

53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

RESIDÊNCIA DE SOFTWARE

**CAPACITAR
TREINAR
EMPREGAR**

TRANSFORMAR



JAVA I

Vetores, Listas, Map e Coleções

07/08/2020

ARRAYS

É um objeto que pode armazenar diversos valores de um determinado tipo.

Declaração

```
String[] varVetor;
```

A declaração não cria o array . Somente diz que esta variável terá um array do tipo especificado.

Criação do array

```
varVetor = new String[5];
```

Aloca um array na memória para 5 elementos do tipo String e atribui o array à variável **varVetor**



A declaração e criação poderia ser feita em uma linha.

```
String [] varVetor = new String[5];
```

EXEMPLOS:

Vetor com 5 posições

```
TesteArray.java X
package aula;

public class TesteArray {
    public static void main(String[] args) {
        String[] varVetor = new String[5];
        varVetor[0] = "Laranja";
        varVetor[1] = "Maça";
        varVetor[2] = "Morango";
        varVetor[3] = "Abacate";
        varVetor[4] = "Goiaba";

        System.out.println("Posição 0: " + varVetor[0]);
        System.out.println("Posição 1: " + varVetor[1]);
        System.out.println("Posição 2: " + varVetor[2]);
        System.out.println("Posição 3: " + varVetor[3]);
        System.out.println("Posição 4: " + varVetor[4]);
    }
}
```

otimizando

```
for (int i = 0; i < 5; i++) {
    System.out.println("Posição" + i + ":" + varVetor[i]);
}
```

```
for (String a: varVetor){
    System.out.println(a);
}
```

Podemos inicializar o vetor na criação e o número de elementos do vetor é determinado pelo número de valores informados no caso abaixo dois elementos

```
TesteArray2.java X
package aula;

public class TesteArray2 {
    public static void main(String[] args) {
        String[] varVetor = new String[] { "Laranja", "Maça",
        "Morango", "Abacate", "Goiaba" };
        System.out.println(varVetor[0]);
        System.out.println(varVetor[1]);
        System.out.println(varVetor[2]);
        System.out.println(varVetor[3]);
        System.out.println(varVetor[4]);
    }
}
```

outra forma de inicialização

```
String[] varVetor = { "Laranja", "Maça",
    "Morango", "Abacate", "Goiaba" };
```

ARRAY MULTIDIMENSIONAL

A linguagem Java não fornece vetores multidimensionais, mas, é possível criar vetores de vetores tendo o mesmo efeito.

```
*TesteArray3.java X
1 package aula;
2
3 public class TesteArray3 {
4     public static void main(String[] args) {
5         String funcionario[][] = new String[][] { { "Marcos", "1343-4352" },
6             { "Pedro", "9084-5909" }, { "Beatriz", "7344-9456" },
7             { "Jorge", "3456-1245" } };
8
9         for (int i = 0; i < funcionario.length; i++) {
10
11             System.out.println("-----");
12
13             for (int e = 0; e < funcionario[i].length; e++) {
14
15                 System.out.println(funcionario[i][e] + " posição:" + i + e);
16
17             }
18         }
19     }
20 }
```

Declarando e inicializando

Para sabermos os tamanhos dos vetores utilizamos o campo length

```
String funcionario[][] = new String[4][2];
funcionario[0][0] = "Marcos";
funcionario[0][1] = "1343-4352";
funcionario[1][0] = "Pedro";
funcionario[1][1] = "9084-5909";
funcionario[2][0] = "Beatriz";
funcionario[2][1] = "7344-9456";
funcionario[3][0] = "Jorge";
funcionario[3][1] = "3456-1245";
```

Declarando e definindo o número de linhas e colunas.

EXERCÍCIOS

1) Criar um novo projeto com o nome **aula9**. Criar um pacote com o nome **aula**.

Criar uma classe com o nome **Funcionario**

- atributos private do **Funcionario** : nome, cargo, e salario

- Métodos da classe **Funcionario**.

