



Quem se prepara, não para.



Linguagens de Programação

3º período

Prof. Dr. João Paulo Aramuni

Sumário

- Java
 - Exceções
 - Classe ArrayList

Sumário

- Java
 - Exceções
 - Classe ArrayList

- Exceções
 - Declaração de Exceções
 - Manipulação de Exceções
 - Tratamento de Exceções

- Exceções
 - Considere o seguinte trecho de código:

```
public class Estoque {  
    private int codProduto;  
    private int qtdeEstoque;  
  
    public void decrementaEstoque(int qtde) {  
        qtdeEstoque = qtdeEstoque - qtde;  
    }  
}
```

- Exceções
 - O que fazer se o estoque não for suficiente?
 - Vamos avaliar três possibilidades:
 - Desconsiderar a operação;
 - Mostrar mensagem de erro;
 - Retornar código de erro.

- Exceções
 - O que fazer se o estoque não for suficiente?
 - Opção 1: Desconsiderar operação.

```
public class Estoque {  
    private int codProduto;  
    private int qtdeEstoque;  
    public void decrementaEstoque (int qtde) {  
        if (qtdeEstoque >= qtde)  
            qtdeEstoque = qtdeEstoque - qtde;  
    }  
}
```

- Exceções
 - Problema: Não há como saber se a operação foi realizada.
 - Nenhuma informação é retornada ao **cliente**.

- Exceções
 - O que fazer se o estoque não for suficiente?
 - Opção 2: Mostrar mensagem de erro.

```
public class Estoque {  
    private int codProduto;  
    private int qtdeEstoque;  
  
    public void decrementaEstoque(int qtde) {  
        if (qtdeEstoque >= qtde) {  
            qtdeEstoque = qtdeEstoque - qtde;  
        } else {  
            System.out.println("Estoque Insuficiente!");  
        }  
    }  
}
```

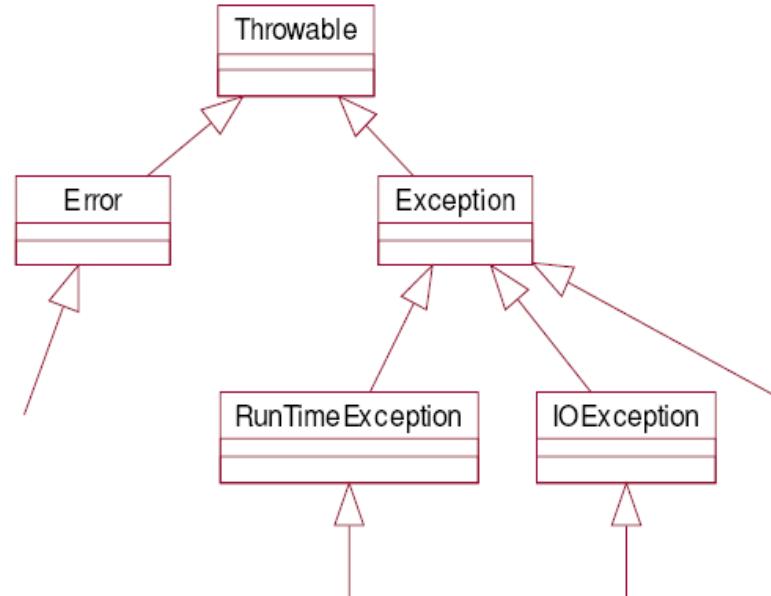
- Exceções
 - Problema: Gera dependência entre a classe e sua interface de usuário.
 - Não retorna informação ao **cliente**.

- Exceções
 - O que fazer se o estoque não for suficiente?
 - Opção 3: Retornar código de erro.

```
public class Estoque {  
    private int codProduto;  
    private int qtdeEstoque;  
  
    public boolean decrementaEstoque(int qtde) {  
        if (qtdeEstoque >= qtde) {  
            qtdeEstoque = qtdeEstoque - qtde;  
            return true;  
        } else  
            return false;  
    }  
}
```

- Exceções
 - Problema: Complica definição e uso do método (cliente tem que fazer testes);
 - Pior para métodos que já retornam valores.

