

## Estruturas de controle.

Em geral as instruções em um programa são executadas uma após a outra mesmo na P.O.O onde podemos criar objetos e chamar seus métodos na medida em que se faz necessário não importando a ordem em que esses métodos foram implementados na classe no final a execução acaba sendo sequencial pois criamos os objetos que chamam os seus métodos dentro do método “main” da classe principal e a execução desses acaba sendo uma linha após a outra, além disso se pensarmos que quando um método é invocado por um objeto ele executa o código em seu corpo linha a linha, ou seja no final tudo é sequencial.

As estruturas de controle permitem alterar esse “fluxo” de execução permitindo que partes do código sejam ignoradas ou que partes do código sejam executadas um determinado número de vezes sem que haja necessidade de replicar esse código.

O JAVA como muitas outras linguagens possui três grupos de estruturas de controle, **estrutura de sequência, estrutura de seleção e estrutura de repetição.**

- **Estrutura de sequência:** É a estrutura padrão do JAVA e a não ser que o programador implemente as outras duas todo o programa quem JAVA vai seguir essa estrutura que consiste em executar uma linha código após a outra até o fim do programa.
- **Estrutura de seleção:** Em uma definição bastante simples essa estrutura permite que uma parte do código seja selecionado para ser executado enquanto que outra parte seja ignorada. Podemos subdividir essa estrutura em outros três grupos “**instrução de seleção simples**”, “**instrução de seleção dupla**” e **instrução de seleção múltipla**.
  - **Instrução de seleção simples (if):** Recebe esse nome porque seleciona ou ignora um único grupo de códigos;
  - **Instrução de seleção dupla (if...else):** Recebe esse nome pois é capaz de selecionar um entre dois grupos de códigos sendo que um sempre será ignorado. É possível encadear várias instruções de seleção dupla uma seguida da outra criando assim uma espécie de bateria de testes;
  - **Instrução de seleção múltipla (switch):** Recebe esse nome devido a sua capacidade trabalhar com muitos blocos de código selecionado sempre um e ignorando o restante.
- **Estrutura de repetição:** Permite que um determinado bloco de código seja executado um certo número de vezes baseado em uma condição que vai definir se a repetição (**loop**) deve ou não continuar o JAVA possui três tipos de estruturas de repetição “**while**”, “**do...while**” e “**for**” além de uma

espécie de “for” especial chamado “foreach” que é especializado no trato de vetores, veremos as estruturas de repetição em detalhes mais à frente nesse curso. **Operadores relacionais e lógicos no JAVA.**

Para poder utilizar as estruturas de controle de forma eficiente no JAVA devemos conhecer os seus **operadores relacionais e lógicos**, então vamos dar uma rápida olhada nesses operadores.

Os relacionais se dividem em dois grupos “**operadores de igualdade**” que comparam se valores são iguais ou diferentes entre si e os “**operadores relacionais**” que comparam a grandeza entre valores (maior, menor, maior ou igual ou menor ou igual). As tabelas abaixo ilustram todos os operadores de igualdade e relacionais do JAVA:

Operadores de igualdade		
Operador	Exemplo	Descrição
==	X == Y	X é igual a Y
!=	X != Y	X é diferente de Y

Operadores relacionais		
Operador	Exemplo	Descrição
>	X > Y	X é maior que Y
<	X < Y	X é menor que Y
>=	X >= Y	X é maior ou igual a Y
<=	X <= Y	X é menor ou igual a Y

Já os operadores lógicos tratam da relação lógica entre o resultado booleano de pelo menos dois testes distintos oriundos dos operadores relacionais e/ou operadores de igualdade. Os operadores lógicos se dividem três grupos, são eles:

- **Operadores lógicos de conjunção** - No JAVA esse operador é representado pelos caracteres “&&”, mas pode ser encontrado em outras linguagens como “AND” e vai retornar verdadeiro quando ambas as proposições (condições a esquerda e a direita) forem verdadeiras. A tabela abaixo ilustra o conceito:

Operador lógico de conjunção		
Resultado do teste 1	Resultado do teste 2	Resultado da conjunção
Verdadeiro (true)	Verdadeiro (true)	Verdadeiro (true)
Verdadeiro (true)	Falso (false)	Falso (false)

Falso (false)	Verdadeiro (true)	Falso (false)
Falso (false)	Falso (false)	Falso (false)

- **Operadores lógicos de disjunção inclusiva** – No JAVA esse operador é representado pelos caracteres “||”, mas pode ser encontrado em outras linguagens como “OR” e vai retornar verdadeiro quando uma das proposições (condições a esquerda e a direita) forem verdadeiras. A tabela abaixo ilustra o conceito:

Operador lógico de disjunção inclusiva		
Resultado do teste 1	Resultado do teste 2	Resultado da conjunção
Verdadeiro (true)	Verdadeiro (true)	Verdadeiro (true)
Verdadeiro (true)	Falso (false)	Verdadeiro (true)
Falso (false)	Verdadeiro (true)	Verdadeiro (true)
Falso (false)	Falso (false)	Falso (false)

- **Operador lógico de negação** – No JAVA esse operador é representado pelos caracteres “!”, mas pode ser encontrado em outras linguagens como “NOT” e sua finalidade é bastante simples, negar ou inverter um estado logico booleano. A tabela abaixo ilustra o conceito:

Operador lógico de negação	
Condição	Resultado da negação lógica
Verdadeiro (true)	Falso (false)
Falso (false)	Verdadeiro (true)

### Exemplos de aplicação de operadores relacionais e lógico do JAVA.

Os testes abaixo são realizados levando em conta que “X” vale dez (X=10) e que “Y” vale cinco (Y=5):

Teste	Descrição	Resultado
$X == Y$	Se a variável "X" é igual a Y.	Falso(false)
$Y != X$	Se a variável Y é diferente de X.	Verdadeiro(true)
$X < Y$	Se a variável X é menor que Y.	Falso(false)
$Y <= X$	Se a variável Y é menor ou igual a X.	Verdadeiro(true)
$(Y <= X) \ \&\& \ (Y != X)$	Se a variável Y é menor ou igual a X <b>E</b> se a variável Y é diferente de X.	Verdadeiro(true)
$(X == Y) \    \ (X < Y)$	Se a variável "X" é igual a Y <b>OU</b> Se a variável X é menor que Y.	Falso(false)
$!((Y <= X) \ \&\& \ (Y != X))$	Negação de: Se a variável Y é menor ou igual a X <b>E</b> se a variável Y é diferente de X.	Falso(false)