- Crie o método **abonoSalario** na classe **Funcionario** .

O abono é acrescido ao salário do funcionário

- Construa dois objetos em um vetor em outra classe com o nome **TestaFuncionario** com quaisquer dados.

- Exiba os dados dos funcionários com o abono salarial.

```
TestaFuncionario.java ✕
package aula;

public class TestaFuncionario {
    public static void main(String[] args) {
        Funcionario func[] = new Funcionario[2];
        func[0] = new Funcionario();
        func[0].setCargo("analista de sistemas");
        func[0].setNome("Adriana");
        func[0].setSalario(5000.);

        func[1] = new Funcionario();
        func[1].setCargo("Advogado");
        func[1].setNome("Leonardo");
        func[1].setSalario(4200.);

        for (int i = 0; i < func.length; i++) {
            System.out.println(func[i].getNome() + "-" + func[i].getCargo());
            System.out.println(func[i].abonoSalario(100));
        }
    }
}
```

```
*Funcionario.java ✕

public class Funcionario {
    private String nome;
    private String cargo;
    private Double salario;

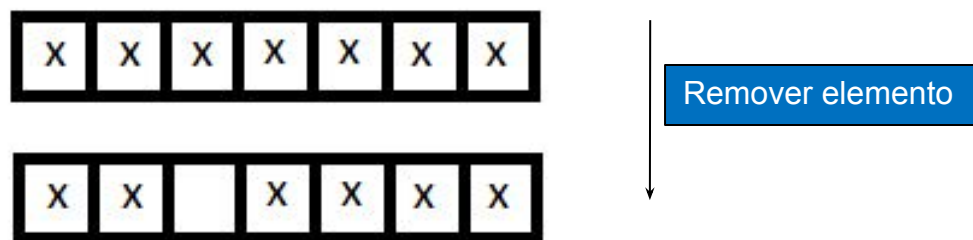
    getters e setters.....

    public double abonoSalario(double abono) {
        return this.salario += abono;
    }
}
```

