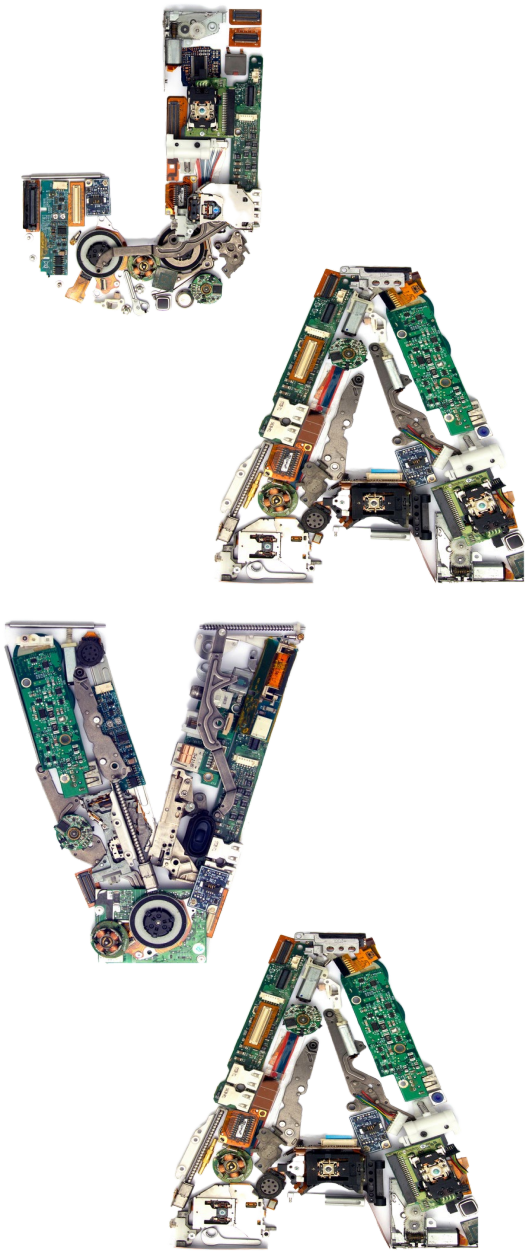
Programação Orientada a Objetos com Java e WEB



ArrayList

1. Coleções

- A API do Java fornece várias estruturas de dados predefinidas, chamadas ***coleções (collection)***, utilizadas para armazenar grupos de objetos relacionados.

Coleção → objeto que agrupa vários elementos em uma unidade (objeto)

- Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem que seja necessário conhecer como os dados são armazenados.

- Isso reduz o tempo de desenvolvimento de aplicativos.

