Coleções e Aplicações

Rafael Santos

Instituto Nacional de Pesquisas Espaciais
www.lac.inpe.br/∼rafael.santos

Tópicos

Objetivos

- Entender que estruturas de dados podem ser representadas em Java.
- Aplicar estas estruturas a problemas comuns.
- Ver como características de Java 5 como *generics*, *autoboxing*, novas formas do laço `for` podem ser usadas.
- Ver muitos exemplos.

Orientação a Objetos

- Classes encapsulam atributos e métodos para processar estes atributos.
- Criamos instâncias das classes para uso em outras classes e aplicações.
- Instâncias são criadas com a palavra-chave `new`.
- Métodos chamados construtores são executados quando instâncias são criadas.

Herança

- Usamos uma classe já existente (ancestral) para derivar uma nova classe.
- Nova classe *herda* campos e métodos públicos e protegidos da classe ancestral.
- Nova classe (herdeira) pode sobrepor métodos e campos.
- Métodos na classe herdeira podem executar métodos na ancestral através de `super`.

Polimorfismo

- Se a classe `B` herda da classe `A`, ela contém os mesmos métodos de `A` (ou métodos sobrepostos).
- Chamamos de polimorfismo a capacidade de executar métodos que estão presentes em classes que tem relação de herança.
- Muito útil quando temos várias instâncias de classes que herdam de uma ancestral comum.

Interfaces

- Diferente do conceito de interfaces gráficas!
- Tipos de classes onde métodos são declarados mas não implementados.
- Servem como *contratos* para classes que implementarão estas interfaces.
- Classes que implementam as interfaces **devem** implementar os métodos declarados.

Arrays

- Agrupamentos de várias instâncias de classes ou valores nativos em uma única variável.
- Exemplos:
  - `int[] a = new int[50];`
  - `Image[] v = new Image[30];`
- Vantagens:
  - Simples, rápido, acesso direto indexado.
- Desvantagens:
  - Restrito a classes com relação de herança, tamanho imutável, índice somente numérico.

Sets ou conjuntos

- Coleção de objetos que não admite objetos em duplicata.
- Interface `Set`: define contrato.
- Métodos:
  - `add`: adiciona um objeto ao *set*.
  - `remove`: remove um objeto do *set*.
  - `contains`: retorna `true` se o *set* contém o objeto.
- Classe `HashSet`: *set* com boa performance.
- Classe `TreeSet`: *set* que garante que elementos estarão em ordem implícita.

Exemplo de *Sets*

```
public class ExemploSet1
  {
  public static void main(String[] args)
    {
    HashSet<Object> set = new HashSet<Object>();
    set.add(new Integer(123));
    set.add(123);
    set.add("ABC");
    set.add("ABC");
    set.add(new Date());
    set.remove("ABC");
    System.out.println(set);
    // [Wed Sep 14 14:13:53 BRT 2005, 123]
    }
  }
```

Exemplo de *Sets*

```
public class ExemploSet2
  {
  public static void main(String[] args)
    {
    TreeSet<Object> set = new TreeSet<Object>();
    set.add(new Integer(123));
    set.add(123);
    set.add("ABC"); // !!!!
    set.add("ABC");
    set.add(new Date());
    set.remove("ABC");
    System.out.println(set);
    // Exception in thread "main" java.lang.ClassCastException:
    //    java.lang.Integer
    }
  }
```

Exemplo de *Sets*

```
public class ExemploSet3
  {
  public static void main(String[] args)
    {
    HashSet<Object> set = new HashSet<Object>();
    set.add(new Integer(123)); set.add(123);
    set.add("ABC");            set.add("ABC");
    set.add(new Date());
    Iterator i = set.iterator();
    while(i.hasNext())
      {
      Object o = i.next();
      if (o instanceof Integer)
        System.out.println("Achei um Integer:"+o);
      }
    // Achei um Integer:123
    }
  }
```

Exemplo de *Sets*

```
public class ExemploSet4
  {
  public static void main(String[] args)
    {
    HashSet<Integer> set = new HashSet<Integer>();
    set.add(new Integer(123)); set.add(111);
    set.add(new Integer(877)); set.add(123);
    int sum = 0;
    System.out.print("Soma de ");
    for(Integer i:set)
      {
      System.out.print(i+" ");
      sum += i;
      }
    System.out.println("é "+sum);
    // Soma de 111 877 123 é 1111
    }
  }
```