DESVANTEGENS DE ARRAY – INTRODUÇÃO A COLLECTION

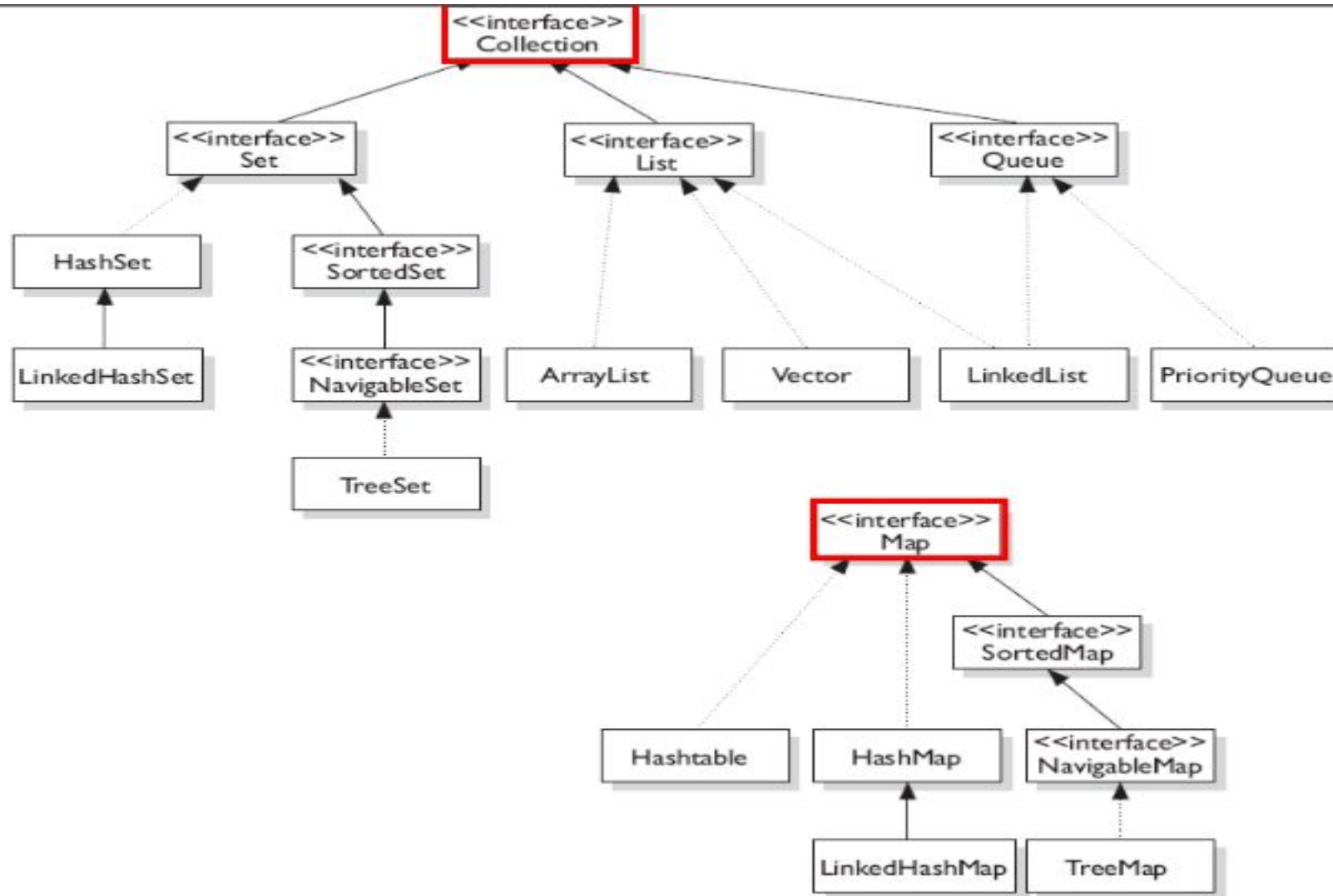
Como vimos no capítulo de arrays, manipulá-las é bastante trabalhoso. Essa dificuldade aparece em diversos momentos:

- não podemos redimensionar um array em Java;
- é impossível buscar diretamente por um determinado elemento cujo índice não se sabe;
- não conseguimos saber quantas posições do array já foram populadas sem criar, para isso, métodos auxiliares.



- Se formos adicionar algo no vetor, vamos ter que percorrer ele em busca de posições vazias
- Como saber quantas posições estão ocupadas no array?
- Vou precisar sempre percorrer o array inteiro para conseguir essa informação?

JAVA COLLECTIONS FRAMEWORK



É um contrato de interfaces que representa uma coleção de objetos. Toda estrutura é baseada em duas interfaces principais:

- **Java.util.Collection**
- **Java.util.Map**

LISTAS

A lista é uma coleção que permite elementos duplicados e mantém uma ordenação específica entre os elementos. A interface `java.util.List`, que especifica o que uma classe deve ser capaz de fazer para ser uma lista. A implementação mais utilizada da interface `List` é a `ArrayList`, que trabalha com um array interno para gerar uma lista. Toda lista trabalha do modo mais genérico possível. Todos os métodos trabalham com `Object`.

*ExemploListas.java

```
package aula;

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

public class ExemploListas {
    public static void main(String[] args) {
        List lista = new ArrayList();
        lista.add("pato");
        lista.add("gambá");
        lista.add("boi");
        lista.add("urso");
        lista.add("cachorro");
        lista.add("gato");
        lista.add("rato");

        System.out.println(lista.size());
        System.out.println(lista.get(2));
        System.out.println(lista);
        lista.remove(0);

        System.out.println(lista);
    }
}
```

Diferente de um vetor não dizemos qual é o tamanho da lista. Podemos acrescentar quantos elementos quisermos que a lista cresce conforme for necessário.

