POO - Programação Orientda a Objetos - Java

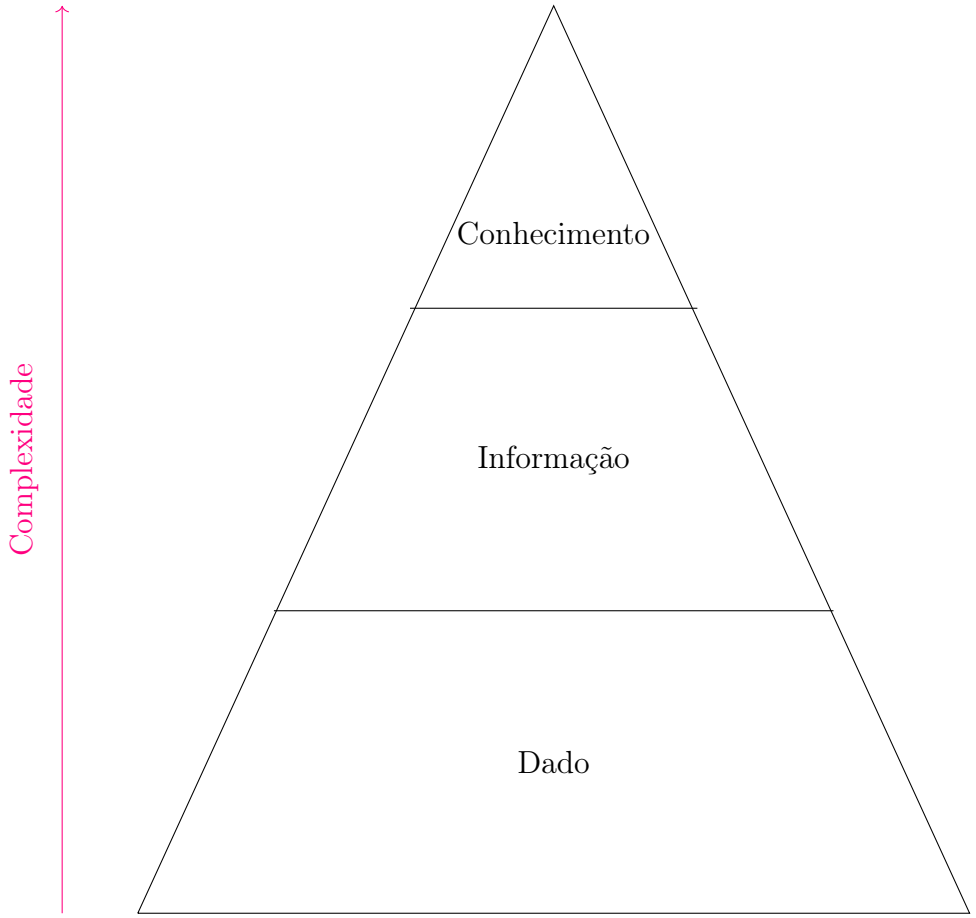
Conteúdo

1 Dado, conhecimento e informação

Dados são itens referentes a uma descrição primária de objetos, eventos, atividades e transações que são gravados, classificados e armazenados, mas não chegam a ser organizados de forma a transmitir algum significado específico.

Informação é todo conjunto de dados organizado de forma a terem sentido e valor para seu destinatário. Este interpreta o significado, tira conclusões e faz deduções a partir deles. Os dados processados por um programa aplicativo têm uso mais específico e maior valor agregado do que aqueles simplesmente recuperados de um banco de dados.

Conhecimento consiste de dados e informações organizados e processados para transmitir compreensão, experiência, aprendizado acumulado e técnica, quando se aplicam a determinado problema ou atividade. Os dados processados para extrair deduções críticas e para refletir experiência e perícia anteriores fornecem a quem os recebe *conhecimento organizacional* de alto valor potencial.



Dado: simples observações sobre o estado do mundo. Facilmente estruturado, obtido por máquina, de fácil transferência e frequentemente quantificado.

Informação: são dados dotados de relevância e propósito. Requer unidade de anáise. Exige consenso em relação ao significado e é necessária a mediação humana.