- Exceções
 - Usar esquema de tratamento de exceções.
 - Hierarquia de exceções em Java:



- Exceções
 - **Error**
 - Descreve erros internos. Ex: falta de memória, disco cheio.
 - Não se deve lançar um objeto desse tipo (Error).
 - Pouco se pode fazer se um erro interno ocorre, além de notificar o usuário e tentar finalizar o programa.
 - **Exception**
 - **RunTimeException**
 - Ocorre porque houve um erro de programação (culpa do programador).
 - Ex: conversão explícita de tipo (cast), acesso a elemento de array além dos limites, acesso de apontador nulo.
 - Outras exceções
 - Ex: tentar ler além do final de um arquivo, tentar abrir URL incorreta, etc.

- Exceções
 - As exceções que derivam da classe **Error** ou da classe **RunTimeException** são chamadas exceções **não verificadas**. Todas as demais são chamadas exceções verificadas.
 - Um método precisa declarar todas as exceções verificadas que ele lança.
 - Palavra-chave **throws** no cabeçalho do método.

- Exceções
 - Exemplo ao gravar dados em um arquivo:

```
public void gravaDados() throws IOException {
    ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("rh.dat"));
    out.writeObject(funcionarios);
    out.close();
}
```

- Se o método não conseguir criar o arquivo ou gravar os dados, ocorrerá uma exceção do tipo **IOException**, que deverá ser tratada.

- Exceções

- A hierarquia de exceções possui várias classes para diversos tipos de exceções que podem ser lançadas pelos programas.
- Para lançar uma exceção, use a palavra-chave **throw**.

```
public String lerDados (BufferedReader ent) throws EOFException {  
    . . .  
    while ( . . . ) {  
        if (ch == -1)  
            if (n < compr)  
                throw new EOFException();  
    } }
```

- A classe **EOFException** sinaliza que ocorreu um fim de arquivo inesperado durante a entrada de dados.

- Exceções
 - Como criar suas próprias classes de **Exceção**?
 - Um programa pode ter um problema que não está descrito adequadamente em nenhuma das classes de exceção padrão.
 - Para criar um classe de exceção basta derivá-la de `Exception` ou de uma classe descendente de `Exception`.

- Exceções
 - Como criar suas próprias classes de **Exceção**?
 - É habitual criar um construtor padrão e um construtor que contenha uma mensagem detalhada.
 - Construtores/método **getMessage** da classe Throwable:
 - Throwable() : constrói um novo objeto sem mensagem detalhada.
 - Throwable (String mensagem) : constrói um novo objeto com a mensagem detalhada especificada.
 - String getMessage() : este método obtém a mensagem detalhada do objeto Throwable.

- Exceções
 - Lançamento de uma exceção criada:

```
public class Estoque {  
    private int codProduto;  
    private int qtdeEstoque;  
  
    // ...  
    public void decrementaEstoque(int qtde) throws EIException {  
        if (qtdeEstoque >= qtde)  
            qtdeEstoque = qtdeEstoque - qtde;  
        else  
            throw new EIException(codProduto, qtdeEstoque);  
    }  
}
```

- Criação de uma classe de exceção: **EIException**

```
3  public class EIException extends Exception {
4      private int codProd;
5      private int qtdeEst;
6
7  public EIException() {
8      super("Estoque Insuficiente!");
9  }
10
11 public EIException(int cod, int qtde) {
12     super("Estoque Insuficiente!");
13     codProd = cod;
14     qtdeEst = qtde;
15 }
16
17 public int getCodProduto() {
18     return codProd;
19 }
20
21 public int getQtdeEstoque() {
22     return qtdeEst;
23 }
24 }
```