Classe `ArrayList` implementa `List`

Insere um elemento no final da lista

Retorna o tamanho da lista

Retorna o terceiro elemento da lista

Remove o primeiro elemento

Podemos criar uma lista de outra forma utilizando o método estático **asList** da classe **Arrays**

```
public class TesteListas {  
    public static void main(String[] args) {  
        List listaDeAnimais = Arrays.asList("Cachorro", "Papagaio", "Tigre", "Gato", "Coelho");  
  
        if (listaDeAnimais.isEmpty()) {  
            System.out.println("Lista está vazia");  
        } else {  
            Collections.sort(listaDeAnimais);  
            System.out.println("Os animais são:" + listaDeAnimais);  
        }  
  
        boolean pesquisa = listaDeAnimais.contains("Rato");  
        System.out.println("A lista contém rato:" + pesquisa);  
    }  
}
```

O método **contains** serve para pesquisarmos se existe um elemento na lista

EXERCÍCIOS

Criar uma classe com o nome **Pessoa** com os atributos nome, cpf e idade.

Criar uma classe com o nome **Cliente** com o atributo nome.

Criar uma classe com o nome **TestaListas** com 4 objetos da classe **Pessoa** e gere uma lista com estes dados do nome e a idade. Remova da lista o segundo elemento.

```
TestaListas.java X
package aula;

import java.util.ArrayList;

public class TestaListas {
    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("Manoel", "123.899.879-09", 76);
        Pessoa p2 = new Pessoa("Josefina", "423.999.879-09", 26);
        Pessoa p3 = new Pessoa("Antonio", "123.899.879-09", 26);
        Pessoa p4 = new Pessoa("Solimar", "890.980.080-87", 18);

        List lista = new ArrayList();
        lista.add(p1);
        lista.add(p2);
        lista.add(p3);
        lista.add(p4);

        lista.remove(1);

        System.out.println(lista.size());

        for (int i = 0; i < lista.size(); i++) {
            Pessoa p = (Pessoa) lista.get(i);
            System.out.println(p.getNome() + "-" + p.getIdade());
        }
    }
}
```

O método get retorna um Object por isso fazemos o casting

EXERCÍCIOS

Instanciar um objeto da classe Cliente e adicionar na mesma lista que os objetos da classe “Pessoa” foram adicionados.
Imprimir os elementos dessa lista

TestaListas.java ✕

```
package aula;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class TestaListas {
    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("Manoel", "123.899.879-09", 76);
        Pessoa p2 = new Pessoa("Josefina", "423.999.879-09", 26);
        Pessoa p3 = new Pessoa("Antonio", "123.899.879-09", 26);
        Pessoa p4 = new Pessoa("Solimar", "890.980.080-87", 18);
        Cliente c1 = new Cliente("Roni");
        List lista = new ArrayList();
        lista.add(p1);
        lista.add(p2);
        lista.add(p3);
        lista.add(p4);
        lista.add(c1);
        for (int i = 0; i < lista.size(); i++) {
            if (lista.get(i) instanceof Pessoa) {
                Pessoa p = (Pessoa) lista.get(i);
                System.out.println(p.getNome() + "-" + p.getIdade());
            } else {
                Cliente c = (Cliente) lista.get(i);
                System.out.println(c.getNome());
            }
        }
    }
}
```

Imprimindo lista com vários objetos. Criar uma instância de Cliente.

LISTAS E GENERICS

Em uma lista podemos inserir qualquer tipo de objeto, mas dificilmente iremos trabalhar com objetos diferentes em uma lista. Usando o Generics o compilador não permite colocar na lista elementos incompatíveis. Um tipo genérico é *um tipo* que possui um ou mais parâmetros de tipo.

```
*TestaListas.java ✕  
  
package aula;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
public class TestaListas {  
    public static void main(String[] args) {  
        Pessoa p1 = new Pessoa("Manoel", "123.899.879-09", 76);  
        Pessoa p2 = new Pessoa("Josefina", "423.999.879-09", 26);  
        Pessoa p3 = new Pessoa("Antonio", "123.899.879-09", 26);  
        Pessoa p4 = new Pessoa("Solimar", "890.980.080-87", 18);  
  
        List <Pessoa> lista = new ArrayList<Pessoa>();  
        lista.add(p1);  
        lista.add(p2);  
        lista.add(p3);  
        lista.add(p4);  
  
        for (int i = 0; i < lista.size(); i++) {  
            Pessoa p = lista.get(i);  
            System.out.println(p.getNome() + "-" + p.getIdade());  
        }  
    }  
}
```