### Estrutura de seleção simples (if).

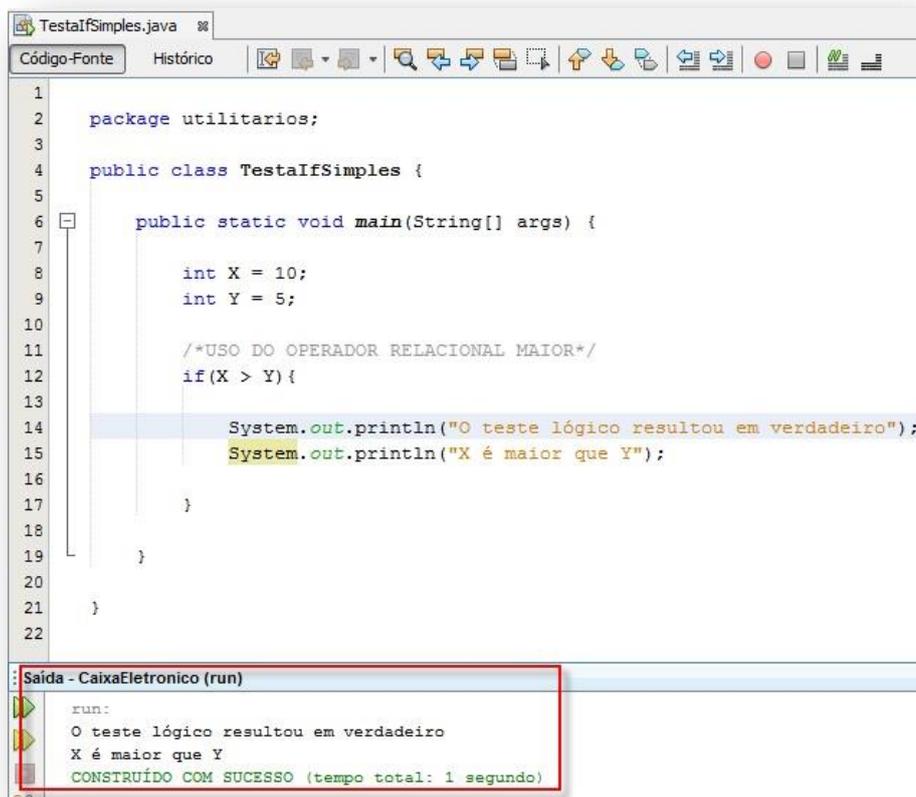
Essa estrutura permite que o programa escolha um fluxo alternativo na execução do código, essa escolha é baseada em um teste lógico que pode retornar verdadeiro ("true") ou falso ("false") e geralmente esses testes envolvem operadores de comparação ou relacionais ou ambos quando precisamos realizar teste múltiplos. A estrutura de seleção simples só trata casos em que o teste lógico resulte em um valor booleano verdadeiro.

#### Sintaxe:

```
if(teste_lógico){ bloco_de_código_caso_verdadeiro
}
```

A imagem abaixo ilustra a aplicação do exemplo das variáveis "X" e "Y" das

tabelas:



```
1
2 package utilitarios;
3
4 public class TestaIfSimples {
5
6     public static void main(String[] args) {
7
8         int X = 10;
9         int Y = 5;
10
11         /*USO DO OPERADOR RELACIONAL MAIOR*/
12         if(X > Y){
13
14             System.out.println("O teste lógico resultou em verdadeiro");
15             System.out.println("X é maior que Y");
16
17         }
18
19     }
20
21 }
22
```

Saida - CaixaEletronico (run)

```
run:
O teste lógico resultou em verdadeiro
X é maior que Y
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

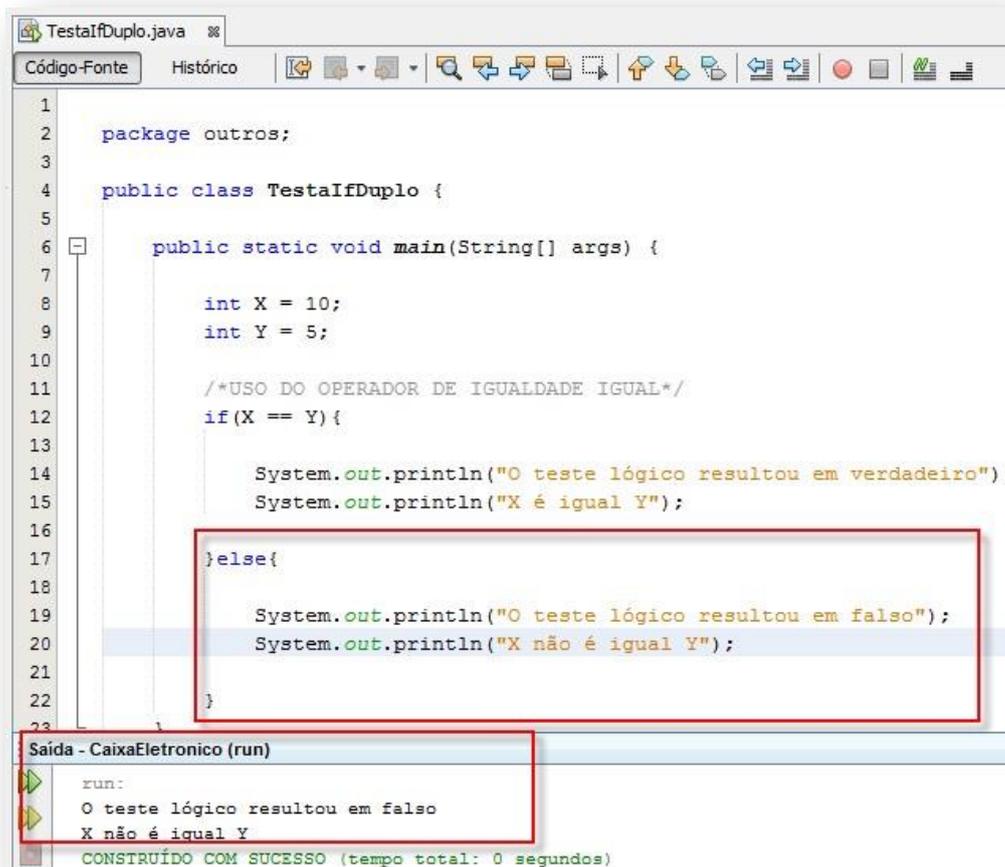
Observe o teste lógico usando operadores relacionais na linha doze (12) onde é inquirido se a variável “X” é maior “Y” o resultado é verdadeiro (**true**) então o código do corpo do “if” será executado, caso o resultado fosse falso (**false**) o programa terminaria sem realizar nenhuma saída dado o fato de que a estrutura de seleção simples “if” não trata resultados booleanos falsos.

### Estrutura de seleção dupla/composta (if...else).

Essa estrutura de seleção trabalha com o mesmo conceito da estrutura de seleção simples porém aqui teremos tratamento tanto para resultados booleanos verdadeiro (**true**) quanto para resultados booleanos falsos (**false**).

A imagem abaixo ilustra a aplicação do exemplo das variáveis “X” e “Y” das

tabelas:



```
1 package outros;
2
3
4 public class TestaIfDuplo {
5
6     public static void main(String[] args) {
7
8         int X = 10;
9         int Y = 5;
10
11         /*USO DO OPERADOR DE IGUALDADE IGUAL*/
12         if(X == Y){
13
14             System.out.println("O teste lógico resultou em verdadeiro");
15             System.out.println("X é igual Y");
16
17         }else{
18
19             System.out.println("O teste lógico resultou em falso");
20             System.out.println("X não é igual Y");
21
22         }
23     }
24 }
```

Saída - CaixaEletronico (run)

```
run:
O teste lógico resultou em falso
X não é igual Y
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Observe que dessa vez mudamos o teste lógico da linha doze (12) verificando agora se “X” é igual a “Y” o que resulta em falso, logo as linhas de código adjacentes ao “if” serão ignoradas, note que na linha dezessete (17) temos uma estrutura o “else” que trata os resultados falsos.

### Alterando a aplicação de “Caixa Eletrônico”.

Agora vamos aplicar esses conceitos em nossa aplicação de caixa eletrônico tornando essa mais “inteligente” e autônoma para atender nosso usuário final.

Até esse momento os dados referentes a depósitos e saque eram passados manualmente na classe principal da aplicação no momento da chamada dos métodos vamos realizar a primeira alteração focando nesse problema criando uma entrada (input) de dados.

Para tal vamos alterar a classe principal para que essa possa pedir valores para saque e depósito, receber e armazenar esses dados em variáveis e finalmente passa esses valores para os respectivos métodos que executam essas ações.

Como ainda estamos trabalhando em ambiente de texto vamos usar uma classe específica do JAVA que recupera dados enviados pelo teclado para ambientes de texto o nome dessa classe é “**Scanner**”.

O JAVA é composto por várias classes prontas que ajudam o programador a realizar tarefas corriqueiras e comuns a quase todos os programas entradas (**inputs**) e saídas (**outputs**) de dados se encaixam entre essas tarefas, essas classes podem já estar integradas aos nossos programas JAVA como é o caso da classe “**System**” que temos usado para realizar saída de dados no terminal. As classes internas do JAVA são organizadas em pacotes a classe “**System**” por exemplo está dentro do pacote “**java.lang**” esse pacote é adicionado automaticamente a todo projeto do JAVA, logo não precisamos realizar nenhum tipo de ação no código para utilizar a classe “**System**”.

Já a classe “Scanner” está no pacote “**java.util**” e esse pacote não está incluso por padrão nos nossos projetos do JAVA, logo para poder utilizar essa classe precisamos importar (**import**) esse pacote.

A imagem abaixo ilustra as alterações feitas na classe principal:

```
1 package caixaeletronico;
2
3 import utilitarios.Conta;
4 import java.util.Scanner;
5
6 public class CaixaEletronico {
7
8     public static void main(String[] args) {
9
10        //Cria um objeto da classe Scanner.
11        Scanner sc = new Scanner(System.in);
12        //Cria um objeto da classe Conta.
13        Conta objConta = new Conta();
14
15        //Declara uma variável que vai receber os valores digitados.
16        double valor;
17
18        //Mostra o saldo antes de qualquer operação.
19        objConta.saldo();
20
21        //Solicita um valor para depósito
22        System.out.println("Digite um valor para depósito:");
23        //Recebe um double valor digitado para depósito
24        //através do método nextDouble do objeto de Scanner.
25        valor = sc.nextDouble();
26
27        //Realiza um depósito.
28        objConta.deposito(valor);
29        System.out.println("Realizado um depósito de R$ 100,00");
30
31        //Mostra o saldo após a operação de depósito.
32        objConta.saldo();
33
34        //Solicita um valor para depósito
35        System.out.println("Digite um valor para saque:");
36        //Recebe um double valor digitado para saque através do método
37        //nextDouble do objeto de Scanner.
38        valor = sc.nextDouble();
39
40        //Realiza um saque.
41        objConta.saque(valor);
42        System.out.println("Realizado um saque de R$ 250,00");
43
44        //Mostra o saldo após a operação de saque.
45        objConta.saldo();
46
47    }
48
49 }
```

A linha quatro (4) realiza a importação da classe “**Scanner**” do pacote “**java.util**” isso nos permite criar um objeto dessa classe dentro da classe principal do nosso projeto e explorar seus métodos de recuperação de valores digitados pelo usuário em um teclado.

A linha onze (11) cria um objeto da classe “**Scanner**” em uma variável de instancia de objeto chamada de “**sc**”.

A linha dezesseis (16) declara uma variável local do tipo “**double**” para receber os valores digitados pelo usuário.

A linha vinte e dois (22) envia uma mensagem ao usuário solicitando que esse informe um valor para depósito e a linha vinte e cinco (25) recebe esse valor e o atribui a variável “valor”, observe que o valor é recebido chamando o método “**nextDouble()**” através do objeto da classe “**Scanner**” “**sc**”, existem vários tipos de dados no JAVA, logo existem vários tipos de métodos da classe “**Scanner**” para receber diferentes tipos de dados (“**nextByte()**”, “**nextFloat()**”, “**nextInt()**” etc.).

A linha vinte e oito (28) passa variável “valor” como parâmetro para o método deposito, lembre-se que antes isso era feito manualmente.

A linha vinte e nove (29) realiza uma saída formatada “**System.out.printf**(“Realizado um depósito de R\$ %.2f\n”, valor)” o método “**printf**” permite definir uma formatação de saída de dados nesse caso “**%.2f**” a “**%**” indica uma saída formatada o “**.2**” que pode existir até dois valores numéricos após o ponto e o “**f**” que vai ser um valor real.

Todo o processo é repetido para a operação de saque da linha trinta e cinco (35) até a linha quarenta e dois (42).

Se analisarmos esse código vamos notar que primeiro as “transações bancárias” são feitas sequencialmente sem dar opção para o usuário e segundo que o código está crescendo sem nenhum controle, vamos resolver esses dois problemas em breve.

### **Validando os métodos de depósito e saque.**

Nossa aplicação tem dois problemas nos métodos de depósito e de saque é possível depositar valores negativos e realizar saques com valores acima do saldo (não é uma conta especial, ainda!).

Para resolver esses dois problemas precisamos alterar esses dois métodos adicionando uma estrutura de seleção dupla em cada um, onde em depósito será realizado um teste para verificar se o valor informado é maior que zero e em saque se o valor informado é menor ou igual ao valor de saldo.

A imagem abaixo ilustra a alteração realizada na no método “deposito” da

classe “Conta”:

```
24 public void deposito(double valor) {
25
26     if(valor > 0){
27
28         setSaldo(getSaldo() + valor);
29
30     }else{
31
32         System.out.println("Valor inválido.");
33
34     }
35 }
```

Como podemos observar no trecho de código acima a solução foi bastante simples a linha vinte e seis (26) declara uma instrução “if” que verifica se a variável de parâmetro “valor” é maior que zero (0) se o resultado do teste lógico for verdadeiro a operação de depósito é realizada caso contrário (seja falsa) o “else” assume o fluxo do programa e emite uma mensagem para o usuário informando o erro.

A imagem abaixo ilustra a alteração realizada na no método “saque” da classe “Conta”:

```
10 public void saque(double valor) {
11
12     if(valor <= getSaldo()){
13
14         setSaldo(getSaldo() - valor);
15
16     }else{
17
18         System.out.println("Saldo insuficiente.");
19
20     }
21
22 }
```

Aqui temos uma solução muito semelhante a aplicada no método “deposito” na linha doze (12) é testado se o conteúdo da variável “valor” é menor ou igual ao do atributo “saldo”, caso o teste resulte em verdadeiro a operação de saque é realizada do contrário o comando “else” assume o fluxo do programa e emite uma mensagem de erro para o usuário, observe o uso do método “getSaldo” isso é uma aplicação correta do conceito de encapsulamento o atributo em questão não é acessado diretamente.

### A instrução de seleção múltipla (switch).

Essa instrução realiza ações diferentes com base no teste de um valor pertencente aos tipos “byte”, “short”, “int” ou “char” onde um dado pertencente a um desse valores vai passar por uma série de teste de comparação de igualdade (o único teste que a estrutura “switch” é capaz de executar) quando um valor verdadeiro for encontrado o bloco de código adjacente ao teste será executado, caso não haja teste de resultado verdadeiro uma opção para falso pode ser definida.

#### Sintaxe:

```
switch(valor_a_ser_comparado){ case  
    valor_de_comparação:  
        bloco_de_código_caso_verdadeiro  
    break; case valor_de_comparação:  
        bloco_de_código_caso_verdadeiro  
    break; case valor_de_comparação:  
        bloco_de_código_caso_verdadeiro  
    break; default:  
        bloco_de_código_caso_verdadeiro  
}
```

Observe que no final de cada comando “case” temos um comando “break”, isso se faz necessário para que o fluxo do programa dentro da estrutura “switch” seja abortado sempre um teste resultar em verdadeiro sem o uso do comando “break” os outros “cases” seriam executados também.

Observe também que no exemplo da sintaxe o comando “default” foi colocado no final da sequência isso não é obrigatório poderíamos ter colocado o comando em questão no início ou até mesmo no meio da estrutura, observe também que o comando “default” não possui comando “break” isso é porque o último

comando da estrutura não precisa, logo se tirarmos o comando default do “fim da fila” teríamos que adicionar um comando “**break**”.

### **Resolvendo os problemas da classe principal do projeto do caixa eletrônico.**

Agora que temos mais um conceito de estruturas de seleção a nosso favor vamos resolver os dois problemas (excesso de código e execução sequencial sem opção para o usuário) da classe principal da nossa aplicação de caixa eletrônico.

Queremos que nosso usuário possa escolher a opção que quer executar em nosso caixa eletrônico então vamos definir um menu de três opções, onde:

1 – Saque;

2 – Depósito;

3 – Saldo;

A imagem abaixo ilustra as alterações da classe principal do projeto caixa eletrônico:

```
1 package caixaeletronico;
2
3 import utilitarios.Conta;
4 import java.util.Scanner;
5
6 public class CaixaEletronico {
7
8     public static void main(String[] args) {
9
10         //Cria um objeto da classe Scanner.
11         Scanner sc = new Scanner(System.in);
12         //Cria um objeto da classe Conta.
13         Conta objConta = new Conta();
14
15         //Declara uma variável que vai receber a opção digitada pelo usuário.
16         int opcao;
17         //Declara uma variável que vai receber os valores digitados.
18         double valor;
19
20         //Monta o menu de opções para o usuário escolher a operação desejada e
21         //solicita uma escolha.
22         System.out.println("Escolha uma opção:");
23         System.out.println("1 - Saque");
24         System.out.println("2 - Depósito");
25         System.out.println("3 - Saldo");
26
27         //Recebe a opção escolhida pelo usuário.
28         opcao = sc.nextInt();
29
30         //Estrutura switch que testa qual foi a opção escolhida pelo usuário
31         switch(opcao){
32
33             //Trata a opção de saque.
34             case 1:
35                 System.out.println("Digite o valor desejado para saque:");
36                 valor = sc.nextDouble();
37                 objConta.saque(valor);
38                 objConta.saldo();
39                 break;
40
41             //Trata a opção de depósito.
42             case 2:
43                 System.out.println("Digite o valor do depósito:");
44                 valor = sc.nextDouble();
45                 objConta.deposito(valor);
46                 objConta.saldo();
47                 break;
48
49             //Trata a opção de saldo.
50             case 3:
51                 objConta.saldo();
52                 break;
53
54             //Trata opções inválidas.
55             default:
56                 System.out.println("Opção inválida");
57
58         }
59     }
60 }
61
62 }
```

A linha dezesseis (16) declara uma nova variável “ **opcao**” que vai receber o valor da operação que o usuário quer executar.

As linhas de vinte e dois a vinte e cinco (22-25) montam o menu com as opções e solicita a escolha de uma delas para o usuário.

A linha vinte e oito (28) recebe o valor digitado através do método “**nextInt()**” e o atribui a variável “**opcao**”.

A linha trinta e um (31) inicia a estrutura de seleção múltipla “**switch**” passando como valor a ser testado a variável “**opcao**”.

A linha trinta e quatro (34) traz o primeiro comando “**case**” (**case 1:**) que trata da operação de saque, caso o valor da variável “opcao” seja “1” será solicitado ao usuário que insira um valor de saque que é recebido e atribuído a variável “**valor**” (**valor = sc.nextDouble()**) linha trinta e seis (36) e passado para o método saque através do objeto da classe “**Conta**” (**objConta.saque(valor)**) a linha trinta e oito (38) mostra o saldo após a operação de saque (**objConta.saldo()**) e finalmente o comando “**break**” aborta a execução da estrutura.

As linhas quarenta e dois (42) até a quarenta e sete (47) realizam o tratamento para a operação de depósito seguindo a mesma linha de raciocínio vista anteriormente.

As linhas cinquenta (50) até cinquenta e dois (52) tratam a operação de saldo que é bem mais simples que as operações anteriores e dispensa a explicação.

Por último temos as linhas cinquenta e cinco (55) e cinquenta e seis (56) que tratam o “**default**” que vai ser executado sempre que uma opção inválida for digitada, observe a ausência do comando “**break**”, lembre-se a última estrutura da “fila” não precisa pois a estrutura acaba ali.

Ainda temos um problema a ser resolvido o caixa eletrônico só realiza uma operação e termina a aplicação, resolveremos esse problema em nossa próxima aula.