- Exceções levantadas indiretamente também devem ser tratadas:

```
7  public class ExemploExcecoes {
8
9@   public class Estoque {
10      private int codProduto;
11      private int qtdeEstoque;
12
13      // ...
14@     public void decrementaEstoque(int qtde) throws EIException {
15         if (qtdeEstoque >= qtde)
16             qtdeEstoque = qtdeEstoque - qtde;
17         else
18             throw new EIException(codProduto, qtdeEstoque);
19     }
20 }
21
22@   public static void main(String[] args) {
23
24 }
25
26@   public class Pedido {
27     // ...
28@     public void adicionaItem(Estoque est, int qtde) throws EIException {
29         // ...
30         est.decrementaEstoque(qtde); // <-lança a exceção
31         // ...
32     }
33 }
```

- Exceções
 - Se não se desejar criar uma classe de exceção própria pode-se levantar uma exceção com uma mensagem personalizada através de uma classe pré-existente.

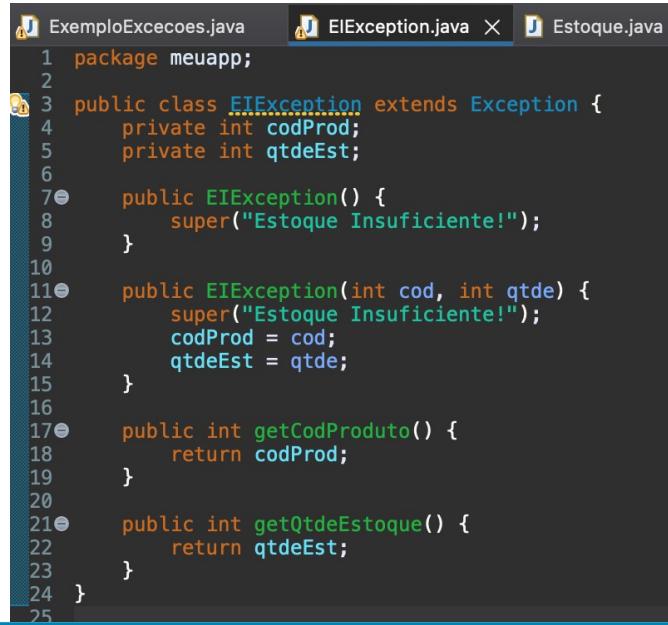
```
public void decrementaEstoque(int qtde) throws Exception {  
    if (qtdeEstoque >= qtde)  
        qtdeEstoque = qtdeEstoque - qtde;  
    else  
        throw new Exception("Estoque Insuficiente!");  
}
```

Mas lembre-se que a classe **Exception** é muito genérica e captura qualquer exceção.

- Exceções

- Criar uma classe de exceção própria, declarar (com **throws**) e levantar (com **throw**) a exceção:

```
public class Estoque {  
    private int codProduto;  
    private int qtdeEstoque;  
  
    public void decrementaEstoque(int qtde) throws EIException {  
        if (qtdeEstoque >= qtde)  
            qtdeEstoque = qtdeEstoque - qtde;  
        else  
            throw new EIException(codProduto, qtdeEstoque);  
    }  
}
```



```
ExemploExcecoes.java  
1 package meuapp;  
2  
3 public class EIException extends Exception {  
4     private int codProd;  
5     private int qtdeEst;  
6  
7     public EIException() {  
8         super("Estoque Insuficiente!");  
9     }  
10  
11    public EIException(int cod, int qtde) {  
12        super("Estoque Insuficiente!");  
13        codProd = cod;  
14        qtdeEst = qtde;  
15    }  
16  
17    public int getCodProduto() {  
18        return codProd;  
19    }  
20  
21    public int getQtdeEstoque() {  
22        return qtdeEst;  
23    }  
24}  
25
```

- Exceções
 - Captura de exceções:
 - As exceções devem ser capturadas e tratadas (verificadas) adequadamente.
 - Se ocorrer uma exceção e esta não for capturada em nenhum lugar, o programa termina mostrando uma mensagem de erro.
 - Para capturar uma exceção, especifica-se um bloco **try, catch** (p/ verificar) e **finally** (bloco de finalizações opcional).