O Generics restringe a lista a um determinado tipo de objeto.

Não precisamos mais fazer casting
pois os objetos são todos do tipo
Pessoa

ORDENAÇÃO JAVA.LANG.COMPARABLE

Comparable é uma interface que quando as classes a implementam adquirem a propriedade de ordenação que permite que os objetos dessas classes sejam ordenados automaticamente. As classes **String** e **Date** implementam **Comparable** basta apenas usar o comando **Collections.sort(variável)** para ordenar a lista. A interface **Comparable** possui um único método: **int compareTo(Object o)**. Este método deve retornar zero, se o objeto comparado for igual a este objeto, um número negativo, se este objeto for menor que o objeto dado, e um número positivo, se este objeto for maior que o objeto dado. Caso os objetos não possam ser comparados é lançada uma **ClassCastException**.

OrdenaString.java ✕

```
package aula;

import java.util.ArrayList;

public class OrdenaString {
    public static void main(String[] args) {
        List lista = new ArrayList();
        lista.add("Goiaba");
        lista.add("Uva");
        lista.add("Abacate");
        lista.add("Jabuticaba");
        boolean b = lista.contains("Morango");
        System.out.println(b);

        Collections.sort(lista);
        System.out.println(lista);
    }
}
```

A classe String implementa Comparable implementando assim o método compareTo

o método contains verifica se um elemento existe na lista

Método para ordenar a lista que encontra-se em java.util.Collections. Esta linha não compilará senão implementarmos a interface Comparable na classe Pessoa

Alterar o código abaixo na classe Pessoa

```
public class Pessoa implements Comparable<Pessoa> {
```

Implementar o método abaixo na classe Pessoa

```
@Override
public int compareTo(Pessoa outraPessoa) {
    if (this.idade < outraPessoa.idade)
        return -1;
    else if (this.idade > outraPessoa.idade)
        return 1;
    else
        return 0;
    //return this.idade - outraPessoa.idade;
}
```

Se a idade da Pessoa atual é menor do que da outraPessoa retornamos -1 ou qualquer inteiro negativo, se for maior retornamos 1 ou qualquer inteiro positivo e se for igual então retornamos 0.

O método pode funcionar somente com esta linha.

Inserir o código abaixo na classe TestaListas após a linha lista.add(p4);

```
Collections.sort(lista);
```

Os objetos do grupo **Set** não permitem elementos duplicados. A ordem em que os elementos são armazenados pode não ser a ordem na qual eles foram inseridos no conjunto. Conjuntos são extensamente usados pra representação de dados.

Conjuntos em Java são encapsulados por uma instância de uma das classes que implementam Set. Esta por sua vez declara vários métodos que possibilitam a inclusão e remoção de objetos na coleção.

Duas Classes implementam Set diretamente, são elas : **HashSet e TreeSet**

HashSet – Através de um mecanismo interno reduz o elemento a um número, e com base nele, realiza uma pesquisa em um sublista de elementos contendo esse mesmo número, é mais rápido que o TreeSet para as operações de modificação do conjunto e não estabelece nenhuma ordem particular entre os objetos.

TreeSet– Preserva a ordem natural dos elementos, porém ocorre uma perda de performance na inserção e deleção TreeSet. O algoritmo usado é de árvore binária para ordenar os elementos.