Exemplo de *Sets*

```
public class ExemploSet5
  {
  public static void main(String[] args)
    {
    TreeSet<Integer> set = new TreeSet<Integer>();
    set.add(new Integer(123)); set.add(111);
    set.add(new Integer(877)); set.add(123);
    int sum = 0;
    System.out.print("Soma de ");
    for(Integer i:set)
      {
      System.out.print(i+" ");
      sum += i;
      }
    System.out.println("é "+sum);
    // Soma de 111 123 877 é 1111
    }
  }
```

Operações úteis em *sets*

- `addAll`: adiciona um *set* a outro.
- `retainAll`: retém em um *set* tudo o que estiver em outro: interseção de *sets*.
- `removeAll`: remove de um *set* tudo o que estiver em outro.
- `containsAll`: retorna `true` se o *set* conter todos os elementos de outro.

Exemplo de *Sets*

```
public class ExemploSet6
  {
  public static void main(String[] args)
    {
    HashSet<String> solteiros = new HashSet<String>();
    solteiros.add("Tom");     solteiros.add("Larry");
    HashSet<String> casados = new HashSet<String>();
    casados.add("Nathan");    casados.add("Jeffrey");
    casados.add("Randal");    casados.add("Sriram");
    HashSet<String> tenistas = new HashSet<String>();
    tenistas.add("Tom");      tenistas.add("Jeffrey");
    tenistas.add("Larry");
    HashSet<String> nadadores = new HashSet<String>();
    nadadores.add("Nathan"); nadadores.add("Sriram");
    nadadores.add("Tom");
```

Exemplo de *Sets*

```
// Todos os autores
HashSet<String> todos = new HashSet<String>(casados);
todos.addAll(solteiros);
// [Nathan, Tom, Jeffrey, Larry, Randal, Sriram]

// Nadadores e tenistas
HashSet<String> nadadoresETenistas =
  new HashSet<String>(nadadores);
nadadoresETenistas.retainAll(tenistas);
// [Tom]

// Tenistas e casados
HashSet<String> tenistasCasados = new HashSet<String>(tenistas);
tenistasCasados.retainAll(casados);
System.out.println(tenistasCasados);
// [Jeffrey]
```

Exemplo de *Sets*

```
    // Tenistas ou casados
    HashSet<String> tenistasOuCasados = new HashSet<String>(tenistas);
    tenistasOuCasados.addAll(casados);
    System.out.println(tenistasOuCasados);
    // [Nathan, Tom, Jeffrey, Larry, Randal, Sriram]

    // Casados mas não atletas
    HashSet<String> casadosMasNãoAtletas =
      new HashSet<String>(casados);
    casadosMasNãoAtletas.removeAll(tenistas);
    casadosMasNãoAtletas.removeAll(nadadores); // [Randal]

    // Todo nadador é tenista ?
    System.out.println(tenistas.containsAll(nadadores)); // false
    // Todo solteiro é tenista ?
    System.out.println(tenistas.containsAll(solteiros)); // true
    }
}
```

Listas

- Coleção de objetos em forma de lista, aceita duplicatas.
- Interface `List`: define contrato.
- Métodos:
  - `add`: adiciona um objeto à lista.
  - `remove`: remove um objeto da lista.
  - `get`: recupera um objeto da lista.
  - `contains`: retorna `true` se a lista contém o objeto.
  - `indexOf`: retorna o índice do objeto na lista ou -1.
- Classe `LinkedList`: performance razoável em todas as condições.
- Classe `ArrayList`: boa performance, mas pode cair quando tamanho é redimensionado.
- Classe `Stack`: métodos adicionais para *push* e *pop*.

Exemplo de Listas

```
public class ExemploLista1
  {
  public static void main(String[] args)
    {
    ArrayList<Object> lista = new ArrayList<Object>();
    lista.add(new Integer(123));
    lista.add(123);
    lista.add("ABC");
    lista.add("ABC");
    lista.add(new Date());
    lista.remove("ABC");
    System.out.println(lista);
    // [123, 123, ABC, Wed Sep 14 15:04:18 BRT 2005]
    }
  }
```