- Captura de exceções:

```
try { // Capturar a exceção
    comando1;
    comando2; . . . comandon;
}
catch ( TipoExceção1 e1 ) { tratamento de e1 } //verificar
catch ( TipoExceção2 e2 ) { tratamento de e2 } . . .
catch ( TipoExceçãon en ) { tratamento de en } //verificar
finally { finalizações } //bloco de finalizações
```

```
try {
    // código normal
} catch (<exceção 1>) {
    // código de tratamento do primeiro tipo de erro
} catch (<exceção 2>) {
    // código de tratamento do segundo tipo de erro
} catch (<exceção 3>) {
    // código de tratamento do terceiro tipo de erro
}
```

```
public class ExemploExcecoes {
    public static void main(String[] args) {
        Estoque est = new Estoque(0, 200);

        try {
            est.decrementaEstoque(100);
        } catch (EIException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

- Captura de exceções:

```
public static void main(String[] args) {  
    Estoque est = new Estoque(0, 200);  
  
    try {  
        est.decrementaEstoque(100);  
    } catch (EIException e) {  
        System.out.println(e.getMessage() + " Produto: " + e.getCodProduto() + " Estoque: " + e.getQtdeEstoque());  
    } finally {  
        // Exemplos  
        // Fechar arquivo  
        // Encerrar conexão com o banco de dados  
        // Mensagem padrão, etc  
        System.out.println("Programa finalizado.");  
    }  
}
```

- Podem existir inúmeros blocos *catch* no tratamento de erros (cada um para um tipo de exceção).
- Com o *TRY-FINALLY* uma rotina de finalização é sempre executada (independentemente de ocorrer um erro). O trecho de código contido em *FINALLY* será sempre executado.

- Captura de exceções:
 - Se o código dentro do bloco try não lançar nenhuma exceção:
 - O programa executa o código dentro de try, **pula a cláusula catch**, executa o código dentro da cláusula finally se estiver presente e o restante do código.
 - Se qualquer parte do código dentro do bloco try lançar uma exceção da classe especificada na cláusula catch:
 - O programa pula o restante do código do bloco try a partir do ponto onde a exceção foi lançada, executa o código da cláusula catch correspondente e, então, o código da cláusula finally.
 - Se a cláusula catch não lançar nenhuma exceção, o programa executa a primeira linha depois do bloco try. Senão, a exceção é lançada de volta para o chamador do método.

- Captura de exceções:
 - Se código dentro do bloco **try** lançar uma exceção que não é capturada por nenhuma cláusula **catch**:
 - o programa pula o restante do código do bloco try a partir do ponto onde a exceção foi lançada, executa o código na cláusula **finally** e lança a exceção de volta para o chamador do método.
 - Para capturar múltiplas exceções:
 - Usam-se cláusulas **catch** separadas, uma para cada tipo de exceção.
 - Deve-se colocar as classes mais específicas primeiro.

- Exemplo de múltiplas exceções:

```
2
3  public class ExemploExcecoes {
4
5@   public static void main(String[] args) {
6      Estoque est = new Estoque(0, 200);
7
8      try {
9          est.decrementaEstoque(100);
10     } catch (EIException e) {
11         System.out.println(e.getMessage() + " Produto: " + e.getCodProduto() + " Estoque: " + e.getQtdeEstoque());
12     } catch (NullPointerException e) {
13         System.out.println(e.getMessage());
14     } catch (Exception e) {
15         System.out.println(e.getMessage());
16     } finally {
17         // Exemplos
18         // Fechar arquivo
19         // Encerrar conexão com o banco de dados
20         // Mensagem padrão, etc
21         System.out.println("Programa finalizado.");
22     }
23
24 }
25 }
```

- Exceções podem ser relançadas dentro da cláusula **catch**.
 - Relançar uma exceção com **throw** dentro de um bloco **catch** é útil quando você deseja propagar uma exceção capturada para um nível superior da pilha de chamadas de métodos, permitindo que ela seja tratada em um contexto mais abrangente ou em um local mais adequado.