Conhecimento: resultado do processo cognitivo, iniciado por um novo estímulo qualquer. Inclui reflexão, síntese e contexto. De difícil estruturação, captura em máquina e transferência.

OBS:**Cognição:** função psicológica individual ou coletiva atuante na aquisição de conhecimento através de alguns processos: percepção, atenção, associação, memória, racio-

cínio, juízo, imaginação, pensamento e linguagem.É um conjunto de processos psicológicos usados no pensamento que realizam o reconhecimento, a organização e a compreensão das informações.

2 Sistema

Sistema é um conjunto de elementos interconectados, de modo a formar um todo organizado. (definição geral para várias áreas - bioligia, engenharias...)

2.1 Componentes do Sistema

- **Objetivos:** é a finalidade para o qual o sistema foi criado
- **Entradas do sistema:** é o que inicia o sistema, traz a informação para a operação do sistema
- **Processamento:** fenômeno que realiza as mudanças, é o mecanismo que converte as entradas em saídas
- **Saídas do sistema:** são os resultados do processamento
- **Feedback do sistema:** é a informação gerada pelo sistema que informa sobre o comportamento do mesmo
- **Ambiente:** é o meio que envolve externamente o sistema

2.2 Sistema de Informação

SI: é um conjunto organizado de dados, cujo elemento principal é a informação. Sua função principal é o armazenamento, tratamento e fornecimento de informação que de forma organizada servem de apoio a funções ou processos. Um SI não necessita, necessariamente, ser computadorizado.

Exemplos:

- biblioteca
- fichas de pacientes

Note que os dois exemplos podem ocorrer de forma computadorizada ou não. Uma biblioteca pode ter seu cadastro de livros computadorizados, facilitando as pesquisas, como também pode não ter este sistema já implementado, assim, as pesquisas (se determinado livro existe na biblioteca) ficam a cargo de um ficheiro físico.

2.3 Modelagem de sistemas

Modelar é:

- Representar de forma gráfica ou textual partes reais ou imaginárias do sistema
- Por no papel a concepção que se tem de como funcionará o sistema concebido
- Documentar de forma gráfica ou em texto um sistema existente (engenharia reversa)

Importância de modelar:

- Construção civil: planta da casa
- Desenvolvimento de sistemas: a partir dos requisitos do sistema/cliente, do perfil dos usuários e de outras informações relevantes → desenha-se o sistema em alguma linguagem padrão → aprovação do projeto → implementação

3 UML - Unified Modeling Language

UML (Linguagem de Modelagem Unificada) é uma linguagem para modelagem de softwares, ela é utilizada no processo de definição do software, na análise e estruturação de como o programa será implementado. Ela permite que os desenvolvedores de software possam modelar seus programas de forma unificada, isso quer dizer que qualquer pessoa que entenda UML poderá entender a especificação de um software. O objetivo da UML é descrever 'o que fazer', 'como fazer', 'quando fazer' e 'porque deve ser feito'.

Para modelar os sistemas, a UML possui um conjunto de diagramas, estes seguem o padrão orientado a objetos.

Diagrama de Classes: mostra o conjunto de classes com seus atributos e métodos e os relacionamentos entre classes.

Classe: "Numa série ou num conjunto, grupo ou divisão que apresenta características ou atributos semelhantes."(Ferreira, 1989)

Classes são definidas por:

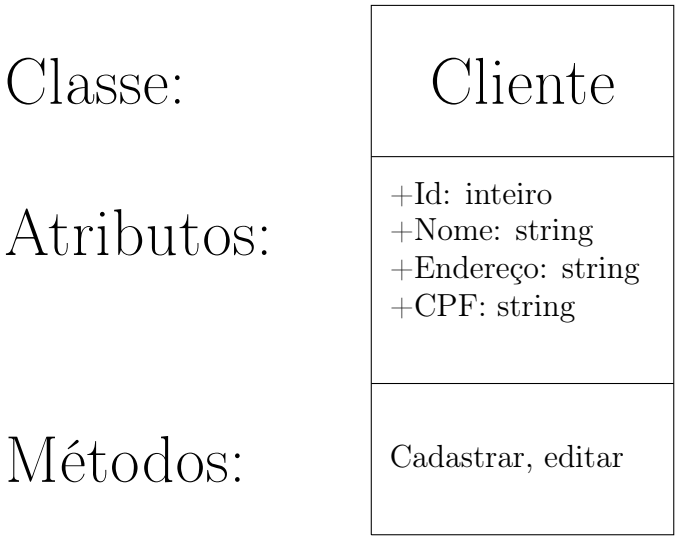
- Atributos: características que definem os objetos da classe (descritivos).
- Métodos: definem o comportamento dos objetos.