Exemplo de Listas

```
public class ExemploLista2
  {
  public static void main(String[] args)
    {
    LinkedList<Float> lista = new LinkedList<Float>();
    lista.add(new Float(1.4));  lista.add(1f);
    lista.add(new Float(2.61));
    float sum = 0;
    System.out.print("Soma de ");
    for(Float f:lista)
      {
      System.out.print(f+" ");
      sum += f;
      }
    System.out.println("é "+sum);
    // Soma de 1.4 1.0 2.61 é 5.01
    }
  }
```

Exemplo de Listas

```
public class Sorteio
  {
  public static void main(String[] args)
    {
    ArrayList<Integer> números = new ArrayList<Integer>(60);
    for(int i=1;i<60;i++) números.add(i);
    Collections.shuffle(números);
    for(int i=0;i<6;i++) System.out.print(números.get(i)+" ");
    // 56 11 48 46 4 21
    }
  }
```

Mapas ou *arrays* associativos

- Coleção de objetos como *arrays*, mas índices são objetos.
- Outra interpretação: conjunto de pares (chave,valor) de objetos. Chaves não podem ser duplicadas.
- Interface `Map`: define contrato.
- Métodos:
  - `put`: adiciona um objeto ao mapa.
  - `remove`: remove um objeto do mapa.
  - `get`: recupera um objeto do mapa.
  - `keySet`: retorna um *set* com todas as chaves.
  - `values`: retorna uma coleção com todos os valores.
- Classe `HashMap`: boa performance.
- Classe `TreeMap`: elementos ordenados por chave.

Exemplo de Mapas

```
public class ExemploMap1
  {
  public static void main(String[] args)
    {
    HashMap<Object,Object> mapa = new HashMap<Object,Object>();
    mapa.put(1,"um");
    mapa.put(2,"dois");
    mapa.put(3,"quatro");
    mapa.put(3,"três");
    mapa.remove("dois"); // ?
    mapa.remove(2); // ok
    mapa.put(0.0,"zero");
    mapa.put(0,"zero");
    System.out.println(mapa);
    // {1=um, 3=três, 0=zero, 0.0=zero}
    }
  }
```

Exemplo de Mapas

```
public class ExemploMap2
  {
  public static void main(String[] args)
    {
    TreeMap<String,Integer> mapa = new TreeMap<String,Integer>();
    mapa.put("um",1);
    mapa.put("dois",2);
    mapa.put("três",3);
    mapa.put("quatro",4);
    mapa.put("cinco",5);
    System.out.println(mapa);
    // {cinco=5, dois=2, quatro=4, três=3, um=1}
    System.out.println(mapa.get("quatro")+mapa.get("dois")); // 6
    }
  }
```

Arquivos duplicados em dois diretórios

```
public class ListaDeArquivos
  {
  private TreeSet<String> lista;
  public ListaDeArquivos(String dirName)
    {
    File dir = new File(dirName);
    String[] files = dir.list();
    lista = new TreeSet<String>();
    for(String s:files) lista.add(s);
    }
  public String toString()
    {
    String r = "[";
    for(String s:lista)
      r += s+" ";
    return r+"]";
    }
```

Arquivos duplicados em dois diretórios

```
public TreeSet<String> únicos(ListaDeArquivos outra)
  {
  TreeSet<String> únicos = new TreeSet<String>(lista);
  únicos.removeAll(outra.lista);
  return únicos;
  }

public TreeSet<String> duplicados(ListaDeArquivos outra)
  {
  TreeSet<String> duplicados = new TreeSet<String>(lista);
  duplicados.retainAll(outra.lista);
  return duplicados;
  }
```

Arquivos duplicados em dois diretórios

```
public TreeSet<String> nãoSincronizados(ListaDeArquivos outra)
  {
  TreeSet<String> nãoSincronizados1 = new TreeSet<String>(lista);
  nãoSincronizados1.removeAll(outra.lista);
  TreeSet<String> nãoSincronizados2 = new TreeSet<String>(outra.lista);
  nãoSincronizados2.removeAll(lista);
  nãoSincronizados1.addAll(nãoSincronizados2);
  return nãoSincronizados1;
  }
}
```

Arquivos duplicados em dois diretórios

```
public class DemoListaDeArquivos
  {
  public static void main(String[] args)
    {
    ListaDeArquivos l1 = new ListaDeArquivos("/tmp/1");
    ListaDeArquivos l2 = new ListaDeArquivos("/tmp/2");
    System.out.println(l1); // [SJC2.JPG inscritos.html java.sh]
    System.out.println(l2); // [inscritos.html java.csh java.sh]
    System.out.println(l1.únicos(l2)); // [SJC2.JPG]
    System.out.println(l2.únicos(l1)); // [java.csh]
    System.out.println(l1.duplicados(l2));
    // [inscritos.html, java.sh]
    System.out.println(l1.nãoSincronizados(l2));
    // [SJC2.JPG, java.csh]
    }
  }
```