```
try {  
    est.decrementaEstoque(100);  
} catch (EIEException e) {  
    System.out.println(e.getMessage() + " Produto: " + e.getCodProduto() + " Estoque: " + e.getQtdeEstoque());  
} catch (NullPointerException e) {  
    System.out.println(e.getMessage());  
} catch (Exception e) {  
    System.out.println(e.getMessage());  
    throw e; // <- relançar exceção  
} finally {  
    // Exemplos  
    // Fechar arquivo  
    // Encerrar conexão com o banco de dados  
    // Mensagem padrão, etc  
    System.out.println("Programa finalizado.");  
}
```

- Outro exemplo com múltiplas exceções:

```
try {
    leDados();
} catch (ClassNotFoundException e) {
    System.out.println("O arquivo de dados não é compatível. " + "Um novo arquivo será criado!");
} catch (IOException e) {
    System.out.println("Arquivo de dados não encontrado. " + "Um novo arquivo será criado!");
} catch (Exception e) {
    System.out.println(e.getMessage());
} finally {
    System.out.println("Programa finalizado.");
}
```

```
private static void leDados() throws IOException, ClassNotFoundException {
    ObjectInputStream in = new ObjectInputStream(new FileInputStream("estoque.dat"));
    ArrayList<Estoque> estoque = new ArrayList<Estoque>();
    estoque = (ArrayList<Estoque>) in.readObject();
    System.out.println(estoque);
    in.close();
}
```

- Outro exemplo com múltiplas exceções:

```
int j = 10;  
  
try {  
  
    while (j > Integer.parseInt (args [0])) {  
  
        System.out.println ("+"+j);  
  
        j--;  
    }  
  
} catch (ArrayIndexOutOfBoundsException e){  
  
    System.out.println ("Não foi fornecido um argumento.");  
}  
catch (java.lang.NumberFormatException e){  
  
    System.out.println("Argumento não é um inteiro válido.");  
}
```

Por exemplo, podemos criar um programa que precisa receber um número inteiro da **linha de comando** (args [0]).

Como os argumentos são passados em um vetor de *strings*, precisamos transformar a *string* contendo o número para um inteiro.

Se a conversão gerar um erro, significa que o argumento não é um número inteiro válido. A exceção usada, nesse caso, é a **java.lang.NumberFormatException**.

Sumário

- Java
 - Exceções
 - **Classe ArrayList**

- Classe ArrayList

- A classe ArrayList implementa vetores dinâmicas (ou redimensionáveis).
- Principais métodos:
 - boolean **add** (Object element): Adiciona o elemento especificado no final da lista.
 - void **add(int index, Object element)**: Insere o elemento especificado na posição indicada da lista.
 - void **clear()**: Remove todos os elementos da lista.
 - boolean **contains(Object element)**: Retorna verdadeiro se a lista contém o elemento especificado e falso caso contrário.

- Classe ArrayList
 - A classe ArrayList implementa vetores dinâmicas (ou redimensionáveis).
 - Principais métodos:
 - Object **get(int index)**: Retorna o i-ésimo elemento da lista.
 - int **indexOf(Object element)**: Retorna a posição da primeira ocorrência do elemento especificado na lista.
 - boolean **isEmpty**: Retorna verdadeiro se a lista estiver vazia e falso caso contrário.
 - int **lastIndexOf(Object element)**: Retorna a posição da última ocorrência do elemento especificado na lista.

- Classe ArrayList

- A classe ArrayList implementa vetores dinâmicas (ou redimensionáveis).
- Principais métodos:
 - Object **remove(int index)**: Remove o i-ésimo elemento da lista.
 - Object **set(int index, Object element)**: Substitui o i-ésimo elemento da lista pelo elemento especificado.
 - int **size()**: Retorna o número de elementos da lista.

Exemplo Classe ArrayList

Agenda

The screenshot shows an IDE interface with two main sections: a code editor and a terminal window.

Code Editor (ExemploArrayList.java):