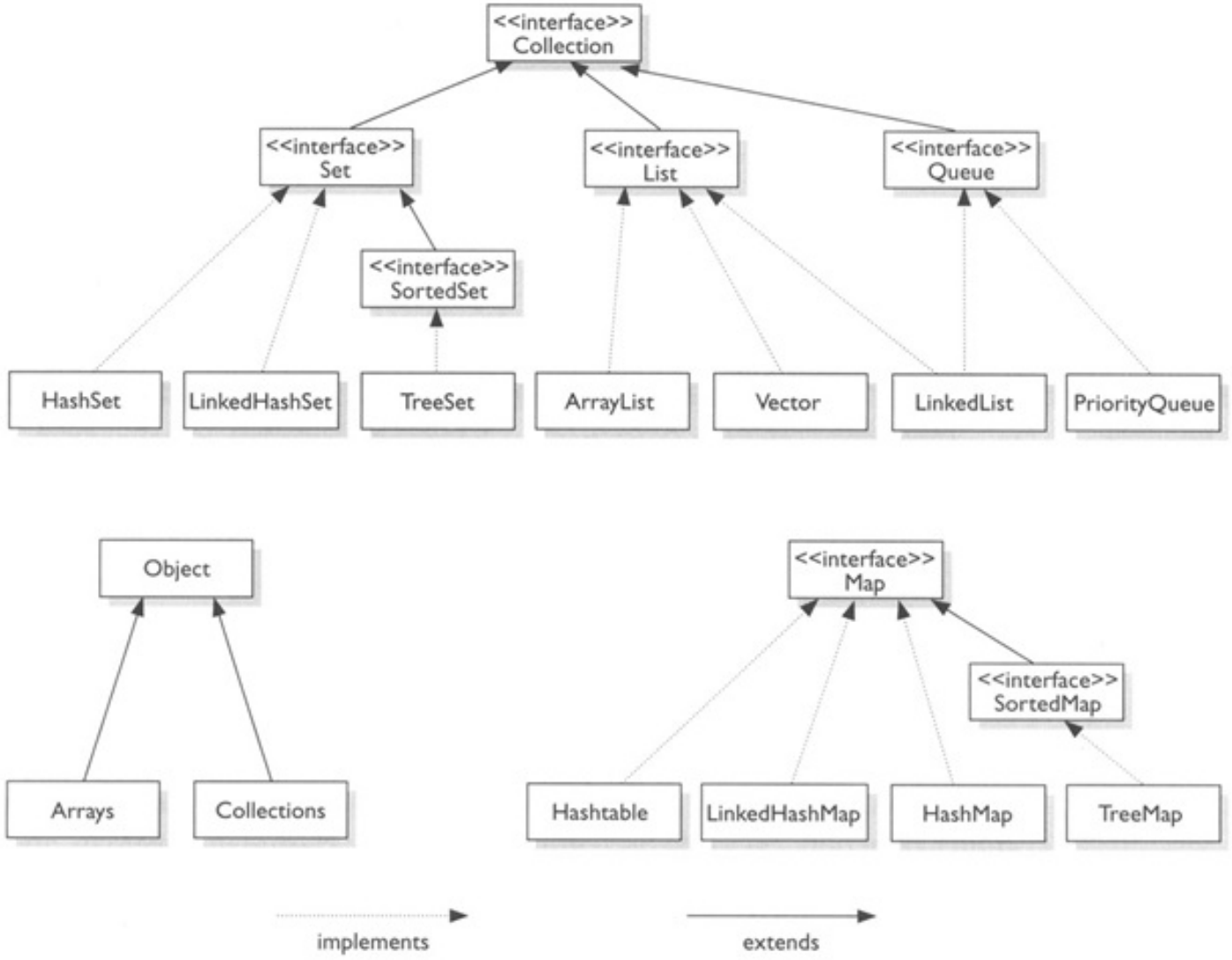
- Os arrays utilizados até agora não alteram automaticamente seu tamanho em tempo de execução para acomodar elementos adicionais.

2. Framework *Collection*

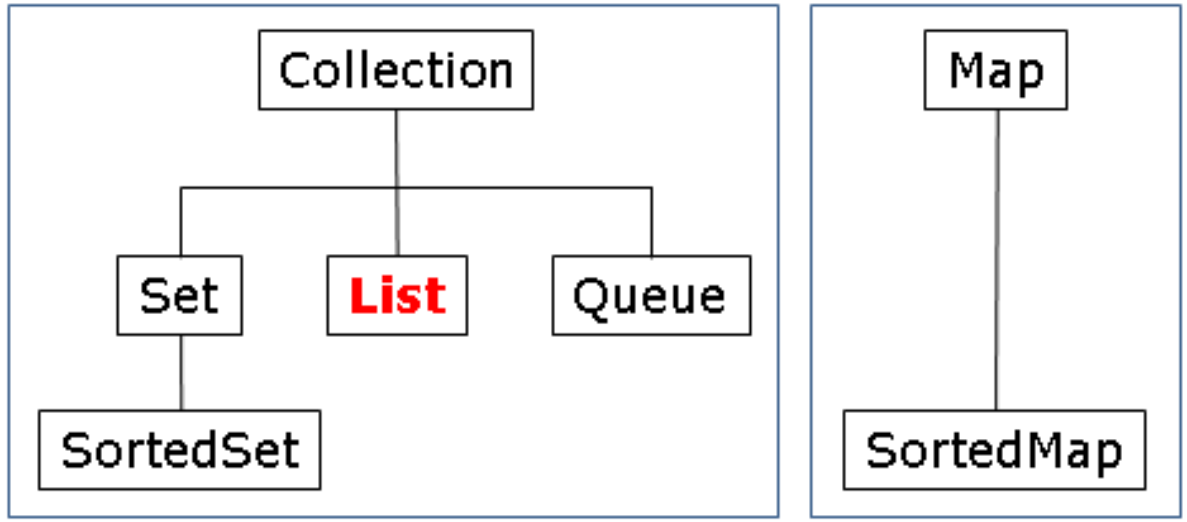
- O que é um framework?
  - Um framework captura a funcionalidade comum a várias aplicações.
  - As aplicações devem ter algo razoavelmente grande em comum: pertencem a um mesmo domínio de problema.

