurações permite manter a aplicação sem a necessidade de recompilar o código fonte a cada mudança nas configurações
  - Arquivos com extensão .properties
  - Uso de método load()

Arquivo de Configuração

```
#Configurações do arquivo mail.properties
mail.pop3.host=pop.dot.com
mail.debug=false
mail.from=from_user@mail.dot.com
mail.user=to_user
mail.smtp.host=smtp.dot.com
mail.store.protocol=pop3
```

Leitura do Arquivo

```
File file = new File("mail.properties");
Properties props = new Properties();
FileInputStream fis = null;
try {
    fis = new FileOutputStream(file);
    //lê os dados que estão no arquivo
    props.load(fis);
    fis.close(0;
}
catch (IOException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
```

Manipulando Propriedades

```
String user = props.getProperty("mail.user");
String from = props.getProperty("mail.from");
String smtp = props.getProperty("mail.smtp.host");
String pop3 = props.getProperty("mail.pop3.host");
String protocol =
    props.getProperty("mail.store.protocol");
String debug = props.getProperty("mail.debug");

String user = "guj";
props.setProperty("mail.user", user);
```

Atualizando Arquivo

```
File file = new File("mail.properties");
FileInputStream fos = null;
try {
    fos = new FileOutputStream(file);
    //grava os dados no arquivo
    props.store(fos,
       "Configurações do arquivo mail.properties");
    fos.close();
}
catch (IOException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
```

Propriedades do Sistema

```
import java.util.*;
public class TestaPropriedades {
  public static void main(String[] args) {
    Properties props = System.getProperties();
    Enumeration nomes_prop = props.propertyNames();
    while (nomes_prop.hasMoreElements() ) {
       String nome_propriedade = (String)
              nomes_prop.nextElement();
       String propriedade =
              props.getProperty(nome_propriedade);
       System.out.println("propriedade '" +
              nome_propriedade + "' é '" +
              propriedade + "'");
    }
  }
}
```

Enumerações no J2SE 5.0

- Uso de constantes para a representação de tipos enumerados

```
public static final int INVERNO = 0;
public static final int PRIMAVERA = 1;
public static final int VERAO = 2;
public static final int OUTONO = 3;
```

- Problemas
  - Não é seguro: é possível passar qualquer valor inteiro quando uma estação é necessária
  - Os valores não são informativos: São impressos números, o que não diz nada a respeito do que representam ou do tipo que pertencem

Enumerações no J2SE 5.0

- Tipos enumerados foram incorporados a linguagem no J2SE
  - Não pertencem a pacote util

```
enum ESTACAO {INVERNO, PRIMAVERA, VERAO, OUTONO}
```

- A declaração `enum` define uma nova classe
- É possível adicionar métodos e campos a um tipo enumerado e implementar interfaces
- Tipos enumerados possuem implementações dos métodos da classe Object e podem ser comparáveis e serializáveis.

Enumerações no J2SE 5.0

```
import java.util.*;
public class Carteado {
    public enum Cartas {
        DOIS, TRÊS, QUATRO, CINCO, SEIS, SETE, OITO,
        NOVE, DEZ, VALETE, RAINHA, REI, AS
    }
    public enum Naipes { PAUS, OUROS, COPAS, ESPADAS }
    private final Cartas carta;
    private final Naipes naipe;
    private Carteado(Cartas carta, Naipes naipe) {
        this.carta = carta;
        this.naipe = naipe;
    }
    public Cartas carta() { return carta; }
    public Naipes naipe() { return naipe; }
```

Enumerações no J2SE 5.0

```
    public String toString() {
       return carta + " of " + naipe;
    }
    private static final List baralho =
       new ArrayList();
    static {
      for (Naipes naipe : Naipes.values())
        for (Cartas carta : Cartas.values())
          baralho.add(new Carteado(carta, naipe));
    }
    public static ArrayList novoBaralho() {
     return new ArrayList (novoBaralho);
    }
}
```