EXEMPLO

```
public class TesteSet {  
    public static void main(String[] args) {  
        Set<String> times = new HashSet<String>();  
        times.add("Flamengo");  
        times.add("Serrano");  
        times.add("Fluminense");  
        times.add("Botafogo");  
        times.add("Vasco");  
        times.add("Flamengo");  
        times.add("América");  
  
        for (String valor : times) {  
            System.out.println(times);  
        }  
  
        System.out.println("Total de times da coleção" + times.size());  
  
        Iterator<String> valores = times.iterator();  
        while (valores.hasNext()) {  
            String time = valores.next();  
            System.out.println(time);  
        }  
    }  
}
```

Elemento repetido não será adicionado

O método hasNext() verifica se existe um elemento depois do atual a ser percorrido.

java.util.Iterator

É uma interface do pacote java.util. Serve para percorrer uma coleção seqüencialmente. Permite remover um elemento da coleção

Alterar as linhas em destaque para testarmos o TreeSet

```
ExemploSet.java ✕  
package aula;  
  
import java.util.Set;  
import java.util.TreeSet;  
  
public class ExemploSet {  
    public static void main(String[] args) {  
        Set<String> s = new TreeSet<String>();  
        s.add("Flamengo");  
        s.add("Serrano");  
        s.add("Goytacaz");  
        s.add("Tupi");  
        s.add("Fluminense");  
        s.add("Ponte Preta");  
        s.add("Flamengo");  
  
        System.out.println(s);  
    }  
}
```

Os elementos são ordenados automaticamente

ITERATOR

- Toda coleção fornece acesso a um *iterator*, um objeto que implementa a interface `Iterator`

```
Iterator<String> i = lista.iterator();
```

- Na versão 5 do Java foi lançada *enhanced-for*. E, na verdade, o *enhanced-for* é apenas um açúcar sintático que usa *iterator* por trás dos panos.

```
for(String palavra : conjunto){  
    System.out.println(palavra);  
}
```

MAPAS - JAVA.UTIL.MAP

- Um mapa é composto por um conjunto de associações entre um objeto chave a um objeto valor. É equivalente ao conceito de dicionário, usado em várias linguagens.

```
public class Main {  
    public static void main(String[] args) {  
        HashMap<String, String> mapaEstados = new HashMap<>();  
        mapaEstados.put("AC", "Acre");  
        mapaEstados.put("AL", "Alagoas");  
        mapaEstados.put("AP", "Amapá");  
        mapaEstados.put("AM", "Amazonas");  
        mapaEstados.put("BA", "Bahia");  
        mapaEstados.put("CE", "Ceará");  
        mapaEstados.put("DF", "Distrito Federal");  
        mapaEstados.put("ES", "Espírito Santo");  
        mapaEstados.put("GO", "Goiás");  
  
        for (String sigla : mapaEstados.keySet()) {  
            System.out.print(sigla + " ");  
        }  
    }  
}
```

Suas principais implementações são:

- HashMap
- TreeMap
- Hashtable

- Grande parte das coleções usam, internamente, um array para armazenar os seus dados. Quando esse array chega ao limite, é criado um maior e o conteúdo do antigo é copiado. Se tivermos uma coleção que cresce muito, isso pode ocorrer muitas vezes. Então, criamos uma coleção já com uma capacidade grande, para evitar o excesso de redimensionamento.
- Evite usar coleções que guardam os elementos pela sua ordem de comparação quando não há necessidade.
- Não itere sobre uma List utilizando um for de 0 até list.size() e usando get(int) para receber os objetos. Isso parece não ter problema, mas algumas implementações da List não são de acesso aleatório como a LinkedList, fazendo esse código ter uma péssima performance computacional. (use Iterator)

EXERCÍCIOS

- Criar uma classe Aluno com os atributos nome, nota1, e nota2.
- Criar construtor e getters.
- Criar um método calcularMedia que deverá retornar um double do cálculo da média
- Lançar uma exceção caso a primeira ou segunda nota for menor que zero ou superior a dez.
- Criar uma classe com o nome AlunoException com sua própria exceção.
- Criar uma classe de teste com o main e instanciar um objeto do tipo aluno passando o nome e as notas no construtor de aluno.
- Tratar possíveis erros de exceção de notas inválidas
- Criar uma lista de alunos, percorrer e imprimir.