- O que é o framework *Collections* de Java?
  - Uma arquitetura unificada para representar e manipular coleções.
  - Interfaces, implementações e algoritmos.

2. Framework *Collection*



2. Framework *Collection* (resumido)

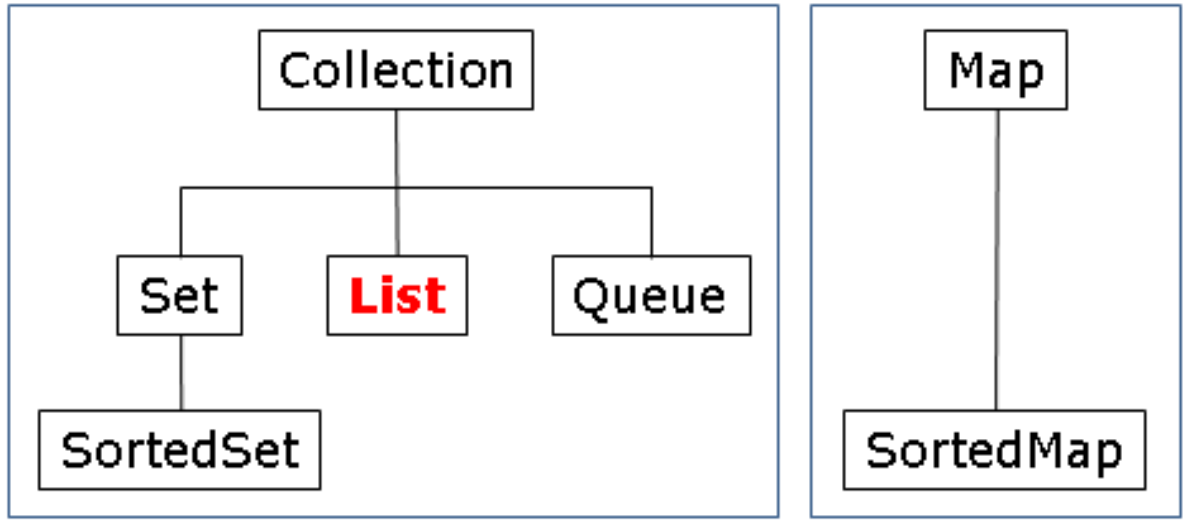


1. **Conjunto (*Set* e *SortedSet*):**
Uma coleção de elementos que modela a abstração matemática para conjuntos. Não mantém indexação e nem contagem dos elementos pertencentes. Cada elemento pertence ou não pertence ao conjunto (**não há elementos repetidos**). Podem ser mantidos ordenados (***SortedSet***) ou não.

2. **Lista (*List*):**
Uma coleção indexada de objetos às vezes chamada de sequência. Como nos vetores, índices de ***List*** são baseados em zero, isto é, o índice do primeiro elemento é zero. Além dos métodos herdados de ***Collection***, ***List*** fornece métodos para manipular elementos baseado na sua posição (ou índice) numérica na lista, remover determinado elemento, procurar as ocorrências de um dado elemento e percorrer sequencialmente (***ListIterator***) todos os elementos da lista. A interface ***List*** é implementada por várias classe, incluídas as classes ***ArrayList*** (implementada como vetor), ***LinkedList*** e ***Vector***.

