Orientação a Objetos – Coleções de objetos

Programação 2 – Aulas 11

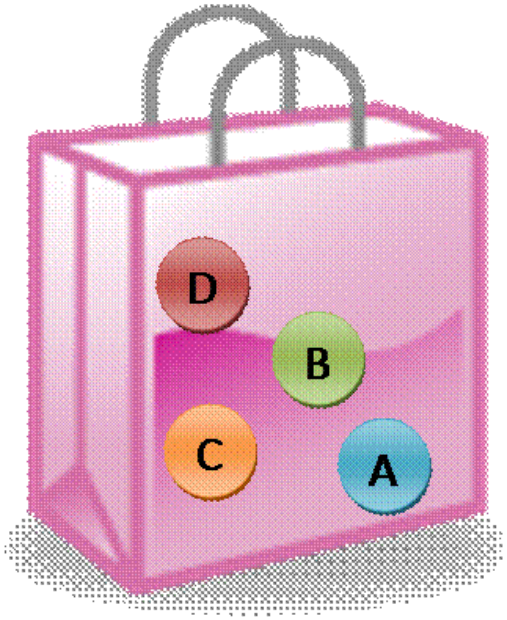
Objetivos da seção

- Apresentar o conceito de uma coleção de objetos
- Caracterizar o comportamento de uma coleção
- Ver como iterar numa coleção usando índices
- Entender a ausência de casts (devido ao uso de Generics)
- Ver como iterar numa coleção usando um iterador
- Pesquisa em coleções

Coleções de Objetos: Iteração Usando Índices

- O que é uma coleção: apresentando o saco de objetos

Uma coleção



— Nós somos clientes (usuários) da coleção: o que queremos fazer com ela?

— Nem sempre queremos fazer as mesmas coisas, mas as possibilidades úteis são:

- Adicionar um objeto dentro da coleção
- Remover um objeto da coleção
- Pesquisar (achar a referência a) um objeto particular da coleção, dada uma chave
- Iterar (ou varrer) os objetos da coleção

  i) Isso significa fazer um *loop* tratando de cada objeto da coleção, um de cada vez

— Nem toda coleção permite todas as operações acima

— O programa Cartas5.java mostra que Baralho é uma coleção

- Contém outros objetos (Cartas)
- Porém, é uma coleção peculiar, porque:

  i) Já vem cheia de objetos

  ii) Não tem método para adicionar Carta

  iii) O método pegaCarta() permite fazer uma iteração dos objetos, conjuntamente com a remoção dos objetos

— Normalmente, queremos coleções com comportamento mais completo

O Array como Coleção

— O saco de objetos não poderia ser um array?

— Sim, como mostra o programa Cadastro1.java a seguir

```
package p2.exemplos;

/*
 * Uso de arrays
 */

import java.util.Arrays;
import p1.io.Entrada;

public class Cadastro1 {
   private final int MAX_PESSOAS = 10;
   private final String prompt = "Digite o nome de uma pessoa: ";
   private String[] cadastro;
   private int numPessoas = 0;

   public Cadastro1() {
      cadastro = new String[MAX_PESSOAS];
   }

   public void cadastraPessoas() {
      String nome;
      while ((nome = Entrada.in.lerLinha(prompt)) != null) {
        cadastro[numPessoas++] = nome;
      }
   }

   public void cadastraPessoa() {
```

```
    String nome;
    if ((nome = Entrada.in.lerLinha(prompt)) != null
          && numPessoas < MAX_PESSOAS) {
      cadastro[numPessoas++] = nome;
    }
  }

  public void imprimeCadastro() {
    System.out.println();
    for (int i = 0; i < numPessoas; i++) {
      System.out.println(cadastro[i]);
    }
  }

  public void ordenaCadastro() {
    String[] tmp = new String[numPessoas];
    for (int i = 0; i < tmp.length; i++) {
      tmp[i] = cadastro[i];
    }
    Arrays.sort(tmp);
    for (int i = 0; i < tmp.length; i++) {
      cadastro[i] = tmp[i];
    }
  }

  public static void main(String[] args) {

    Cadastro1 cadastro = new Cadastro1();
```

```
        // entrada dos dados de cadastro

        cadastro.cadastraPessoas();

        // imprime o cadastro antes da ordenação
        cadastro.imprimeCadastro();

        // ordena o cadastro
        cadastro.ordenaCadastro();

        // imprime o cadastro antes da ordenação
        cadastro.imprimeCadastro();
    } // main
} // Cadastro1
```