```
1 package meuapp;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5 import java.util.Scanner;
6
7 public class ExemploArrayList {
8     public static void main(String[] args) {
9         Scanner ler = new Scanner(System.in);
10
11         // [ A ] declarando e instanciando uma agenda de contatos
12         ArrayList<String> agenda = new ArrayList<String>();
13
14         // [ B ] usando o método add() para gravar 4 contatos na agenda
15         agenda.add("Juca Bala;11 1111-1111");
16         agenda.add("Marcos Paqueta;22 2222-2222");
17         agenda.add("Maria Antonieta;33 3333-3333");
18         agenda.add("Antônio Conselheiro;44 4444-4444");
19     }
20 }
```

Terminal Console Output:

```
Console X Problems Javadoc Declaration Search Breakpoints
<terminated> ExemploArrayList [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/H
Percorrendo o ArrayList (usando o índice)
Posição 0- Juca Bala;11 1111-1111
Posição 1- Marcos Paqueta;22 2222-2222
Posição 2- Maria Antonieta;33 3333-3333
Posição 3- Antônio Conselheiro;44 4444-4444

Informe a posição a ser excluída:
1

Percorrendo o ArrayList (usando for-each)
Posição 0- Juca Bala;11 1111-1111
Posição 1- Maria Antonieta;33 3333-3333
Posição 2- Antônio Conselheiro;44 4444-4444

Percorrendo o ArrayList (usando iterator)
Posição 0- Juca Bala;11 1111-1111
Posição 1- Maria Antonieta;33 3333-3333
Posição 2- Antônio Conselheiro;44 4444-4444
```

Exemplo Classe ArrayList

Agenda

```
0  public class ExemploArrayList {
1
2  public static void main(String[] args) {
3      Scanner ler = new Scanner(System.in);
4
5      // [ A ] declarando e instanciando uma agenda de contatos
6      ArrayList<String> agenda = new ArrayList<String>();
7
8      // [ B ] usando o método add() para gravar 4 contatos na agenda
9      agenda.add("Juca Bala;11 1111-1111");
10     agenda.add("Marcos Paqueta;22 2222-2222");
11     agenda.add("Maria Antonieta;33 3333-3333");
12     agenda.add("Antônio Conselheiro;44 4444-4444");
13
14     int i;
15
16     // [ C ] mostrando os "n" contatos da agenda (usando o índice)
17     // número de elementos da agenda: método size()
18     System.out.printf("Percorrendo o ArrayList (usando o índice)\n");
19     int n = agenda.size();
20     for (i = 0; i < n; i++) {
21         System.out.printf("Posição %d- %s\n", i, agenda.get(i));
22     }
23
24     System.out.printf("\nInforme a posição a ser excluída:\n");
25     i = ler.nextInt();
26
27     try {
28         // [ D ] remove o i-ésimo contato da agenda
29         agenda.remove(i);
30     } catch (IndexOutOfBoundsException e) {
31         // exceção lançada para indicar que um índice (i)
32         // está fora do intervalo válido (de 0 até agenda.size()-1)
33         System.out.printf("\nErro: posição inválida (%s).", e.getMessage());
34     }
35
36     // [ E ] mostrando os "n" contatos da agenda (usando for-each)
37     System.out.printf("\nPercorrendo o ArrayList (usando for-each)\n");
38     i = 0;
39     for (String contato : agenda) {
40         System.out.printf("Posição %d- %s\n", i, contato);
41         i++;
42     }
43 }
```

Agenda

```
// [ F ] mostrando os "n" contatos da agenda (com iterator)
System.out.printf("\nPercorrendo o ArrayList (usando iterator)\n");
i = 0;
Iterator<String> iterator = agenda.iterator();
while (iterator.hasNext()) {
    System.out.printf("Posição %d- %s\n", i, iterator.next());
    i++;
}
ler.close();
}
```

Outro exemplo classe ArrayList

Loteria

The screenshot shows an IDE interface with a code editor and a terminal window.

Code Editor: The file `ExemploArrayList2.java` contains the following Java code:

```
1 package meuapp;
2
3 import java.util.ArrayList;
4 import java.util.Random;
5 import java.util.Scanner;
6
7 public class ExemploArrayList2 {
8     public static void main(String[] args) {
9         // Gerar a loteria aleatória
10        ArrayList<Integer> numerosLoteria = gerarNumerosLoteria(6, 1, 60);
11        System.out.println(numerosLoteria);
12        // Ler o número do usuário
13        int numeroEscolhido = lerNumero();
14
15        // Verificar se o número acertou na loteria
16        boolean acertou = verificarAcerto(numeroEscolhido, numerosLoteria);
17
18        // Exibir o resultado
19        if (acertou) {
20            System.out.println("Parabéns! Seu número acertou na loteria.");
21        } else {
22            System.out.println("Não foi dessa vez. Tente novamente.");
23        }
24    }
25
26    // Método para gerar os números da loteria aleatoriamente
27    private static ArrayList<Integer> gerarNumerosLoteria(int quantidadeNumeros, int minimo, int maximo) {
28        ArrayList<Integer> numerosLoteria = new ArrayList<>();
29        Random random = new Random();
30
31        while (numerosLoteria.size() < quantidadeNumeros) {
32            int numero = random.nextInt(maximo - minimo + 1) + minimo;
33            if (!numerosLoteria.contains(numero)) {
34                numerosLoteria.add(numero);
35            }
36        }
37
38        return numerosLoteria;
39    }
40}
```

Terminal: The terminal window shows the application's output:

```
Console X Problems Javadoc Declaration Search Breakpoints
<terminated> ExemploArrayList2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (26 de jul. de 2023 15:47:32 - 15:47:32)
[15, 51, 3, 57, 38, 56]
Digite um número entre 1 e 60: 3
Parabéns! Seu número acertou na loteria.
```

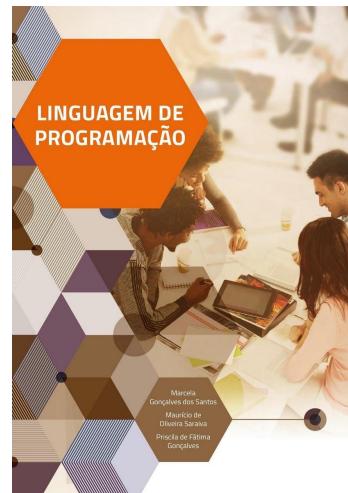
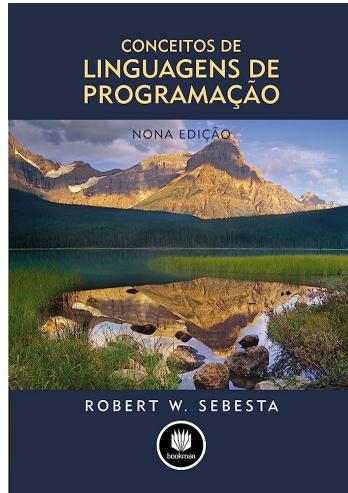
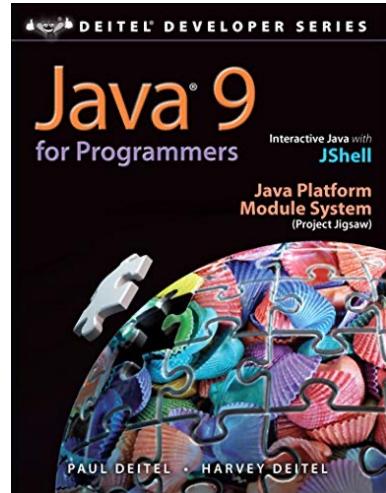
Outro exemplo classe ArrayList

Loteria

```
// Método para ler o número do usuário via teclado
private static int lerNumero() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Digite um número entre 1 e 60: ");
    int numero = scanner.nextInt();
    scanner.close();
    return numero;
}

// Método para verificar se o número do usuário acertou na loteria
private static boolean verificarAcerto(int numeroEscolhido, ArrayList<Integer> numerosLoteria) {
    return numerosLoteria.contains(numeroEscolhido);
}
```

Referências





Obrigado!

joaoaramuni@newtonpaiva.br