Objetos: "O que se apresenta à percepção com um caráter fixo e estável". (Ferreira, 1989)

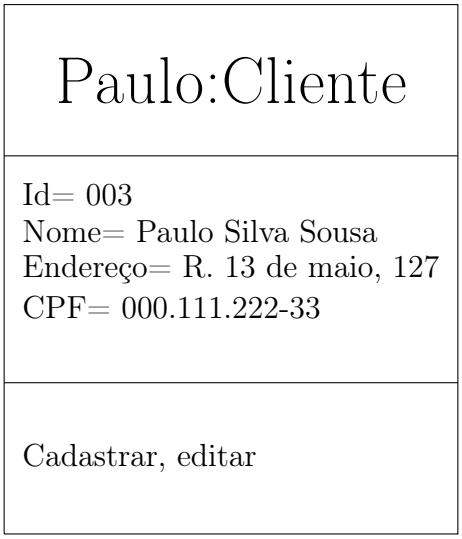
Objetos são elementos únicos dentro da classe. Podemos dizer que a classe é uma generalização de um conjunto de objetos que possuem atributos e métodos em comum.

3.1 Padrão UML

Cada classe é apresentada da seguinte forma, em um retângulo dividido horizontalmente em três partes, onde na primeira (de cima) vem o nome dado a classe, na segunda (do meio) seus atributos e na terceira (de baixo) seus métodos. Exemplo:



O objeto é uma entrada espesífica de uma classe, com valores relacionados para seus atributos. Exemplo:



Observações:

- Para outras representações (ER), o termo `Entidade` pode aparecer como 'sinônimo' de classe
- Nas definições, *string* e *varchar* também são sinônimos, a depender de qual linguagem de programação a pessoa usa como base.
- Usar *string* para CPF: '000.111.222-33' enquanto usar *inteiro/int/integer* para CPF: '00011122233'.
- Para Id o melhor é usar *integer* pois garante que '1=01=001=0001' por exemplo, com o uso de *integer* essas comparações seriam falsas.

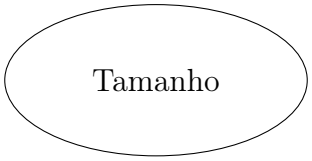
Exercício: Uma loja quer digitalizar seu cadastro de clientes e produtos.

- Crie/indique quais classes devem constar neste banco de dados.

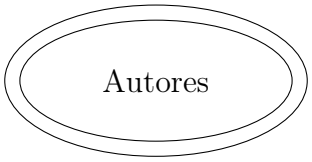
Tipos de Atributos

Obs: as representações abaixo estão no padrão ER

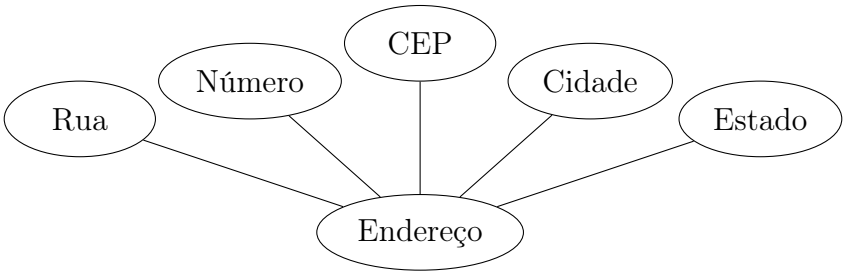
Simples (atômico): atributo que possui apenas uma entrada.



Multivalorado: atributo que pode apresentar mais de uma entrada. Ex: `autores` para uma classe livro, um livro pode ter sido escrito por mais de um autor.