— A saída do programa

```
Digite o nome de uma pessoa: Raquel
Digite o nome de uma pessoa: Jacques
Digite o nome de uma pessoa: Fubica
Digite o nome de uma pessoa: Tiago
Digite o nome de uma pessoa: Dalton
Digite o nome de uma pessoa: Joseana
Digite o nome de uma pessoa: ^z
Raquel
Jacques
Fubica
Tiago
Dalton
Joseana
```

```
Dalton
Fubica
Jacques
Joseana
Raquel
Tiago
```

- Observe que ^Z (CTRL+Z) é a forma de indicar fim de arquivo no teclado
- Tente rodar o programa tendo o cadastro pronto num arquivo de texto

```
java -classpath .;package1\p1.jar Cadastro1 < cadastro.txt
```

- Pergunta: Usando um array como coleção, como implementar as 4 operações?
  - Adicionar
  - Remover
  - Pesquisar
  - Iterar
- O que ocorre se digitar mais do que 10 nomes na entrada?
  - Tente!
- Como solucionar?

```
public void cadastraPessoas() {
    String nome;
    while ((nome = Entrada.in.lerLinha(prompt)) != null) {
      if(numPessoas < MAX_PESSOAS)
         cadastro[numPessoas++] = nome;
      else
          System.err.println("Tem dados demais " + "(max de "
          + MAX_PESSOAS + "). " + nome + " nao foi cadastrado.");
```

```
        }
    }
```

- Existem arrays que crescem automaticamente?
  - Java tem várias coleções que fazem isso
- Um exemplo é a coleção ArrayList que se comporta como um array que ajusta seu tamanho
  - Ver o programa Cadastro3.java a seguir

```
package p2.exemplos;

/*
 * Uso de arrays
 */

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import p1.io.Entrada;

public class Cadastro3 {
   private final String prompt = "Digite o nome de uma pessoa: ";
   private List<String> cadastro;

   public Cadastro3() {
      cadastro = new ArrayList<String>();
```

```
}

public void cadastraPessoas() {
    String nome;
    while ((nome = Entrada.in.lerLinha(prompt)) != null) {
      cadastro.add(nome);
    }
}

public void cadastraPessoa() {
    String nome;
    if ((nome = Entrada.in.lerLinha(prompt)) != null) {
      cadastro.add(nome);
    }
}

public void imprimeCadastro() {
    System.out.println();
    for (int i = 0; i < cadastro.size(); i++) {
       System.out.println(cadastro.get(i));
    }
}

public void ordenaCadastro() {
    Collections.sort(cadastro);
}

public static void main(String[] args) {
```

```
        Cadastro3 cadastro = new Cadastro3();
        // entrada dos dados de cadastro

        cadastro.cadastraPessoas();

        // imprime o cadastro antes da ordenação
        cadastro.imprimeCadastro();

        // ordena o cadastro
        cadastro.ordenaCadastro();

        // imprime o cadastro antes da ordenação
        cadastro.imprimeCadastro();
    } // main
} // Cadastro3
```

— Observe a declaração de cadastro

- O tipo genérico é List
- O objeto que criamos é da classe ArrayList
- ArrayList "é uma" List
- Tem outros tipos de objetos que se comportam como List
- Falaremos disso mais na frente quando falarmos de interfaces

— Observações sobre ArrayList

- Qualquer objeto pode ser colocado dentro de um ArrayList
- Aqui, colocamos objetos do tipo String

  i) Usamos *generics* para verificar em tempo de compilação que apenas objetos do tipo

String serão realmente manipulados

ii) Garante que a coleção será de um tipo específico

iii) Por isso usamos <String>

iv) Evita conversões de tipo (transformações explícitas) repetitivas no código

v) Diminui probabilidade de surpresas indesejáveis em tempo de execução

- Em Java, um objeto genérico que pode ser qualquer coisa é um Object
- O número de objetos no ArrayList é size()
- Para acessar o i-ésimo elemento de um ArrayList, use get(i)

Coleções de Objetos: Iteração Usando um Iterador

— Tem outra forma de iterar uma coleção sem usar índices

— Usando um "iterador", podemos dizer: "me dê o próximo, me dê o próximo, me dê o próximo, ..." até acabar

— O primeiro exemplo usa iteradores com ArrayList

— Veja o método abaixo:

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

    }
}
```

— Você pode iterar sobre coleções de qualquer tipo, basta informar isso ao criar o iterador.

Veja o programa a seguir que itera sobre coleção de cartas:

```
package p2.exemplos;
/*
 * Laço: Coleção genérica List
 */

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import p1.aplic.cartas.Baralho;
import p1.aplic.cartas.Carta;
import p1.io.Entrada;

public class Cartas6 {
   public static void main(String[] args) {

      Baralho baralho = new Baralho();
      baralho.baralhar();
      // List é uma "Coleção" de objetos genéricos
      List<Carta> aMao = new ArrayList<Carta>();

      int n = Entrada.in.lerInt("Quantas cartas na mao? ");

      for (int i = 0; i < n; i++) {
         aMao.add(baralho.pegaCarta());
      }

      // iteraNaMao serve para varrer (iterar) as cartas na mao
      Iterator<Carta> iteraNaMao = aMao.iterator();
```

```
        Carta maiorCarta = null;
        while (iteraNaMao.hasNext()) {
            Carta proximaCarta = iteraNaMao.next();
            if (maiorCarta == null || proximaCarta.compareTo(maiorCarta) > 0) {
                maiorCarta = proximaCarta;
            }
        }

        // List tem um toString bonitinho também! :-)
        System.out.println("A mao: " + aMao);
        System.out.println("A maior carta: "
                + (maiorCarta == null ? "nao tem" : maiorCarta));
    }
}
```

— A saída do programa é:

```
Quantas cartas na mao? 4
A mao: [VALETE de ESPADAS, AS de PAUS, QUATRO de COPAS, SETE de COPAS]
A maior carta: VALETE de ESPADAS
```

Coleções de Objetos: Pesquisa

— Uma operação importante que fazemos com coleções é a pesquisa

— Examine a classe Cadastro4

```
package p2.exemplos.aula;

import java.util.ArrayList;
import java.util.List;
```

```java
import p1.aplic.geral.Pessoa;
import p1.io.Entrada;

public class Cadastro4 {

   private List<Pessoa> cadastro;

   public Cadastro4() {
      cadastro = new ArrayList<Pessoa>();
   }

   public boolean cadastraPessoa(String nome, String cpf){
      if(nome != null && cpf != null){
         cadastro.add(new Pessoa(nome, cpf));
         return true;
      }
      return false;
   }

   public String pesquisaPorCPF(String cpf){
      // pesquisa de dados por cpf
      // Seria possivel usar o método indexOf de List, mas
      // queremos mostrar como fazer pesquisa sequencial num array

      for (int i = 0; i < cadastro.size(); i++) {
         Pessoa p = cadastro.get(i);
         if (p.getCPF().equals(cpf)) {
            return p.getNome();
         }
      } // for
      return null;
```

```
    }

    public static void main(String[] args) {
        final String PROMPT1 = "Digite o nome de uma pessoa (\"fim\" para
terminar): ";
        final String PROMPT2 = "Digite o CPF dessa pessoa: ";
        final String PROMPT3 = "Digite o CPF a pesquisar (\"fim\" para
terminar): ";

        Cadastro4 cadastro = new Cadastro4();
        String nome;
        String cpf;
        // entrada dos dados de cadastro
        while ((nome = Entrada.in.lerLinha(PROMPT1)) != null
                && !nome.equals("fim")) {
            cpf = Entrada.in.lerLinha(PROMPT2);
            cadastro.cadastraPessoa(nome, cpf);
        } // while

        //pesquisa por cpf
        while ((cpf = Entrada.in.lerLinha(PROMPT3)) != null
                && !cpf.equals("fim")) {
            if((nome = cadastro.pesquisaPorCPF(cpf)) != null){
                System.out.println("CPF "+ cpf + " pertence a " + nome);
            }else{
                System.out.println("CPF "+ cpf + " nao encontrado");
            }
        }//while

    }
} // Cadastro4
```

— Saída do programa:

```
Digite o nome de uma pessoa ("fim" para terminar): livia
Digite o CPF dessa pessoa: 1234
Digite o nome de uma pessoa ("fim" para terminar): raquel
Digite o CPF dessa pessoa: 4445
Digite o nome de uma pessoa ("fim" para terminar): joao
Digite o CPF dessa pessoa: 1134
Digite o nome de uma pessoa ("fim" para terminar): manoel
Digite o CPF dessa pessoa: 1414
Digite o nome de uma pessoa ("fim" para terminar): fim
Digite o CPF a pesquisar ("fim" para terminar): 1234
CPF 1234 pertence a livia
Digite o CPF a pesquisar ("fim" para terminar): 2345
CPF 2345 nao encontrado
Digite o CPF a pesquisar ("fim" para terminar): 4445
CPF 4445 pertence a raquel
Digite o CPF a pesquisar ("fim" para terminar): fim
```

— Veja como incluir um " dentro de um string: usando \"

— Que coleção estamos usando? Para quê?

— Objetos de qual classe são colocados na coleção?

— Há coleções melhores do que List para fazer pesquisa

◦ Veremos isso em aulas posteriores

Os Comandos break e continue

— Vamos agora ver um exemplo dos comandos “break” para sair de um loop e “continue” para passar para a próxima iteração do loop

— Também estamos vendo um exemplo de loops aninhados