Enumerações no J2SE 5.0

```
public enum Planeta {
    MERCURIO (3.303e+23, 2.4397e6),
    VENUS    (4.869e+24, 6.0518e6),
    TERRA    (5.976e+24, 6.37814e6),
    MARTE    (6.421e+23, 3.3972e6),
    JUPITER  (1.9e+27,   7.1492e7),
    SATURNO  (5.688e+26, 6.0268e7),
    URANO    (8.686e+25, 2.5559e7),
    NETUNO   (1.024e+26, 2.4746e7),
    PLUTAO   (1.27e+22,  1.137e6);
    private final double massa;
    private final double raio;
    Planeta(double massa, double raio) {
        this.massa = massa;
        this.raio = raio;
    }
```

Enumerações no J2SE 5.0

```
    public static final double G = 6.67300E-11;
    double gravidadeSuperficie() {
        return G * massa / (raio * raio);
    }
    double pesoSuperficie(double outraMassa) {
        return outraMassa * gravidadeSuperficie();
    }
}
```

Enumerações no J2SE 5.0

```
public class UsaEnum {
   public static void main(String[] args) {
     double pesoTerra = double.parseDouble(args[0]);
     double massa =
        pesoTerra/TERRA.gravidadeSuperficie();
     for (Planeta p: Planeta.values()){
       System.out.printf("Seu peso em %s eh %f%n",
            p, p.pesoSuperficie(massa));
     }
   }
}
```

Java Generics no J2SE 5.0

- Polimorfismo paramétrico
  - Código genérico que pode ser aplicado a diferentes tipos de dados
    - Listas
  - Tipos parametrizados
    - Tipo é um parâmetro para o código
  - Permite eliminar a necessidade de downcast na maior parte das situações
  - Evita a inclusão de elementos de tipos inadequados na compilação

Java Generics

- Sem tipos parametrizados

```
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) myIntList.iterator().next();
```

- Com tipos parametrizados

```
List<Integer> myIntList = new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

Definição de Generics

```
public interface List<E> {
    void add(E x);
    Iterator<E> iterator();
}
public interface Iterator<E> {
    E next();
    boolean hasNext();
}
```

Generics e Subtipos

- Tipo do elemento não é levado em conta na hierarquia de classes

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls; // nao e permitido
...
lo.add(new Object());
String s = ls.get(0);
```

Generics e Código Polimórfico

- Antes de J2SE 5.0

```
void printCollection(Collection c) {
    Iterator i = c.iterator();
    for (k = 0; k < c.size(); k++) {
        System.out.println(i.next());
    }
}
```

Generics e Código Polimórfico

- Generics restringiria
  - Deixa de ser polimórfico
  - Só coleções de objetos

```
void printCollection(Collection<Object> c) {
    for (Object e: c) {
        System.out.println(e);
    }
}
```

Generics e Código Polimórfico

- Generics com wildcards (coringas)
  - Restaura polimorfismo

```
void printCollection(Collection<?> c) {
    for (Object e: c) {
        System.out.println(e);
    }
}
```

  - Código com wildcards não permitem a adição de elementos

```
Collection<?> c = new ArrayList<String>();
c.add(new Object()); // erro em tempo de compilação
```

Wildcards

- Wildcards podem ser muito amplos

```
public abstract class Forma {
    public abstract void desenhar();
}
public class Circulo extends Forma {
    private int x, y, raio;
    public void desenhar() {
       System.out.println(“Circulo”);
    }
}
public class Retangulo extends Forma {
    private int x, y, largura, altura;
    public void desenhar() {
       System.out.println(“Forma”);
    }
}
```

Wildcards Limitados

- Sem wildcards não funciona como desejado

```
public void desenhaTudo(List<Forma> formas) {
    for (Forma f: formas) {
        f.desenhar();
    }
}
```

- Com wildcards permite o uso com listas de qualquer tipo de elemento

```
public void desenhaTudo(List<?> formas) {
    for (Object f: formas) {
        ((Forma) f).desenhar();
    }
}
```

Wildcards Limitados

- Só funciona com lista de forma ou suas subclasses

```
public void desenhaTudo(List<? extends Forma>
    formas){
   ...
}
```

- Não é possível adicionar elementos

```
public void addRetangulo(List<? extends Forma>
    formas) {
    formas.add(0, new Retangulo()); // erro em
    // tempo de compilação - poderia ser circulo
}
```

Métodos Genéricos

- Não é possível adicionar elementos

```
static void fromArrayToCollection(Object[] a,
                                  Collection<?> c) {
    for (Object o : a) {
        c.add(o); // erro de compilação
    }
}
```