Mega-Sena

- 60 dezenas, 6 são sorteadas.
- Dois sorteios semanais.
- Prêmio máximo freqüentemente acumula.
- Prêmios para acertos de 6, 5 e 4 dezenas: sena, quina e quadra.
- É fácil ganhar?

Probabilidades

- Sena: probabilidade de acerto jogando seis dezenas é

$$\frac{C_{6-6}^{60-6} \times C_6^6}{C_6^{60}} = \frac{1}{50.063.860}$$

- Quina: probabilidade de acerto jogando seis dezenas é

$$\frac{C_{6-5}^{60-6} \times C_5^6}{C_6^{60}} = \frac{1}{154.518}$$

- Quadra: probabilidade de acerto jogando seis dezenas é

$$\frac{C_{6-4}^{60-6} \times C_4^6}{C_6^{60}} = \frac{1}{2.332}$$

Probabilidades

- Mas jogando 15 dezenas em vez de seis:
- Sena: probabilidade de acerto de ao menos uma é

$$\frac{C_{6-6}^{60-15} \times C_6^{15}}{C_6^{60}} = \frac{1}{10.003}$$

- Quina: probabilidade de acerto de ao menos uma é

$$\frac{C_{6-5}^{60-15} \times C_5^{15}}{C_6^{60}} = \frac{1}{370}$$

- Quadra: probabilidade de acerto de ao menos uma é

$$\frac{C_{6-4}^{60-15} \times C_4^{15}}{C_6^{60}} = \frac{1}{37}$$

O problema é o custo: 7.507,50 reais!

A idéia: ganhar quadras

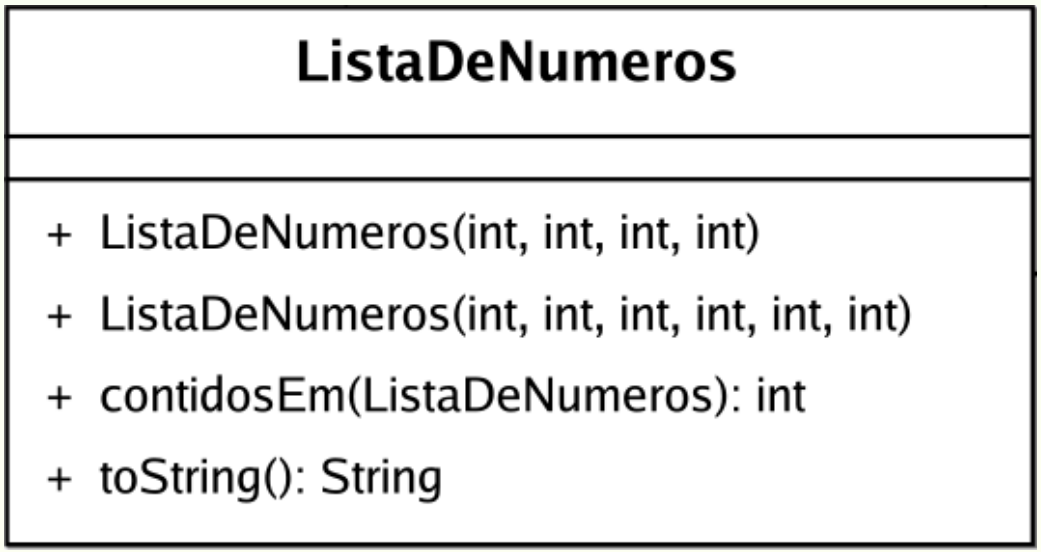
- Se apostarmos 15 dezenas *em um cartão* e acertarmos seis dezenas, ganharemos:
  - 1 sena
  - 54 quinas
  - 540 quadras
- É possível apostar 15 dezenas em mais de um cartão (com preço menor) e acertar várias quadras?
- Quinze dezenas em um cartão é o mesmo preço de 5.005 cartões!
- Criar combinações das 540 quadras *em menos que* 5.005 cartões!

Algoritmo Básico

1. Criar todas as combinações de quadras com 15 dezenas.
2. Criar todas as combinações de cartões com 15 dezenas.
3. Selecionar um dos cartões e com ele:
   - Apagar todas as quadras compostas dos números deste cartão.
   - Guardar o cartão para apostas.
   - Remover o cartão da lista de cartões.
4. Repetir passo 3 enquanto houver combinações de quadras.

Classe `ListaDeNumeros` (1/3)

- Herda de `ArrayList<Integer>`.
- Representa uma lista com 4 ou 6 números.
- Método para ver quantos números de uma lista estão contidos em outra.



Classe ListaDeNumeros (2/3)

```
public class ListaDeNumeros extends ArrayList<Integer>
  {
  public ListaDeNumeros(int n1,int n2,int n3,int n4)
    {
    super();
    add(n1); add(n2); add(n3); add(n4);
    }

  public ListaDeNumeros(int n1,int n2,int n3,
                        int n4,int n5,int n6)
    {
    super();
    add(n1); add(n2); add(n3); add(n4); add(n5); add(n6);
    }
```

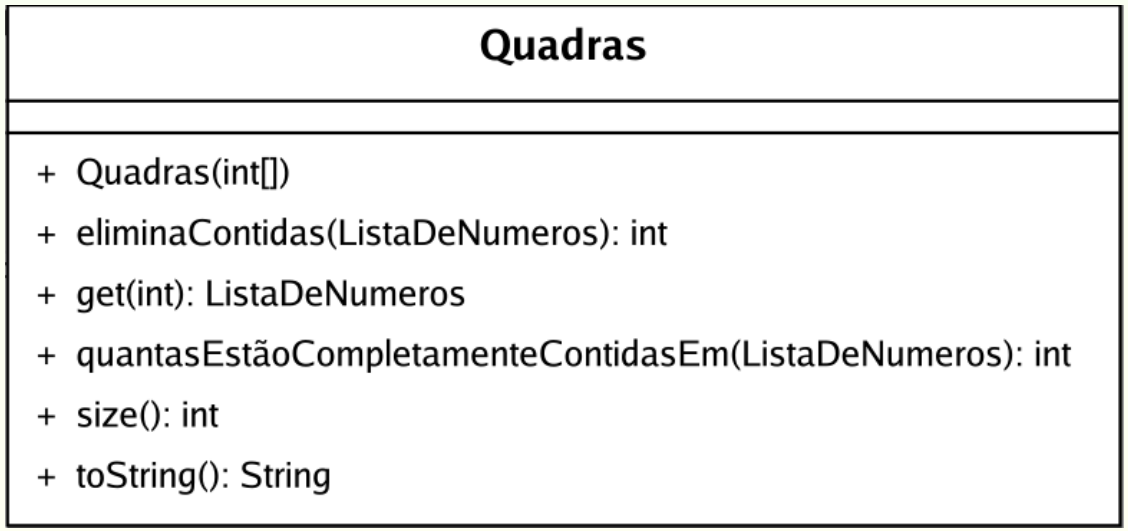
Classe ListaDeNumeros (3/3)

```
public int contidosEm(ListaDeNumeros outra)
  {
  ArrayList<Integer> temporária =
     new ArrayList<Integer>(this);
  temporária.removeAll(outra);
  return this.size()-temporária.size();
  }

public String toString()
  {
  String res = "[";
  for(Integer i:this)
    res += String.format("%2d ",i);
  res += "]";
  return res;
  }
```

Classe Quadras (1/5)

- Quadras: lista de `ListaDeNumeros` com quatro valores.
- Construtor para criar quadras a partir da combinação de vários números.
- Método para eliminar quadras contidas em uma lista.
- Método para contar quantas quadras estão completamente contidas em uma lista de números.



Classe Quadras (2/5)

```
public class Quadras
  {
  private ArrayList<ListaDeNumeros> quadras;

  public Quadras(int[] números)
    {
    quadras = new ArrayList<ListaDeNumeros>();
    for(int d1=0;d1<números.length-3;d1++)
      for(int d2=d1+1;d2<números.length-2;d2++)
        for(int d3=d2+1;d3<números.length-1;d3++)
          for(int d4=d3+1;d4<números.length;d4++)
           quadras.add(
            new ListaDeNumeros(números[d1],números[d2],
                               números[d3],números[d4]));
    }
```

Classe Quadras (3/5)

```
public ListaDeNumeros get(int índice)
  {
  return quadras.get(índice);
  }
public int size()
  {
  return quadras.size();
  }
public String toString()
  {
  StringBuffer res = new StringBuffer();
  for(ListaDeNumeros q:quadras)
    res.append(q.toString()+"\n");
  return res.toString();
  }
```

Classe Quadras (4/5)

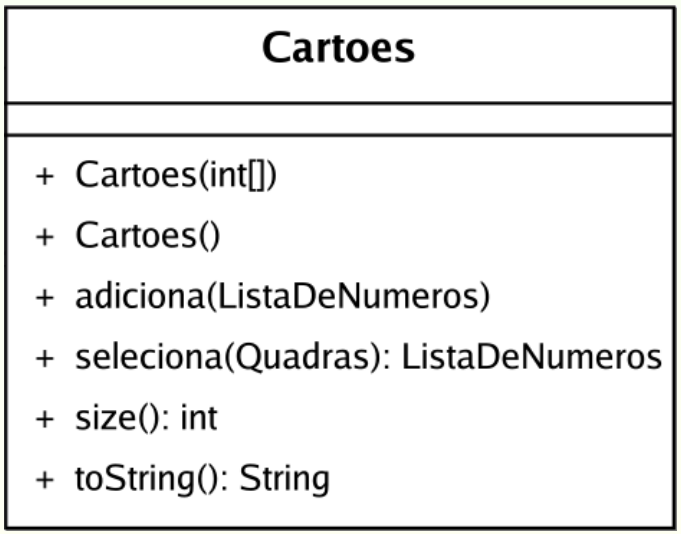
```
public int eliminaContidas(ListaDeNumeros lista)
  {
  int contador = 0; int qualQuadra = 0;
  while(true)
    {
    if (quadras.get(qualQuadra).contidosEm(lista) == 4)
      {
      quadras.remove(qualQuadra);
      contador++;
      }
    else qualQuadra++;
    if (qualQuadra >= size()) break;
    }
  return contador;
  }
```

Classe Quadras (5/5)

```
public int quantasEstãoCompletamenteContidasEm(
                ListaDeNumeros lista)
  {
  int contador = 0;
  for(int q=0;q<quadras.size();q++)
    {
    int interseção = quadras.get(q).contidosEm(lista);
    if (interseção == 4) contador++;
    }
  return contador;
  }
```

Classe Cartoes (1/4)

- Lista de `ListaDeNumeros` com seis valores.
- Construtor para criar a lista de `ListaDeNumeros` a partir da combinação de vários números.
- Método para selecionar um cartão de acordo com máxima eliminação de `Quadras`.



Classe Cartoes (2/4)

```
public class Cartoes
  {
  private ArrayList<ListaDeNumeros> cartões;
  public Cartoes(int[] números)
    {
    cartões = new ArrayList<ListaDeNumeros>();
    for(int d1=0;d1<números.length-5;d1++)
     for(int d2=d1+1;d2<números.length-4;d2++)
      for(int d3=d2+1;d3<números.length-3;d3++)
       for(int d4=d3+1;d4<números.length-2;d4++)
        for(int d5=d4+1;d5<números.length-1;d5++)
         for(int d6=d5+1;d6<números.length;d6++)
          cartões.add(
            new ListaDeNumeros(números[d1],números[d2],
                               números[d3],números[d4],
                               números[d5],números[d6]));
    }
```

Classe Cartoes (3/4)

```
public Cartoes()
  {
  cartões = new ArrayList<ListaDeNumeros>();
  }

public void adiciona(ListaDeNumeros n)
  {
  cartões.add(n);
  }

public int size()
  {
  return cartões.size();
  }
```

Classe Cartoes (4/4)

```
public ListaDeNumeros seleciona(Quadras quadras)
  {
  int índiceMelhor = 0;
  int melhor =
    quadras.quantasEstãoCompletamenteContidasEm
      (cartões.get(0));
  for(int c=1;c<cartões.size();c++)
    {
    int eliminadas =
      quadras.quantasEstãoCompletamenteContidasEm
        (cartões.get(c));
    if (eliminadas > melhor)
      { eliminadas = melhor; índiceMelhor = c; }
    }
  ListaDeNumeros selecionada = cartões.get(índiceMelhor);
  cartões.remove(índiceMelhor);
  return selecionada;
  }
```

Classe Aplicacao (1/3)

1. Criar todas as combinações de quadras com 15 dezenas.
2. Criar todas as combinações de cartões com 15 dezenas.

```
public class Aplicacao
  {
  public static void main(String[] args)
    {
    int[] números =
       {1,2,8,15,16,17,18,20,24,28,32,36,40,48};
    Quadras quadras = new Quadras(números);
    Cartoes cartões = new Cartoes(números);
    Cartoes selecionados = new Cartoes();
```

Classe Aplicacao (2/3)

3. Selecionar um dos cartões e com ele:
   - Apagar todas as quadras compostas dos números deste cartão.
   - Guardar o cartão para apostas.
   - Remover o cartão da lista de cartões.
4. Repetir passo 3 enquanto houver combinações de quadras.

```
while(true)
  {
  ListaDeNumeros cartão = cartões.seleciona(quadras);
  int eliminadas = quadras.eliminaContidas(cartão);
  selecionados.adiciona(cartão);
  if (quadras.size() == 0) break;
  }
```

Classe Aplicacao (3/3, completa)

```
public static void main(String[] args)
  {
  int[] números =
    {1,2,8,15,16,17,18,20,24,28,32,36,40,48};
  Quadras quadras = new Quadras(números);
  Cartoes cartões = new Cartoes(números);
  Cartoes selecionados = new Cartoes();
  while(true)
    {
    System.out.println("Temos "+quadras.size()+" quadras "+
                       "e "+cartões.size()+" cartões.");
    ListaDeNumeros cartão = cartões.seleciona(quadras);
    int eliminadas = quadras.eliminaContidas(cartão);
    System.out.println("Selecionei o cartão "+cartão);
    System.out.println("Eliminei "+eliminadas+" quadras.");
    selecionados.adiciona(cartão);
    if (quadras.size() == 0) break;
    }
  System.out.println(selecionados);
  System.out.println(selecionados.size()+" cartões.");
  }
```

Resultado (1/3)

```
Temos 1365 quadras e 5005 cartões.
Selecionei o cartão [ 1  2  4  8 15 16 ]
Eliminei 15 quadras.
Temos 1350 quadras e 5004 cartões.
Selecionei o cartão [24 28 32 36 40 48 ]
Eliminei 15 quadras.
Temos 1335 quadras e 5003 cartões.
Selecionei o cartão [18 20 32 36 40 48 ]
Eliminei 14 quadras.
...
Temos 906 quadras e 4965 cartões.
Selecionei o cartão [ 4 15 18 28 36 40 ]
Eliminei 11 quadras.
```

Resultado (2/3)

```
Temos 633 quadras e 4940 cartões.
Selecionei o cartão [ 4  8 15 16 40 48 ]
Eliminei 9 quadras.
...
Temos 229 quadras e 4892 cartões.
Selecionei o cartão [ 2  8 15 17 24 40 ]
Eliminei 7 quadras.
...
Temos 159 quadras e 4881 cartões.
Selecionei o cartão [ 4  8 16 17 20 40 ]
Eliminei 5 quadras.
...
Temos 48 quadras e 4855 cartões.
Selecionei o cartão [ 2  8 15 20 32 40 ]
Eliminei 3 quadras.
```

Resultado (3/3)

```
Temos 18 quadras e 4845 cartões.
Selecionei o cartão [15 16 17 20 28 48 ]
Eliminei 2 quadras.
...
Temos 1 quadras e 4837 cartões.
Selecionei o cartão [ 4 16 32 36 40 48 ]
Eliminei 1 quadras.
...
[ 1  2  4  8 15 16 ]
[24 28 32 36 40 48 ]
...
[ 1  2  4 17 18 48 ]
[ 4 16 32 36 40 48 ]
169 cartões.
Resultado Final: quadra garantida (15 dezenas) em 169 cartões.
```

Coleções

- Estruturas bastante flexíveis!
- Complexidade de implementação mas facilidade de uso.
- Java 5 com *generics*, *autoboxing* e novas formas do laço `for` tornam uso mais simples.

Para saber mais...

- Perguntas?
- Esta apresentação e o código-fonte comentado está em www.lac.inpe.br/∼rafael.santos.
- Capítulo sobre Coleções em *Introdução à Programação Orientada a Objetos Usando Java*.
- Participe do SJCJUG (www.sjcjug.org)!