```java
package p2.exemplos;

// Programa feio e sem utilidade, para demonstrar uso de break e continue
public class BreakEContinue {

   public static void main(String[] args) {
      // imprime os numeros entre 0 e 78 que sao multiplos de 9
      for (int i = 0; i < 100; i++) {
         if (i == 78)
            break; // sai do loop for
         if (i % 9 != 0)
            continue; // volta para o inicio do loop for
         System.out.print(i + " ");
      }

      System.out.println();

      // imprime todos os multiplos menores que 78, de 4, de 5... até 9
      for (int i = 4; i < 10; i++) {
         int j = 0;
         while (j++ < 100) {
            if (j == 78)
               break; // sai do loop while
            if (j % i != 0)
               continue; // volta para o inicio do loop while
            System.out.print(j + " ");
         }
         System.out.println();
      }

   }
```

```
}
```

— A saída do programa é

```
Multiplos de 9
9 18 27 36 45 54 63 72
Multiplos de 4
4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76
Multiplos de 5
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75
Multiplos de 6
6 12 18 24 30 36 42 48 54 60 66 72
Multiplos de 7
7 14 21 28 35 42 49 56 63 70 77
Multiplos de 8
8 16 24 32 40 48 56 64 72
Multiplos de 9
9 18 27 36 45 54 63 72
```

— Considere o seguinte trecho de código sobre o uso da classe Cadastro4:

- Usar o comando switch (e assim exemplificar o break) para introduzir um menu de opções
- Desconsiderar comentários na entrada de nomes de pessoas, isto é, linhas que começam com #

```
   public static void main(String[] args) {
      final String SAIR = "3";
      final int CADASTRAR = 1;
      final int PESQUISAR = 2;

      final String PROMPT1 = "Digite o nome de uma pessoa: ";
      final String PROMPT2 = "Digite o CPF dessa pessoa: ";
      final String PROMPT3 = "Digite o CPF a pesquisar: ";
      final String PROMPT4 = "1 - cadastrar pessoa\n" +
      "2 - pesquisar cpf\n" +
      "3 - sair\n" +
      "Qual sua opcao de uso do cadastro: ";

      Cadastro4 cadastro = new Cadastro4();
      String nome;
      String cpf;
      String opcao;

      while(!(opcao = Entrada.in.lerLinha(PROMPT4)).equals(SAIR)){
         int op = Integer.parseInt(opcao);
         switch(op){
         case CADASTRAR:
            if((nome = Entrada.in.lerLinha(PROMPT1)) == null ||
nome.startsWith("#")) continue;
            cpf = Entrada.in.lerLinha(PROMPT2);
            cadastro.cadastraPessoa(nome, cpf);
            break;
         case PESQUISAR:
            cpf = Entrada.in.lerLinha(PROMPT3);
            if((nome = cadastro.pesquisaPorCPF(cpf)) != null){
               System.out.println("CPF "+ cpf + " pertence a " + nome);
```

```
            }else{
                System.out.println("CPF "+ cpf + " nao encontrado");
            }
            break;
        default:
            System.out.println("Opcao invalida!");
        }
    }
}
```

Saída do programa:

```
1 - cadastrar pessoa
2 - pesquisar cpf
3 - sair
Qual sua opcao de uso do cadastro: 1
Digite o nome de uma pessoa: livia
Digite o CPF dessa pessoa: 1234
1 - cadastrar pessoa
2 - pesquisar cpf
3 - sair
Qual sua opcao de uso do cadastro: 8
Opcao invalida!
1 - cadastrar pessoa
2 - pesquisar cpf
3 - sair
Qual sua opcao de uso do cadastro: 1
Digite o nome de uma pessoa: raquel
Digite o CPF dessa pessoa: 4445
1 - cadastrar pessoa
2 - pesquisar cpf
3 - sair
Qual sua opcao de uso do cadastro: 2
Digite o CPF a pesquisar: 5678
```

```
CPF 5678 nao encontrado
1 - cadastrar pessoa
2 - pesquisar cpf
3 - sair
Qual sua opcao de uso do cadastro: 2
Digite o CPF a pesquisar: 1234
CPF 1234 pertence a livia
1 - cadastrar pessoa
2 - pesquisar cpf
3 - sair

Qual sua opcao de uso do cadastro: 3
```

Programa – *HP da disciplina*