- Coleção de objetos pode ser usada, mas perde verificação em tempo de compilação

```
static void fromArrayToCollection(Object[] a,
                                  Collection<Object> c) {
    for (Object o : a) {
        c.add(o);
    }
}
```

Métodos Genéricos

- Usar parâmetros tipo em métodos genéricos

```
static <T> void fromArrayToCollection(T[] a,
                      Collection<T> c) {
   for (T o : a) {
      c.add(o); // correto!
   }
}
```

Métodos Genéricos

```
Object[] oa = new Object[100];
Collection<Object> co = new ArrayList<Object>();
fromArrayToCollection(oa, co);// T como Object
String[] sa = new String[100];
Collection<String> cs = new ArrayList<String>();
fromArrayToCollection(sa, cs);// T como String
fromArrayToCollection(sa, co);// T como Object
Integer[] ia = new Integer[100];
Float[] fa = new Float[100];
Number[] na = new Number[100];
Collection<Number> cn = new ArrayList<Number>();
fromArrayToCollection(ia, cn);// T como Number
fromArrayToCollection(fa, cn);// T como Number
fromArrayToCollection(na, cn);// T como Number
fromArrayToCollection(na, co);// T como Object
fromArrayToCollection(na, cs);// erro de compilação
```

Métodos Genéricos e Wildcards

- Wildcards podem ser usados em métodos genéricos

```
class Collections {
    public static <T> void copy(List<T> dest,
       List<? extends T> src){...}
}
```

Generics e Código Legado

- Código legado

```
package br.com.MorteIndolor.coisas;
public interface Parte { ...}
public class Inventario {
 public static void addInstrumento(String nome,
      Collection partes) {...}
   public static Instrumento getInstrumento(
      String nome) {...}
}
public interface Instrumento {
   Collection getPartes(); //Retorna coleção de
                           //partes
}
```

Generics e Código Legado

- Código genérico usando código legado

```
package br.com.minhaEmpresa.inventario;
import br.com.MorteIndolor.coisas.*;
public class  Lamina implements Parte { ...}
public class Suporte implements Parte { ...}
public class Main {
  public static void main(String[] args) {
     Collection<Parte> c = new ArrayList<Parte>();
     c.add(new Suporte()) ;
     c.add(new Lamina());
     Inventario.addInstrumento(”Guilhotina”, c);
     Collection<Parte> k =
         Inventario.getInstrumento(
              ”Guilhotina”).getPartes();
  }
}
```

Generics e Código Legado

- Gera alerta de operação não verificada na compilação
- Pode provocar erro em tempo de execução

```
public String brecha(Integer x) {
    List<String> ys = new LinkedList<String>();
    List xs = ys;
    xs.add(x); // alerta “não verificado”
    return ys.iterator().next(); //ClassCastException
}
```

Generics e Código Legado

- Código genérico

```
package br.com.MorteIndolor.coisas;
public interface Parte { ...}
public class Inventario {
    public static void addInstrumento(String nome,
       Collection<Parte> partes) {...}
    public static Instrumento getInstrumento(
       String nome) {...}
}
public interface Instrumento {
    Collection<Parte> getParts();
}
```

Generics e Código Legado

- Código legado usando código genérico

```
package com.minhaEmpresa.inventario;
import br.com.MorteIndolor.coisas.*;
public class Lamina implements Parte { ...}
public class Guilhotina implements Parte { …}
public class Main {
  public static void main(String[] args) {
    Collection c = new ArrayList();
    c.add(new Guilhotina()) ;
    c.add(new Lamina());
    Inventario.addInstrumento(”Guilhotina”, c); // 1
    Collection k = Inventario.getInstrumento(
       ”Guilhotina”).getPartes();
  }
}
```

- Alerta de operação não verificada em //1

Detalhes de Generics

```
List <String> l1 = new ArrayList<String>();
List<Integer> l2 = new ArrayList<Integer>();
System.out.println(l1.getClass() == l2.getClass());

Collection cs = new ArrayList<String>();
if (cs instanceof Collection<String>) { ...} // ilegal

Collection<String> cstr = (Collection<String>) cs;
// alerta de “não verificado”

<T> T badCast(T t, Object o) {
    return (T) o; // alerta de “não verificado”

 }
```