Composto: 'atributo que possui atributos'. Ex: atributo `endereço` pode ser subdividido em outros atributos, como `rua, número, cidade, estado, CEP`.



Atributo-chave:

- serve para distinguir ocorrências na entrada
- é único na relação
- dessa forma, dois objetos de uma classe não terão os mesmos valores para os atributos. Os objetos serão únicos

Atributo-chave composto: é necessária a combinação de dois atributos para que se tenha essa unicidade nos objetos.

Visibilidade dos atributos: no padrão UML, no diagrama de classes, temos:

- + atributo público
- **-** atributo privado
- # atributo protegido

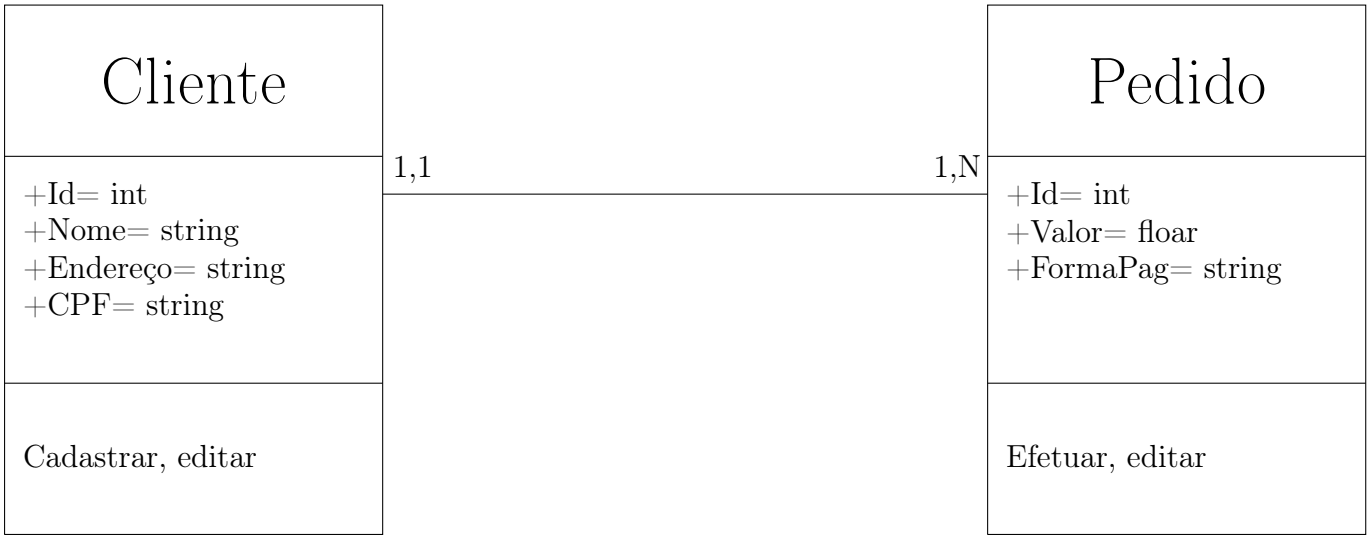
Relacionamento: é uma associação entre classes.

Por exemplo: sejam duas classes `Cliente` e `Produto`, um cliente compra produtos de uma determinada loja, então existe esse relacionamento `Compra` entre esseas duas classes

Temos a cardinalidade dos relacionamentos:

- Participação total (obrigatória): cardinalidade mínima é 1.
- Participação parcial (opcional): cardinalidade mínima é 0.
- Cardinalidade máxima pode ser 1 ou n.

Por exemplo: sejam duas classes `Cliente` e `Pedido`, um cliente faz um pedido em determinada loja, mas essa participação é parcial, pois a pessoa pode fazer o cadastro na loja (ou seja, constar em seu DB como um cliente) mas não efetivar nenhuma compra (nenhum pedido); porém, a volta tem participação total, nenhum pedido pode ser efetivado se não tiver um cliente relacionado. Já quanto a cardinalidade máxima, um cliente pode fazer mais de 1 pedido na mesma loja, porém um pedido é de apenas 1 cliente.



4 Java - primeiros passos

4.1 Estrutura mínima

As coisas em java acontecem dentro de classes. E todo programa inicia em um método de alguma classe declarado com esta assinatura:

```
public static void main (String args[])
```

A hierarquia de pacotes deve ser igual a estrutura de pastas dentro de um computador.

Obs: mostrar exemplo no terminal. Alguns comandos *bash*:

- `ls`: lista diretórios (pastas) e arquivos (documentos) que estão localizados dentro do diretório em questão
- `tree`: mostra os diretórios e arquivos em formato de árvore
- `cd ..`: retorna ao diretório anterior
- `cd`: retorna ao diretório raiz
- `clear`: limpa o terminal
- `ctrl+z`: comando `kill`, interrompe o que quer que esteja rodando

Saída de dados:

- `System.out.print(<expressão>)` : Imprime no console e mantém o cursor na mesma linha
- `System.out.println(<expressão>)` : Imprime no console e pula para a próxima linha
- **Saída de texto:** deve estar entre aspas duplas.

  Ex: System.out.println("Hello, World!");
- **Saída de variável:** deve estar fora de aspas duplas.

  Ex:

  int x=10;

  System.out.println(x);
- **Saída mista (texto e variável):** devem estar conectados pelo sinal '+'.

  Ex:

  int x=10;

  System.out.println("O valor de x é" + x);

Entrada de dados:

- `Passo Zero:` deve-se importar a classe *scanner* com o comando:

  import java.util.Scanner;
- `Primeiro passo:` deve-se abrir um scanner com o comando:

  Scanner scan = new Scanner(System.in);
- `Passos intermediários:` usa-se métodos de leitura para cada tipo de variável
  - float numF = scan.nextFloat();
  - int num1 = scan.nextInt();
  - byte byte1 = scan.nextByte();
  - long lg1 = scan.nextLong();
  - boolean b1 = scan.nextBoolean();
  - double num2 = scan.nextDouble();
  - String nome = scan.nextLine();
- `Último passo:` deve-se fechar o scanner com o comando:

  scan.close();

4.1.1 Exercícios

1. Faça um programa em java que imprime algum texto na tela. Ex: "Hello, World!"; "Calma, calabreso"; "Vamos almoçar?"

2. Faça um programa em java que imprime texto e variável (sem leitura de variável ainda).

3. Faça um programa que lê algum tipo de variável e depois a imprime. Teste para todos os tipos de variável citadas acima.

4. Faça um programa que lê dois inteiros, soma e imprime o resultado.

5. Faça um programa que lê dois inteiros, faz o produto e imprime o resultado.

6. Faça um programa que lê três números reais, soma e imprime o resultado..

7. Faça um programa que lê três números reais, faz o produto e imprime o resultado.

8. **EXTRA** Pedro viaja de carro da cidade A para a cidade B e retorna imediatamente. Na ida ele viaja a uma velocidade média de V_a e na volta ele viaja a uma velocidade média de V_b. Faça um programa que recebe dois valores reais para V_a e V_b e retorna V_m, a velocidade média total da viagem.

4.1.2 Trabalho

A ser feito em duplas.

1. Faça um programa que lê um inteiro x, soma este número com os dias dos aniversários dos participantes do grupo (por exemplo: Ana nasceu em 10 de setembro e Pedro nasceu em 18 de maio; deve-se fazer $x + 10 + 18$). Imprima: 'O resultado é: RESULTADO'.

2. Faça um programa que lê dois inteiros x e y, o primeiro deve ser multiplicado pelos valores de 1 a 12 referentes aos meses dos aniversários dos participantes e o segundo deve ser somado ao ano de nascimento dos participantes (por exemplo: Ana nasceu em 10 de setembro de 2007 e Pedro nasceu em 18 de maio de 2008; deve-se fazer x*9*5 e y+2007+2008). Imprima: 'A resposta é: RESULTADO1 e RESULTADO2'.

4.1.3 Float X BigDecimal

Usamos `float` para tratar de números reais, mas o tipo de dado `float` usa a aritmética de ponto flutuante, o que pode ocasionar erros de precisão. No exercício para somar três reais, usando o tipo de entrada de dados como `float` podem ocorrer problemas como:

```
eduarda@eduarda-HP-Pavilion-14-Notebook-PC:~/2024-Colegio/POO-2/java$ java soma3
r.java
Digite um real: 1,1
Digite um real: 1,7
Digite um real: 1,8
A soma é: 4.6000004
```

Uma possível solução para os casos em que precisão é imprescindível é utilizarmos o tipo de dado `BigDecimal`. Para tanto, devemos primeiramente importar a classe

`java.math.BigDecimal` e a soma, no caso, deve ser feita pelo método `add`. Segue o exemplo resolvido e o print do *terminal*:

```java
import java.math.BigDecimal;
import java.util.Scanner;
public class SomaBD {

  public static void main(String[] args) {
    BigDecimal x,y,z;
    //abre o scanner
    Scanner scan = new Scanner(System.in);
    System.out.print("Digite um real: ");

    // metodo le um Big Decimal
    x = scan.nextBigDecimal();
    System.out.print("Digite um real: ");

    // metodo le um Big Decimal
    y = scan.nextBigDecimal();
    System.out.print("Digite um real: ");

    // metodo le um Big Decimal
    z = scan.nextBigDecimal();


    // fecha scanner
    scan.close();
    //soma os três BigDecimal
    x = x.add(y);
    x = x.add(z);

    // imprime o numero
    System.out.println("A soma é: "+x);
  }
}
```

```
eduarda@eduarda-HP-Pavilion-14-Notebook-PC:~/2024-Colegio/POO-2/java$ java soma3
b.java
Digite um real: 1,1
Digite um real: 1,7
Digite um real: 1,8
A soma é: 4.6
```

4.2 Estrutura condicional simples e composta

4.2.1 If ou If...Else

Condições do tipo `Se` ou `Se...Então`. A linguagem Java suporta condições lógicas. Segue a tabela com a sintaxe específica para cada caso:

Caso	Sintaxe
a menor que b	$a < b$
a menor ou igual a b	$a <= b$
a maior que b	$a > b$
a maior ou igual a b	$a >= b$
a igual a b	$a == b$
a diferente de b	$a! = b$

A sintaxe do condicional é do tipo:

```java
if (condição) {
    // código a ser executado caso a condição seja verdade
    }
```

Quando usamos o `IF...ELSE`, teremos uma condição, mas duas operações distintas, uma quando a condição for verdade e a segunda quando a mesma for falsa. Segue o

exemplo:

```
if (condição) {
    // código a ser executado caso a condição seja verdade
    }
else {
    // código a ser executado caso a condição seja falsa
    }
```

A condição do `IF` deve vir entre parênteses e essa estrutura suporta mais de uma condição. Neste caso, elas devem estar ligadas por conectivos lógicos **E** ou **OU**, segue a sintaxe:

Caso	Sintaxe
E	&&
OU	||

Segue exemplo:

```
if (condição1 && condição2) {
    // código a ser executado caso a condição1 E a condição2 sejam verdades
    }
if (condição1 || condição2) {
    // código a ser executado caso a condição1 OU a condição2 sejam verdades
    }
```

Exemplos:

```
import java.util.Scanner;

public class Ifelse {

  public static void main(String[] args) {
    int x,y;
    //abre o scanner
    Scanner scan = new Scanner(System.in);
    System.out.print("Digite um inteiro: ");

    // metodo le um inteiro
    x = scan.nextInt();
    System.out.print("Digite um inteiro: ");

    // metodo le um inteiro
    y = scan.nextInt();

    // fecha scanner
    scan.close();

    if (x>y) {
        System.out.println("O primeiro é maior");
        }
    else {
        System.out.println("O segundo é maior ou são iguais");
        }

  }
}
```

```java
import java.util.Scanner;

public class Ifelse {

  public static void main(String[] args) {
    int x,y,z;
    //abre o scanner
    Scanner scan = new Scanner(System.in);
    System.out.print("Digite um inteiro: ");

    // metodo le um inteiro
    x = scan.nextInt();
    System.out.print("Digite um inteiro: ");

    // metodo le um inteiro
    y = scan.nextInt();
    System.out.print("Digite um inteiro: ");

    // metodo le um inteiro
    z = scan.nextInt();

    // fecha scanner
    scan.close();

    if (x>y && x>z) {
        System.out.println("O primeiro é maior dos três");
        }
    else if (x>y || x>z){
        System.out.println("O primeiro é maior que um dos dois apenas");
        }

   }
}
```

4.2.2 Função módulo %

Em programação, temos a função módulo (não confundir com função valor absoluto da matemática), ela nos retorna o resto da divisão por determinado número, a sintaxe é do tipo: $a = x\%y$, neste caso, a recebe o resto da divisão de x por y.

Observe que podemos usar para decidir se um número é par ou ímpar usando a função módulo:

- $x\%2 == 0$: x é par
- $x\%2 == 1$: x é ímpar

Observe que fazer $x\%10$ resulta no último dígito de determinado número. Exemplos:

- $123\%10 = 3$
- $1\%10 = 1$
- $45667\%10 = 7$
- $1234\%10 = 4$
- $123456\%10 = 6$
- $12\%10 = 2$

4.2.3 Divisão inteira

Quando declaramos nossas variáveis como inteiras, fazer a divisão de uma pela outra nos dá como resultado o menor inteiro que satisfaz, a divisão sem casas decimais. Exemplos:

- $35/6 = 5$
- $15/7 = 2$
- $7/2 = 3$
- $123/10 = 12$
- $7/5 = 1$
- $13/3 = 4$

Observe que, fazer $x/10$ (divisão inteira) 'elimina' a última casa decimal desse número. Exemplos:

- $123/10 = 12$
- $1/10 = 0$
- $45667/10 = 4566$
- $1234/10 = 123$
- $123456/10 = 12345$
- $12/10 = 1$

4.2.4 Trabalho

Faça um programa em java que lê um inteiro de 3 dígitos e retorna os dígitos de cada casa decimal. Exemplo:

- **Entrada:** 123
- **Saída:**
  - Dígito da unidade: 3
  - Dígito da dezena: 2
  - Dígito da centena: 1

Algoritmo

- Leia x
- Faça $a = x\%10$
- Faça $x = x/10$
- Faça $b = x\%10$
- Faça $c = x/10$

4.2.5 Exercícios:

1. Faça um programa que lê três números inteiros e retorna o maior entre eles.
2. Faça um programa que lê três números reais e retorna o menor entre eles.
3. Faça um programa que lê quatro números reais, caso o terceiro número seja maior que o primeiro, ele faz o produto do segundo pelo quarto e imprime: "Produto: x", onde x é o resultado; caso contrário ele faz a soma do segundo com o quarto e imprime: "Soma: x", onde x é o resultado.
4. **EXTRA** Faça um programa que lê 4 números reais (a, b, c, d) e caso $a = d$ E b seja diferente de c imprima 'Deu certo!!!'; caso contrário, se $b = c$ OU a seja diferente de d imprima 'Esse não!!'; caso contrário imprima 'Eita'. Quais casos o programa irá imprimir 'Eita'.

4.2.6 Switch Case

Quando temos várias condicionais a serem feitas, podemos usar a estrutura **Switch**, esta analisa uma expressão e suas possíveis opções de resultado, os *cases*, cada caso terá um código a ser executado, seguido de um *break;* e ao final devemos ter um caso *default*, que é quando nenhum *case* foi selecionado (não precisa de *break;*). Segue a sintaxe:

```
switch(expressão) {
  case x:
    // código a ser executado caso o valor da expressão seja x
    break;
  case y:
    // código a ser executado caso o valor da expressão seja y
    break;
  default:
    // código a ser executado caso o o valor da expressão não seja nem x nem y
}
```

Observações:

- os valores dos cases não podem ser repetidos
- os valores dos cases não podem ser variáveis
- o valor de cada case deve ser do mesmo tipo da expressão switch
- a variável passada no switch deve ser de um dos tipos aceitos pela estrutura (short, byte, long, int, enum, string)
- Comparativo com *IF...ELSE*
  - **velocidade:** dependendo da quantidade de casos que precisam ser testados no código, a velocidade de execução pode ser afetada. Por isso, se o número de casos for maior que 5, é interessante considerar utilizar a estrutura switch e não o if else
  - **valores fixos vs valores booleanos:** o if else é melhor para testar condições booleanas, já a estrutura switch é melhor para testar valores fixos
  - **legibilidade do código:** em muitos casos, o switch case ajuda a manter o código limpo, pois evita que uma quantidade muito grande de if else if sejam usados no programa
  - **expressão de teste:** outro ponto importante que deve ser analisado é a expressão que será usada como base para o teste condicional. Isso porque o switch case aceita apenas valores fixos para teste. Por outro lado, o if else também é capaz de testar faixas de valores e condições além de valores fixos

Segue um exemplo que testa se número é par ou ímpar:

```java
import java.util.Scanner;
public class SwitchTeste {

  public static void main(String[] args) {
    int num;
    //abre o scanner
    Scanner scan = new Scanner(System.in);
    System.out.print("Digite um inteiro: ");

    // metodo le um inteiro
    num = scan.nextInt();

    // fecha scanner
    scan.close();
    num = num % 2;
    switch(num) {
     case 1:
        System.out.println("O numero é impar");
        break;
     case 0:
        System.out.println("O numero é par");
        break;
     default:
        System.out.println("Programa não deveria entrar aqui");
    }
  }
}
```

4.2.7 Exercícios

1. Escreva um programa usando *switch* que lê um inteiro e testa se ele é divisível por 3.

2. Escreva um programa usando *switch* que lê um inteiro e retorna os casos:

   - Divisível por 4
   - Par, mas não divisível por 4
   - Ímpar

3. Escreva um programa usando *switch* que lê um inteiro de 1 a 12 e retorna o mês referente ao número (Janeiro = 1, Fevereiro = 2,...). Use o *Default* para caso o inteiro da entrada esteja fora desse intervalo.

4. Reescreva o programa exemplo acima (testa se é par ou ímpar) usando a estrutura *IF...ELSE*.

4.2.8 Trabalho

Trabalho em grupo, cada grupo deve resolver um dos problemas abaixo (sorteio):

1. Faça um programa em java que lê um número real com quatro casas decimais e retorna os dígitos de cada casa decimal. Exemplo:

   - Entrada: 1.1234
   - Saída:
     - dígito do décimo: 1
     - dígito do centésimo: 2
     - dígito do milésimo: 3

– dígito do décimo de milésimo: 4

2. Faça um programa em java que lê um inteiro

4.3 Estrutura de Repetição

4.3.1 FOR

4.3.2 Exercícios

1. **EXTRA** - maratona de programação (2022):

beecrowd | 3432

Interceptando Informações

Por Sociedade Brasileira de Computação (SBC), Maratona de Programação da SBC - ICPC - 2022 Brazil

Timelimit: 1

A Spies Breaching Computers (SBC), uma agência privada de espiões digitais, está desenvolvendo um novo dispositivo para interceptação de informações que, através de ondas eletromagnéticas, permite a espionagem mesmo sem contato físico com o alvo.

O dispositivo tenta coletar informações de um byte por vez, isto é, uma sequência de 8 bits onde cada um deles, naturalmente, pode ter valor 0 ou 1. Em determinadas situações, devido a interferências de outros dispositivos, a leitura não pode ser feita com sucesso. Neste caso, o dispositivo retorna o valor 9 para o bit correspondente, informando que não foi possível efetuar a leitura.

De forma a automatizar o reconhecimento das informações lidas, foi feita uma solicitação de um programa que, a partir das informações lidas pelo dispositivo, informe se todos os bits foram lidos com sucesso ou não. Sua tarefa é escrever este programa.

Entrada

A entrada consiste de uma única linha, contendo 8 números inteiros N_1, N_2, N_3, N_4, N_5, N_6, N_7 e N_8, indicando os valores lidos pelo dispositivo (N_i é 0, 1 ou 9 para $1 \leq i \leq 8$).

Saída

Imprima uma única linha contendo a letra maiúscula "**S**" caso todos os bits sejam lidos com sucesso; caso contrário imprima uma única linha contendo a letra maiúscula "**F**", correspondendo a uma falha.

Exemplos de Entrada	Exemplos de Saída
0 0 1 1 0 1 0 1	S
0 0 1 9 0 1 0 1	F