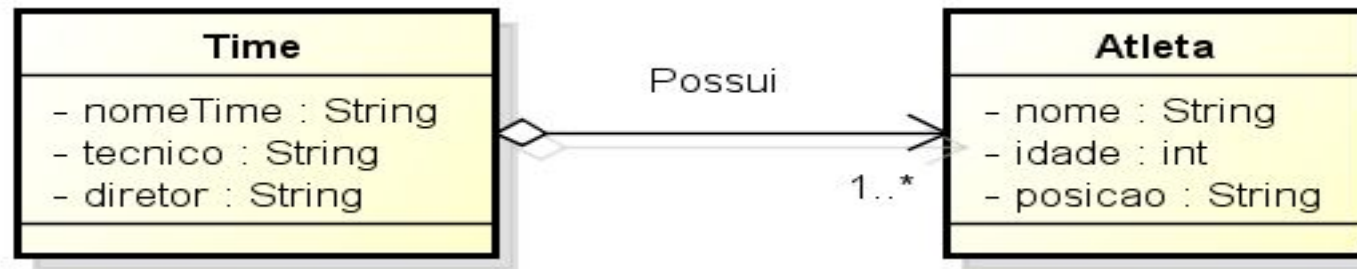
EXERCÍCIOS

- Crie uma classe chamada **Cliente** com os atributos: **id**, **nome**, **idade**, **telefone**. Faça um programa para solicitar os dados de vários clientes usando o **Scanner** e armazenar em um **ArrayList** até que se digite um número de id negativo. Em seguida exiba os dados de todos os clientes via **System.out**, formatando cada objeto separado por linhas.

```
public class Main {  
    public static void main(String[] args) {  
        Scanner in = new Scanner(System.in);  
        System.out.println("Digite um numero positivo ou zero para continuar:");  
        int num = in.nextInt();  
        List<Cliente> clientes = new ArrayList<>();  
        while (num >= 0) {  
            System.out.println("Digite a id");  
            long id = in.nextLong();  
            System.out.println("Digite a nome");  
            String telefone = in.next();  
            System.out.println("Digite a telefone");  
            String nome = in.next();  
            System.out.println("Digite a idade");  
            int idade = in.nextInt();  
  
            Cliente cliente = new Cliente(id, nome, idade, telefone);  
            clientes.add(cliente);  
  
            System.out.println("Digite um numero positivo ou zero para continuar:");  
            num = in.nextInt();  
        }  
        for (Cliente cliente : clientes) {  
            System.out.println(cliente.getNome());  
            System.out.println(cliente.getId());  
            System.out.println(cliente.getIdade());  
            System.out.println(cliente.getTelefone() + "\n");  
        }  
        in.close();  
    }  
}
```

```
public class Cliente {  
    private long id;  
    private String nome;  
    private int idade;  
    private String telefone;  
  
    //getters e setters  
  
    public Cliente(long id, String nome, int idade, String telefone) {  
        super();  
        this.id = id;  
        this.nome = nome;  
        this.idade = idade;  
        this.telefone = telefone;  
    }  
}
```


EXERCÍCIOS



- Crie as classes de acordo com o diagrama.
- Utilize List no relacionamento
- Crie uma main para testar o time. Crie um time, adicione atletas e depois imprima na tela o nome do time e o nome de seus atletas

EXERCÍCIOS

- Implemente uma classe em Java com as funcionalidades de uma agenda telefônica, associando um nome a um número telefônico.
- Crie um programa em Java para testar a classe AgendaTelefonica. Teste a classe com pelo menos 5 contatos diferentes na agenda de telefones.
- A classe deve ser de acordo com o diagrama abaixo:

