Controle de Execução: Seleção Múltipla

CONTEÚDO

Objetivos • Apresentar e estudar a estrutura de controle com múltiplas condições. • Representar a estrutura de computações com várias condições em mapas de execução.

9.1 Seleção de Várias Rotas

Uma rota pode ser dividida em vários outros trechos, cada um deles contendo uma seqüência de distintos passos de computação. Se a cada trecho for associada uma expressão-condição de modo que apenas uma delas resulte em `true` durante o percurso do mapa, a computação segue uma estrutura conhecida por *seleção múltipla*.

Como as expressões que representam condições podem produzir apenas um valor `boolean`, será necessário associar uma condição a cada trecho alternativo. Além disso, apenas uma delas deverá produzir o valor `true` quando avaliada. Este padrão estrutural de mapas é mostrado na Figura 9.1.

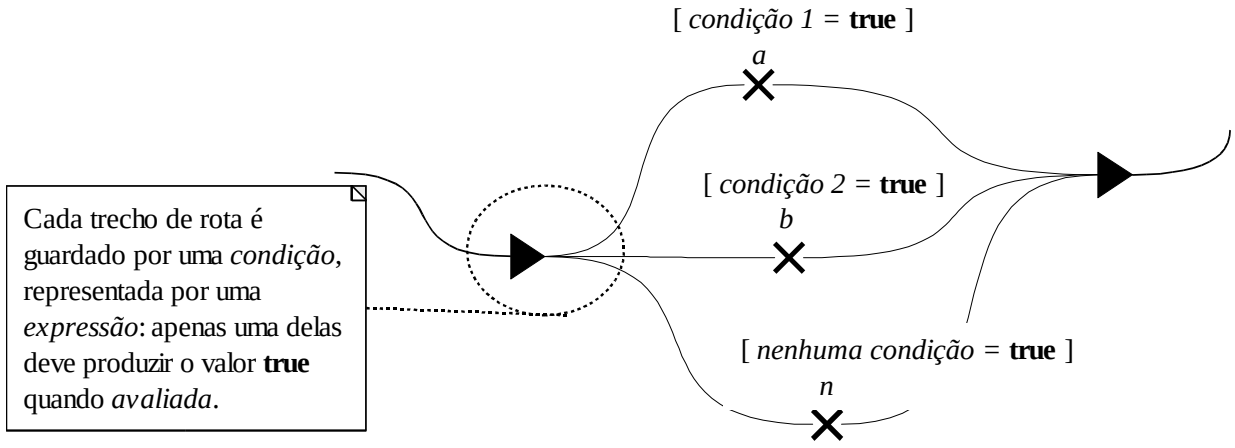


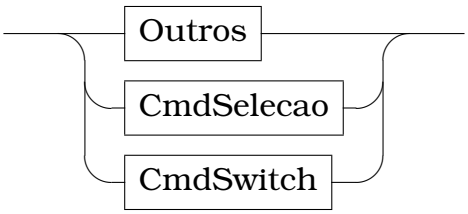
Figura 9.1: Estrutura de um mapa de execução representando a escolha de um dentre vários trechos de rotas.

As condições, neste mapa, estão associadas aos diferentes trechos de rota. As condições devem ser representadas por expressões de modo que apenas uma delas produza o valor `true` quando avaliada.

No caso do mapa da Figura 9.1, se a *condição 1* for válida, a computação prossegue pelo passo *a*. Entretanto, se for a *condição 2* for válida, então a computação prossegue pelo passo *b*. O mapa deve ser elaborado de modo que uma das expressões-condição *sempre* produza o valor `true`, e somente uma delas.

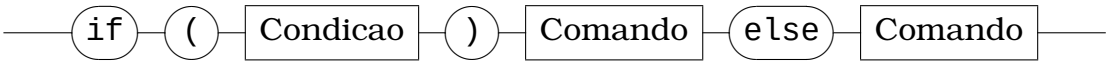
Sintaxe para Seleções Múltiplas Uma computação envolvendo seleções múltiplas pode ser descrita em termos de um encadeamento de comandos `if-else` ou em termos de um comando `switch`:

Comando



Encadeamento de `if-else` No caso de uso com encadeamento de `if-else`, o comando da parte `else` corresponde a outro comando `if-else`, como sugerido a seguir:

CmdSelecao



Tais estruturas podem ser programadas segundo o padrão:

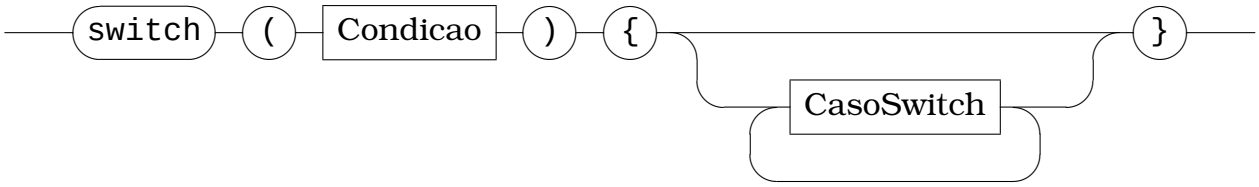
```
if( <condição 1> == true ) {
  // seqüência de passos quando
```



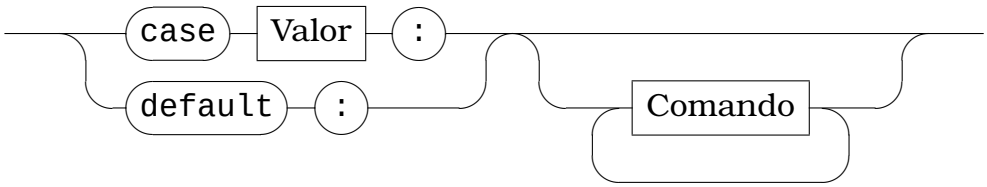
```
  // a condição 1 produzir o valor true
}
else if( <condição 2> == true ) {
  // seqüência de passos quando
  // a condição 2 produzir o valor true
}
...
else {
  // seqüência de passos quando
  // nenhuma condição produzir o valor true.
}
```

Comando `switch` O uso do comando `switch`, por outro lado, destaca cada expressão-condição em uma parte `CasoSwitch`. É comum a existência de uma diferente implementação do mapa da Figura 9.1, quando as condições podem produzir um valor diferente entre si, mas enumerável (valores do tipo `byte`, `short`, `int`, `long` ou `char`). O diagrama sintático para a descrição de computações com seleção múltipla nestes casos é:

CmdSwitch



CasoSwitch



Tais estruturas podem ser programadas segundo o padrão:

```
switch( <condição> ) {
case <valor 1>: {
  // seqüência de passos quando
  // a condição produzir o valor 1
}
case <valor 2>: {
  // seqüência de passos quando
  // a condição produzir o valor 2
}
...
default {
  // seqüência de passos para qualquer
  // outro valor produzido pela condição
}
```

Exemplo 9.1 A *fórmula quadrática* determina a solução de uma equação quadrática da forma $ax^2 + bx + c = 0$. Em função do valor dos parâmetros a, b e c, a fórmula resolve a equação para x. A forma geral desta fórmula é:

$$x = \frac{-b \pm \sqrt{b - 4ac}}{2a}$$

Uma computação baseada nesta fórmula requer a consideração de casos especiais, como aquele no qual $a = 0$.

Um mapa de passos de computação que podem ser utilizados para resolver uma equação quadrática e é apresentado na Figura 9.2.

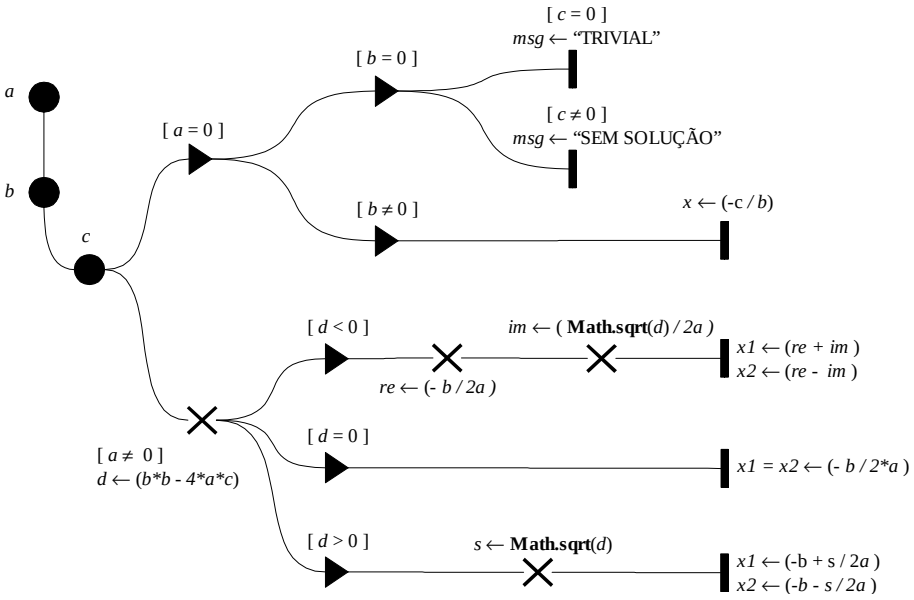


Figura 9.2: Mapa destacando os principais passos das computações que resolvem uma equação quadrática utilizando a fórmula quadrática.

O mapa revela que, inicialmente, são obtidos os coeficientes a, b e c da equação. Em seguida, dependendo da condição $a = 0$, seleciona-se a rota que resolve uma equação linear ou quadrática. Se $a = 0$, a solução pode ser trivial, inexistente ou única, com $x = \frac{-c}{b}$.

Se $a \neq 0$, trata-se de uma equação quadrática própria. O mapa destaca o passo que calcula o determinante $d = b^2 - 4ac$. Dependendo do valor de d, deve-se resolver a equação obtendo-se um par de raízes complexas ou reais (iguais ou diferentes).

Definição de Contextos Do ponto de vista de *organização da computação*, os passos deste mapa devem ser alocados em *contextos de execução*. Uma possibilidade é criar o contexto no qual são obtidos os coeficientes da equação (contexto da classe `Equacao`) e um outro contexto no qual são realizados os passos que resolvem a equação utilizando a fórmula (contexto da classe `Formula`). O redesenho do mapa com estes contextos é apresentado na Figura 9.3.

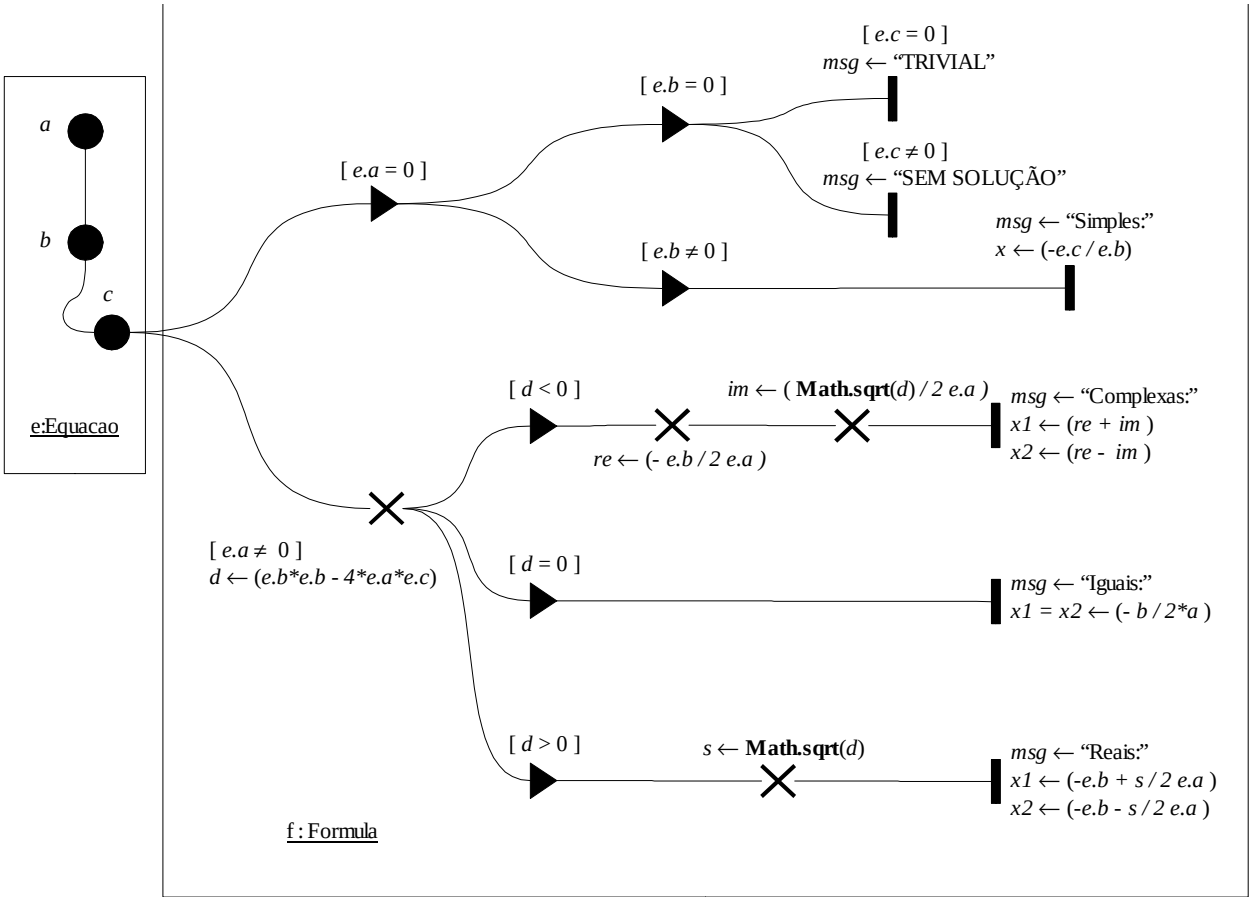


Figura 9.3: Mapa organizado por contextos de execução.

Definição de Métodos Dentro do contexto de uma `Formula`, observa-se que os passos são organizados de acordo com diversas bifurcações. Um outro nível de organização pode ser adotado pela definição de *métodos*. Um método descreve uma seqüência finita de passos de computação que devem ser realizados por um objeto durante a realização de um cálculo. A Figura 9.4 sugere uma possibiliade de organização dos passos de uma `Formula` ao redor de métodos.

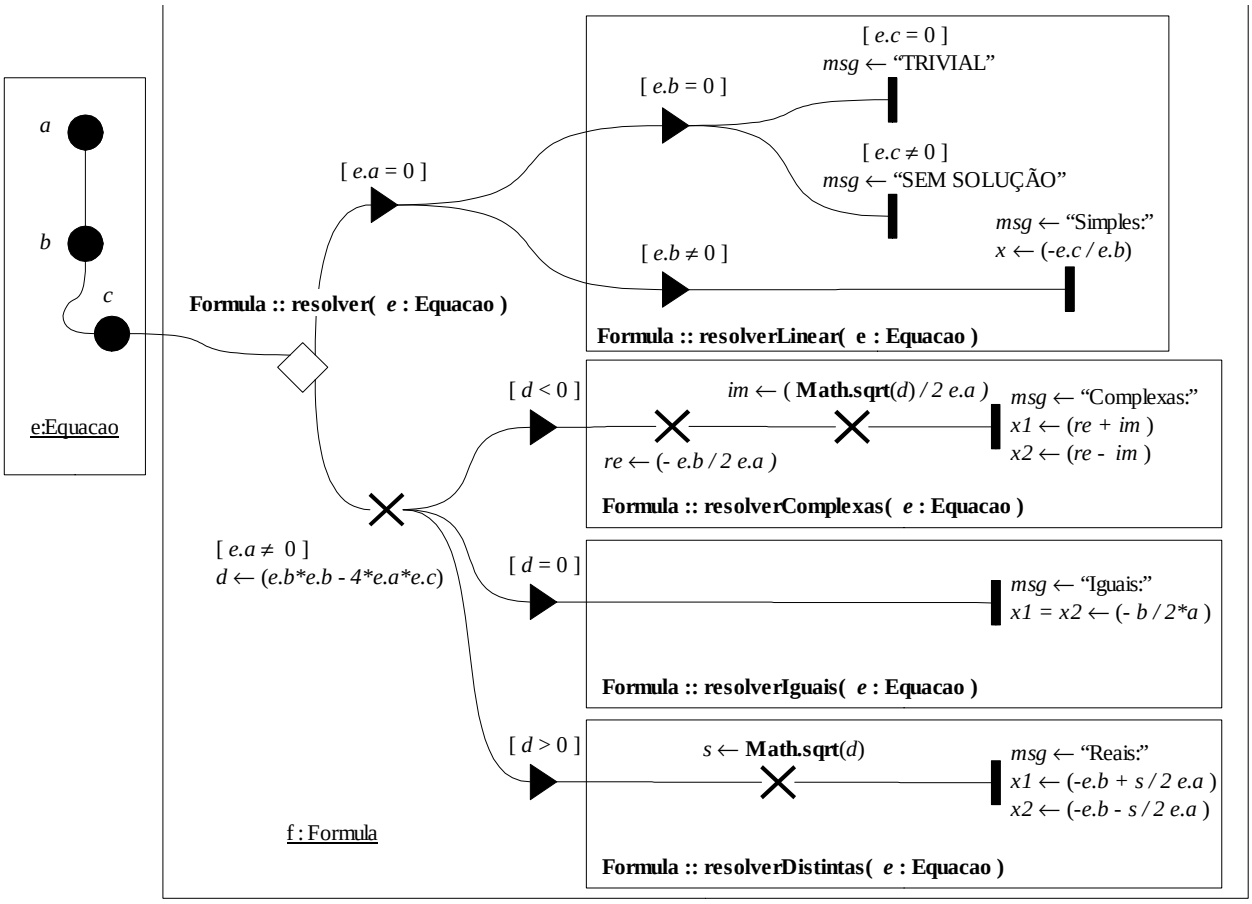


Figura 9.4: Mapa organizado por contextos de execução e métodos.

Os métodos podem ser caracterizados da seguinte maneira:

- `resolver(e:Equacao):void` – este método decide, baseado no valor do coeficiente a da equação e, se a rota envolvendo os passos `calcularLinear()` devem ser seguidos ou não. Caso $a \neq 0$, deve-se resolver uma equação quadrática própria.
- `resolverLinear(e:Equacao):void` – passos para resolver uma equação linear.
- `resolverComplexas(e:Equacao):void` – passos para resolver uma equação com soluções complexas.
- `resolverIguais(e:Equacao):void` – passos para resolver uma equação com soluções reais e iguais.
- `resolverDistintas(e:Equacao):void` – passos para resolver uma equação com soluções reais e distintas.

Classe `Equacao` Pode ser especificada por:

⟨*lab/formula/Equacao.java*⟩≡

```
public class Equacao {
   public double a;
   public void mudarA( double novo ) {
      a = novo;
   }
   public double b;
   public void mudarB( double novo ) {
      b = novo;
   }
   public double c;
   public void mudarC( double novo ) {
      c = novo;
   }
}
```

Classe `Formula` Esta classe incorpora os passos mais importantes para a solução de uma equação:

⟨*lab/formula/Formula.java*⟩≡

```
public class Formula {
   public double d;
   ⟨resolver equação⟩
   ⟨resolver equação linear com raiz simples⟩
   ⟨resolver equação quadrática com raízes complexas⟩
   ⟨resolver equação quadrática com raízes reais iguais⟩
   ⟨resolver equação quadrática com raízes reais distintas⟩
}
```

O método que resolve a equação tem a seguinte estrutura de passos:

⟨*resolver equação*⟩≡

```
public void resolver( Equacao e ) {
   if( e.a == 0 ) {
      System.out.print( "Equacao LINEAR " );
      resolverLinear( e );
   }
   else {
      System.out.print( "Equacao QUADRATICA " );
      d = e.b * e.b - 4 * e.a * e.c;
      if( d < 0 ) {
         resolverComplexas( e );
      }
      if( d == 0 ) {
         resolverIguais( e );
      }
      if( d > 0 ) {
         resolverDistintas( e );
      }
   }
   System.out.println();
}
```


A solução de uma equação linear simples é:

⟨*resolver equação linear com raiz simples*⟩≡
```
public void resolverLinear( Equacao e ) {
   if( e.b == 0 ) {
      if( e.c == 0 ) {
         System.out.println( "TRIVIAL" );
      }
      else {
         System.out.println( "SEM SOLUCAO" );
      }
   }
   else {
      double x = -e.c / e.b;
      System.out.println( "com solucao SIMPLES:" );
      System.out.println( "x = " + x );
   }
}
```

A solução de uma equação quadrática com raízes complexas deve determinar a parte real e a parte imaginária:

⟨*resolver equação quadrática com raízes complexas*⟩≡
```
public void resolverComplexas( Equacao e ) {
   double re = -e.b / (2 * e.a);
   double im = Math.sqrt( -d ) / (2 * e.a);
   System.out.println( "com solucoes COMPLEXAS:" );
   System.out.println( "x1 = " + re + " + " + im + "i");
   System.out.println( "x2 = " + re + " - " + im + "i");
}
```

A solução de uma equação quadrática com raízes reais iguais é calculada por:

⟨*resolver equação quadrática com raízes reais iguais*⟩≡
```
public void resolverIguais( Equacao e ) {
   double x = -e.b / ( 2 * e.a );
   System.out.println( "com solucoes REAIS IGUAIS:" );
   System.out.println( "x1 = x2 =" + x );
}
```

A solução de uma equação quadrática com raízes reais distintas é calculada por:

⟨*resolver equação quadrática com raízes reais distintas*⟩≡
```
public void resolverDistintas( Equacao e ) {
   double s = Math.sqrt( d );
   double x1 = (-e.b + s) / ( 2 * e.a );
   double x2 = (-e.b - s) / ( 2 * e.a );
   System.out.println( "com solucoes REAIS DISTINTAS:" );
   System.out.println( "x1 = " + x1 );
   System.out.println( "x2 = " + x2 );
}
```


Verificação Supondo que $a = 1,2345$, $b = 2,2222$ e $c = 0,4321$, a computação produz:

```
Equacao QUADRATICA com solucoes REAIS DISTINTAS:
x1 = -0.2217686875866632
x2 = -1.578312316868582
```

Agora, para $a, b = 1,2345$ e $c = 0,4321$, a computação produz:

```
Equacao QUADRATICA com solucoes COMPLEXAS:
x1 = -0.5 + 0.3162597842183088i
x2 = -0.5 - 0.3162597842183088i
```

♦

EXERCÍCIOS

9.1 SELEÇÃO DE VÁRIAS ROTAS

Tarefa 9.1.1 Crie o projeto ex9.1.

Tarefa 9.1.2 Crie a classe `Tanque` da seguinte forma:

⟨*Tanque.java*⟩≡
```
public class Tanque {
   public int nivel;
   public void mudarNivel( int novo ) {
      nivel = novo;
   }
}
```

Tarefa 9.1.3 Crie a classe `Detetor` da seguinte forma:

⟨*Detetor.java*⟩≡
```
public class Detetor {
   public Tanque t;
   public void acompanhar( Tanque novo ) {
      t = novo;
   }
   public String estado() {
      String msg = "";
      if( (5 < t.nivel)&&(t.nivel < 10) ) {
         msg = "Acima do limite";
      }
      else if( t.nivel > 10 ) {
         msg = "Muito acima do limite";
      }
      else {
         msg = "Nivel normal";
      }
   }
}
```

Tarefa 9.1.4 Crie os objetos `t:Tanque` e `d:Detetor`. Envie a mensagem `d.acompanhar(t)`.

Envie mensagens ativando a operação `mudarNivel()` de `t`, de modo a ser possível preencher a seguinte tabela:

dado	*Tela*
4	
5	
6	
9	
10	
11	

Tarefa 9.1.5 Faça um mapa de execução destacando:

- as rotas de execução,
- os pontos onde ocorrem as alterações da variável `msg`,
- os guardas dos trechos de rota.

Tarefa 9.1.6 O que acontece quando o nível do tanque é igual a 10?

9.2 MAIOR DE 5 NÚMEROS Um usuário deseja determinar o menor e o maior número de uma seqüência contendo 5 valores de temperaturas. Ele digita os números da seqüência e a computação informa a menor e a maior temperatura.

Tarefa 9.2.1 Proponha um mapa de execução descrevendo uma seqüência de passos para esta computação.

Tarefa 9.2.2 Refaça o mapa proposto acrescentando, no mínimo, o contexto `t:Termometro`.

Tarefa 9.2.3 Crie o projeto ex9.2.

Tarefa 9.2.4 Crie as classes e os objetos que implementam o mapa anteriormente elaborado.

Tarefa 9.2.5 Teste a computação com as listas:

- Lista 1 = $\{10, 8, 2, 5, 1\}$
- Lista 2 = $\{7, 232, 89, 233, 342\}$
- Lista 3 = $\{12, 8, 2, 12, 1\}$

9.3 ANO BISSEXTO Descreva uma computação em Java que produza um ano aleatório entre 1800 e 2000, indicando se se trata de bissexto ou não. Um *ano bissexto* é um inteiro maior do que 1584 divisível por 400 ou divisível por 4 mas não por 100.

(Obs.: para gerar um inteiro o intervalo $1800 \ldots 2000$, use: `int ano = Math.round( 200*x + 1800);`)

Tarefa 9.3.1 Proponha um mapa de execução onde se *garanta* uma seqüência de passos que apresente o resultado desejado.

Tarefa 9.3.2 Refaça o mapa proposto acrescentando, no mínimo, o contexto `c:Calendario`.

Tarefa 9.3.3 Crie o projeto ex9.3.

Tarefa 9.3.4 Escreva as classes que implementam o mapa anteriormente elaborado.

Tarefa 9.3.5 Adapte a computação de forma a ser possível preencher a seguinte tabela:

Ano	*Bissexto*
1492	
1592	
1600	
1700	
1776	
1992	
1999	
2000	

9.4 FÓRMULA QUADRÁTICA

Tarefa 9.4.1 Crie o projeto ex9.4 e codifique o exemplo 9.1.

Tarefa 9.4.2 Substitua eventuais encadeamentos `if-else` por comandos `switch`.

9.5 PAGAMENTO DE FUNCIONÁRIOS Uma empresa precisa calcular o pagamento total e a quantidade de horas extras dos seus funcionários. A taxa horária de remuneração é de $\$6,50$, e a hora extra é paga por todas as horas além de 40, a uma razão de $1,5$ vezes a taxa normal. A computação deve ser feita para cada funcionário, indicando-se o nome e a quantidade de horas por ele trabalhadas.

Tarefa 9.5.1 Descreva uma computação que produza os resultados desejados pela empresa: nome do funcionário, total de horas trabalhadas, total de horas extras e valor do pagamento. Use um mapa de execução para a descrição da computação.

Tarefa 9.5.2 Refaça o mapa proposto acrescentando, no mínimo, os contextos `f:Fucionário` e `e:Empresa`. O objeto `f` deverá conter a variável `nome:String`, cujo valor corresponde ao nome do funcionário, e uma outra variável `horas:int`, indicando o total de horas trabalhadas. O objeto `e` deverá ser responsável pelo passo que apresenta o nome e o valor a ser pago para o funcionário `f`.

Tarefa 9.5.3 Crie o projeto ex9.5.

Tarefa 9.5.4 Crie as classes e os objetos que implementam o mapa anteriormente elaborado.

Tarefa 9.5.5 Verifique a computação implementada nas situações:

Funcionário	*Horas*	*Extras*	*Pagamento*
`ze`	`10`	0	$\$65,00$
`ana`	`25`	0	$\$162,50$
`gil`	`40`	0	$\$260,00$
`pe`	`48`	8	$\$338,00$

9.6 DECISÃO DE COMPRA A empresa ABC cobra $\$0,75$ por rolo de fita adesiva. A empresa XYZ cobra $\$0,90$ por um rolo similar, mas oferece um desconto de $\$5,00$ para compras envolvendo mais do que 10 rolos.

Tarefa 9.6.1 Proponha um mapa de execução descrevendo uma seqüência de passos para uma computação que indique, para uma determinada quantidade de rolos de fitas adesivas, qual o fornecedor "mais barato". Caso não haja vantagem, a computação deverá gerar "INDIFERENTE".

Tarefa 9.6.2 Refaça o mapa proposto acrescentando, no mínimo, os contextos `c:Compra`, `abc:Fornecedor`, `xyz:Fornecedor` e `s:SetorCompras`. Suponha que `c` contenha a variável `q:int` cujo valor indica o total de rolos da compra; `abc` e `xyz` contenham uma variável `nome:String` indicando o *nome* do fornecedor; e que `s` seja capaz de realizar o passo que sugere um dos fornecedores (`abc` ou `xyz`) baseando-se na quantidade de rolos de `c`.

Tarefa 9.6.3 Crie o projeto ex9.6.

Tarefa 9.6.4 Crie as classes e os objetos que implementam o mapa anteriormente elaborado.

Tarefa 9.6.5 Verifique a computação implementada nas situações:

Compra	*ABC*	*XYZ*	*Sugestão*
`1`	$\$0,75$	$\$0,90$	ABC
`3`	$\$2,25$	$\$2,70$	ABC
`6`	$\$4,50$	$\$5,40$	ABC
`10`	$\$7,50$	$\$4,00$	XYZ
`20`	$\$15,00$	$\$13,00$	XYZ
`50`	$\$37,50$	$\$40,00$	ABC

Tarefa 9.6.6 Para compras acima de 10 rolos, qual a quantidade a partir da qual o fornecedor ABC oferece um preço *menor* do que o XYZ?

9.7 SITUAÇÃO DE UM ALUNO COM PROVA SUBSTITUTIVA A situação final de um aluno depende das notas obtidas nas provas *P1* e *P2*. Com base nestas notas, a sua média é calculada pela fórmula:

$$MF = \frac{P1 + P2}{2}$$

Caso o aluno falte a uma destas provas, ele faz uma prova *PS*, cujo valor substitui a nota da prova não realizada.

A situação final do aluno será "APROVADO", caso o valor de MF seja maior ou igual a 5.

Se o aluno não foi aprovado, ele faz uma prova de recuperação *PR*. A média a ser utilizada para determinar a situação do aluno neste caso é:

$$MFR = 0,3MF + 0,7PR$$

A situação do aluno, neste caso, será "APROVADO", se MFR for maior do que 5; senão, será "REPROVADO".

Tarefa 9.7.1 Proponha um mapa de execução onde se *garanta* uma seqüência de passos que apresente a situação final de um aluno.

Tarefa 9.7.2 Refaça o mapa proposto acrescentando, no mínimo, os contextos `a:Aluno` e `p;Professor`.

Tarefa 9.7.3 Escreva as classes que implementam o mapa elaborado na tarefa anterior no diretório ex9.7.

Tarefa 9.7.4 Execute a aplicação e preencha a seguinte tabela:

P1	*P2*	*PS*	*PR*	*MF*	*MFR*	*Situação*
6	4	-	-			
3	-	7	-			
-	5	5	-			
3	2	-	7			
3	2	-	4			
3	-	2	4			

