Algoritmos e Programação II

Baseado no material do Prof. Julio

Coleções

- Java disponibiliza classes que facilitam o agrupamento e processamento de objetos em conjuntos:
  - Coleções (Java Collections Framework).
  - Estruturas de dados + algoritmos para sua manipulação.
- Java Collections framework
  - Arquitetura unificada para representar e manipular coleções, de forma independente dos detalhes de sua representação.

Coleções

- O programador simplesmente utiliza as estruturas de dados sem se preocupar com a maneira como são implementadas.
- Vantagens:
  - Reutilização de código
  - Desempenho superior
    - Maior velocidade de execução.
    - Menos memória.
    - Algoritmos otimizados.

Coleções

- Coleções:
  - De forma simplificada, são objetos capazes de armazenar conjuntos de referências para outros objetos;
    - Listas, pilhas, conjuntos, etc
  - Correspondem a classes oferecidas na biblioteca padrão de Java;
  - Armazenam referências do tipo Object;
  - Obrigatório o uso de polimorfismo.

Coleções

- Os elementos que compreendem a estrutura de coleções estão no pacote java.util.
- Java separa a representação da coleção em:
  - Interfaces
    - Definem métodos que podem ser usados para manipulação dos objetos nas coleções.
  - Implementações
    - Classes que implementam as interfaces, mas, internamente, manipulam os dados de forma diferente.

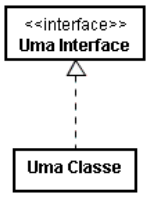
Interface

- Interfaces podem ser utilizadas para separar a especificação do comportamento de um objeto de sua implementação concreta.
  - Trazem a especificação do conjunto de operações públicas
- Dessa forma a interface age como um contrato, o qual define explicitamente quais métodos uma classe deve obrigatoriamente implementar.

Implementações

- Uma interface deve ser implementada por uma classe
  - Uma interface pode ser implementada por diversas classes.
    - POLIMORFISMO
  - Uma classe pode implementar diversas interfaces.

Implementações



Estudo de Caso 1 - Listas

- Uma lista é uma coleção linear de elementos que podem ser percorridos seqüencialmente e permite inserção e remoção de elementos em qualquer posição.
- Conceitualmente, não possui um tamanho máximo.

Estudo de Caso 1 - Listas

- As operações disponíveis sobre listas estão definidas na interface `List`
- A documentação da API de Java lista todas as operações permitidas sobre uma lista.
  - Dica: acesse a documentação via diretório F:\Program Files\Java\jdkXXX\docs\api\ no laboratório da FACIN

Estudo de Caso 1 - Listas

Estudo de Caso 1 - Listas

- Algumas operações:
  - `add(indice,objeto)` adiciona um objeto na posição do índice
  - `add(objeto)` adiciona um objeto na posição final da lista
  - `get(indice)` retorna o objeto armazenado na posição do índice indicado
  - `remove(indice)` remove e retorna o objeto armazenado na posição do índice indicado

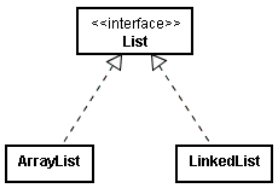
Estudo de Caso 1 - Listas

- Algumas operações:
  - `clear()` limpa a lista
  - `isEmpty()` retorna verdadeiro se a lista está vazia
  - `size()` retorna o número de elementos da lista

Estudo de Caso 1 - Listas

- Duas implementações diferentes da interface `List` são as classes `ArrayList` e `LinkedList`.
  - Implementações com performance diferente para operações diferentes.
- Declaração:
  - Devemos informar o tipo dos elementos da lista ao declararmos uma coleção
  - `List<Tipo> umaLista = new ArrayList<Tipo>();`
  - `List<Tipo> umaLista = new LinkedList<Tipo>();`

Estudo de Caso 1 - Listas

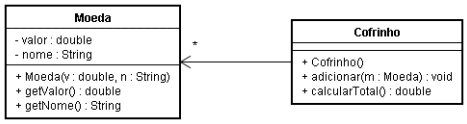


Estudo de Caso 1 - Listas

- Classes `ArrayList` e `LinkedList`
  - Implementações da interface `List`.
  - `ArrayList` é uma implementação sob a forma de vetor de tamanho variável.
  - `LinkedList` é uma implementação da mesma interface sob a forma de lista encadeada.
  - Vantagem:
    - Escolhe-se o tipo de estrutura conforme a necessidade da aplicação, porém, a forma de usá-las é exatamente a mesma.

Exemplo

- Implementar um cofrinho de moedas com a capacidade de receber moedas e calcular o total depositado no cofrinho.



Exemplo

```
public class Cofrinho {
   private List<Moeda> moedas;
   public Cofrinho() {
        moedas = new ArrayList<Moeda>();
   }
}
```

Implementa uma coleção de Moeda como uma lista.

Classe ArrayList foi escolhida como implementação de lista.

Exercícios

1) Altere a classe Cofrinho de modo que ela implemente métodos para:
   - Contar o número de moedas armazenadas
   - Contar o número de moedas de um determinado valor
   - Informar qual a moeda de maior valor

Exercícios

2) Escreva um programa Java que crie 5 objetos Circulo de tamanho diferentes, insira-os em uma lista e depois percorra a lista imprimindo a área de cada círculo armazenado.

Exercícios

3) Implemente uma classe Cadastro que funcione como cadastro de professores (classe Professor desenvolvida em aulas anteriores). Através desta classe deve ser possível adicionar um professor, buscar um professor pelo seu número de matrícula e remover um professor pelo seu número de matrícula. Depois, escreva um programa em Java para testar a classe desenvolvida. Utilize a classe *LinkedList* para implementar a coleção de professores.

Estudo de Caso 2 - Mapas

- Um mapa é um tipo de dado que mantém associações entre *chaves* e *valores*.
  - Não possui valores duplicados de chaves
  - Cada chave somente é associada com um único valor
- Por exemplo: nome de pessoa e sua cor favorita

Estudo de Caso 2 - Mapas

- As operações disponíveis sobre Mapas estão definidas na interface `Map`
- Duas implementações diferentes são as classes `HashMap` e `TreeMap`.
- Declaração:
  - `Map<TipoChave,TipoValor> umMapa = new HashMap<TipoChave,TipoValor>();`

Estudo de Caso 2 - Mapas

- Algumas operações:
  - `put(chave, valor)` associa o valor à chave informada, armazenando na coleção
  - `get(chave)` retorna o valor associado à chave
  - `remove(chave)` remove da coleção a chave e o valor, e retorna o valor associado à chave
  - `containsKey(chave)` retorna verdadeiro se a coleção contém a chave informada
  - `size()` retorna o número de elementos da coleção

Exemplo

- Implementar um coleção do tipo mapa que associa nomes de pessoas a sua cor favorita.

Exercícios

1) Implemente uma classe em Java com as funcionalidades de uma agenda telefônica, associando um nome a um número telefônico. A classe deve possuir a interface abaixo:

Exercícios

2) Crie um programa em Java para testar a classe AgendaTelefonica desenvolvida no exercício anterior. Teste a classe com pelo menos 5 contatos diferentes na agenda de telefones.