Fonte: http://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/24298

2. Framework *Collection* (resumido)



3. Fila (*Queue*):
Uma coleção utilizada para manter uma "fila" de elementos. Existe uma ordem linear para as filas que é a "ordem de chegada". As filas devem ser utilizadas quando os itens deverão ser processados de acordo com a ordem "PRIMEIRO-QUE-CHEGA, PRIMEIRO-ATENDIDO". Por esta razão as filas são chamadas de Listas FIFO, termo formado a partir de "*First-In, First-Out*".

4. Mapa (*Map* e *SortedMap*):
Uma mapa armazena pares, chave e valor, chamados de itens. As chaves não podem ser duplicadas e são utilizadas para localizar um dado elementos associado. As chaves podem ser mantidas ordenadas (***SortedMap***) ou não.

Fonte: http://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/24298

3. Interface *List*

- É uma coleção “ordenada”.
  - Semelhante a uma implementação de um array.
  - Algumas vezes chamada sequência.

- Pode conter elementos duplicados.

- Implementações: ***ArrayList***, ***LinkedList.***
  - ***ArrayList***: ideal para pesquisa randômica.
  - ***LinkedList***: ideal para pesquisa sequencial.

4. Classe *ArrayList*

- A classe de coleção ***ArrayList<T>*** (**pacote java.util**) fornece uma solução conveniente ao problema dos arrays estáticos, pois ela pode alterar dinamicamente seu tamanho para acomodar mais elementos.

- O T é um espaço reservado, que deverá ser substituído pelo tipo do elemento que será armazenado no ***ArrayList.***

- Exemplos de declaração:

```
List<String> lista = new ArrayList<>();
Collections<String> lista = new ArrayList();
ArrayList<String> lista = new ArrayList<>();
ArrayList<Aluno> lista = new ArrayList<>();
```

4. Classe *ArrayList*

Método	Descrição
add(objeto)	Adiciona um elemento ao fim de ArrayList.
clear()	Remove todos os elementos de ArrayList.
contains(objeto)	Retorna true se ArrayList tiver o elemento especificado; do contrário, retorna false.
get(índice)	Retorna o elemento no índice(inteiro não negativo) especificado.
indexOf(objeto)	Retorna o índice da primeira ocorrência do elemento especificado em ArrayList. O índice inicia em zero!!
remove(índice)	Remove a primeira ocorrência do valor especificado.
remove(objeto)	Remove o elemento especificado.
size()	Retorna o número de elementos armazenados no ArrayList.
trimToSize()	Reduz a capacidade de ArrayList de acordo com o número de elementos atual.

4. Classe *ArrayList* – exemplo

```
public class Aluno {
     private int rm;
     private String nome;

    public Aluno(int rm, String nome) {
        this.rm = rm;
        this.nome = nome;
    }

    public String toString() {
        return rm+"\n"+nome;
    }
}
```

4. Classe *ArrayList* – exemplo

```
import java.util.ArrayList;

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

        List<Aluno> lista = new ArrayList<>();
        lista.add(new Aluno(5252, "Maria"));
        lista.add(new Aluno(2323, "Pedro"));
        Aluno a;
        for(int k = 0; k < lista.size(); k++) {
            a = lista.get(k);
            System.out.println(a);
        }
    }
}
```

5. Impressão de um *ArrayList*

Uso do for genérico

```
for(Aluno k : lista) {
    System.out.println(k);
}
```

Uso de iteradores

```
//impressão usando iteradores
Iterator<Aluno> it = lista.iterator();
Aluno a;
while(it.hasNext()) {
    a = it.next();
    System.out.println(a);
}
```

Exercício de programação

Uma empresa de logística deseja criar um sistema para calcular o tempo estimado de entrega de pacotes. Dependendo do tipo de pacote (normal, expresso ou internacional), o tempo de entrega varia. O sistema deve ser flexível e capaz de calcular o tempo de entrega de diferentes tipos de pacotes.

Requisitos:

1. Crie a classe base **Entrega** com os atributos **destino** e **distancia** (em km) e o método **calcularTempoEntrega()**, que calcula o tempo de entrega para pacotes normais. O cálculo base é 1 dia para cada 100 km.
2. Crie a classe **EntregaExpresso** que estende **Entrega** e sobrescreve o método **calcularTempoEntrega()** para calcular o tempo em metade do tempo da entrega normal (0,5 dia para cada 100 km).

.

Exercício de programação

Crie a classe **EntregaInternacional** que estende **Entrega** e sobrescreve o método **calcularTempoEntrega()** para adicionar 5 dias ao cálculo normal (1 dia para cada 100 km + 5 dias adicionais).

Implemente uma classe **SistemaLogistica** com um método **processarEntrega(Entrega entrega)** para calcular o tempo de entrega, independentemente do tipo de entrega.

Crie instâncias de **EntregaExpresso** e **EntregaInternacional** e teste o sistema chamando **processarEntrega()** para calcular o tempo de entrega para pacotes normais, expressos e internacionais.

Exemplo de saída esperada:

Destino: São Paulo - Distância: 300 km - Tempo de entrega (Normal): 3 dias

Destino: Rio de Janeiro - Distância: 300 km - Tempo de entrega (Expresso): 1.5 dias

Destino: Nova York - Distância: 8000 km - Tempo de entrega (Internacional): 85 dias

REFERÊNCIAS

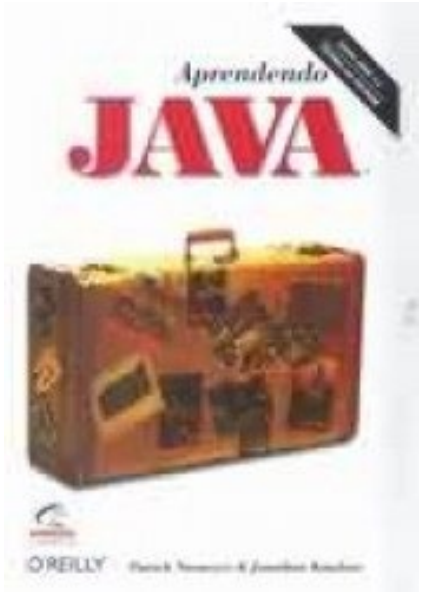
- DEITEL, H. M., DEITEL, P. J. **JAVA como programar**. 8ª edição. São Paulo: Prentice-Hall, 2010.

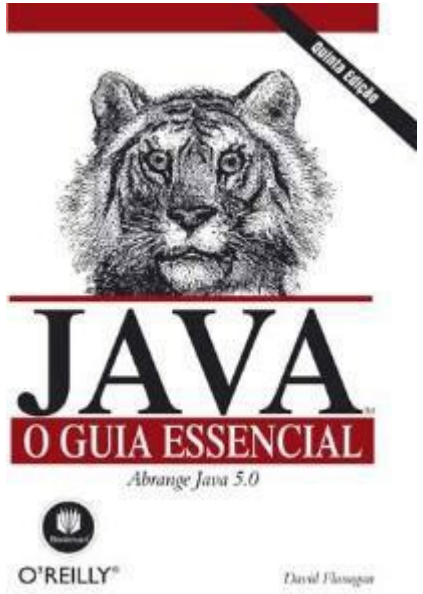
- SCHILDT, H. **Java para Iniciantes – Crie, Compile e Execute Programas Java Rapidamente**. 6ª Edição, Editora Bo o Alegre, RS

REFERÊNCIAS

- KNUDSEN, J., NIEMEYER, P. **Aprendendo Java**. Rio de Janeiro: Editora Elsevier Campus, 2000.

- FLANAGAN, D. **Java – o guia essencial**. Porto Alegre: Editora Bookman, 2006.





REFERÊNCIAS

- ARNOLD, K., GOSLING, J., HOLMES, D., **Java programming language**. 4th Edition, Editora Addison-Wesley, 2005.

- JANDL JUNIOR, P. **Introdução ao Java**. São Paulo: Editora Berkeley, 2002.