Encapsulamento, Construtor, Estrutura de Pacotes

1. Encapsulamento

O encapsulamento é um dos pilares da programação orientada a objetos. Ele ocorre quando alguns atributos são mantidos como privados e métodos públicos para permitir acessá-los são fornecidos. Isso significa manter algumas das informações da classe como privadas (ocultando os detalhes internos dessa classe) e fornecer métodos públicos para acessar essas informações, expondo somente o que é necessário para a utilização de seus objetos. O encapsulamento promove a segurança dos dados e a modularidade.

Quando todos os métodos e campos são mantidos como públicos, qualquer outra classe pode acessá-los. Tudo que essa classe faz torna-se um "livro aberto" para qualquer outra classe. O encapsulamento permite o controle dos dados que você quer compartilhar e daqueles que serão mantidos como privados, dentro de sua classe, determinando como os dados serão manipulados e protegendo-os de modificações indevidas.

Benefícios do encapsulamento:

- Segurança: Impede o acesso direto a dados sensíveis.
- Flexibilidade: Permite a utilização de métodos para adicionar regras de negócio ou validações sem expor os detalhes para outras classes.
- Manutenibilidade: Possibilita que alterações sejam realizadas sem impactar outras partes do código.

1.1. Modificadores de acesso

Para implementação do encapsulamento, é necessário compreender os modificadores de acesso e o nível de visibilidade que eles possibilitam para classes, atributos e métodos.

- **public**: permite que uma classe, atributo ou método sejam acessados diretamente, sem restrições, por qualquer outro código.
- **private**: permite o acesso somente dentro da própria classe, encapsulando os dados e garantindo que código externo à classe não tenha acesso aos atributos ou métodos.
- **protected**: permite aceso aos membros da classe (atributos ou métodos) somente no mesmo pacote, ou por classes derivadas, que utilizam o conceito de herança.
- **default** (sem modificador de acesso): permite o acesso aos membros da classe somente dentro do mesmo pacote.

Exemplo: Classe Pessoa com dois atributos: nome e idade

```
public class Pessoa {
  private String nome;
  private int idade;

  public String getNome() {
    return nome;
  }

  public void setNome(String nome) {
    this.nome = nome;
  }

  public int getIdade() {
    return idade;
  }

  public void setIdade(int idade) {
    this.idade = idade;
  }
}
```

No exemplo acima, a variável nome foi declarada utilizando-se o modificador de acesso *private*. Isso faz com que essa variável possa ser acessada somente dentro da classe Pessoa. Para permitir a leitura dos valores da variável, foi definido um método "get" (também conhecido como método acessor) chamado de *getNome()*. Observe que o método *getNome()* é público. Para permitir a modificação das informações, foi especificado um método "set" (também chamado de mutatório) chamado de *setNome()*. Esse método pode fornecer capacidades de validação das informações, assegurando que o valor da variável nome seja alterado adequadamente.

A variável idade também foi declarada como *private* e possui dois métodos públicos: *getIdade()* e *setIdade()*.

1.2. Dicas

- Para criar os métodos públicos *get* e *set*, utilize os prefixos *get* ou *set* seguidos do nome da variável declarada como *private*. Exemplo. A variável salario teria como métodos públicos *getSalario()* e *setSalario()*. Note que a primeira letra do nome da variável fica em maiúsculo.

- A especificação dos métodos set e get não é obrigatória. Eles devem ser fornecidos somente quando isso fizer sentido. Exemplo: Se você tem uma variável cujo valor nunca será alterado fora classe, somente consultado, não faz sentido especificar um método *set*.

Como testar?

```
public class TestePessoa {
  public static void main(String[] args) {
    Pessoa p = new Pessoa();

    p.setNome("José");
    p.setIdade(32);

    System.out.println("Nome: " + p.getNome());
    System.out.println("Idade: " + p.getIdade());
  }
}
```

1.3. Comparando os níveis de acesso

Modificador	Mesma Classe	Mesmo Pacote	Subclasse (mesmo pacote)	Subclasse (outro pacote)	Fora do Pacote
public	Sim	Sim	Sim	Sim	Sim
private	Sim	Não	Não	Não	Não
protected	Sim	Sim	Sim	Sim	Não
default	Sim	Sim	Sim	Não	Não

1.4. Boas práticas para utilização dos modificadores de acesso

- Encapsulamento com private: Manter os atributos como private e fornecer métodos públicos de acesso (getters e setters).
- Classes públicas e pacotes: Se a classe não precisa ser exposta fora do pacote, utilize default (sem modificador).
- Utilização de public em atributos: Evite atributos públicos, pois isso evita o encapsulamento e torna a manutenção do código mais difícil.
- Herança e protected: Para o design de hierarquia de classes, utilize protected em atributos ou métodos que devem ser acessados por sublcasses.

2. Compreendendo Métodos Estáticos

Quando você invoca um método como Math.sqrt(), por que ele funciona? Parece estranho. O método está sendo invocado diretamente da classe e não de um objeto do tipo Math. Como isso funciona?

O método sqrt() é somente um exemplo dos vários métodos que são independentes de qualquer instância específica de uma Classe. Se sqrt() fosse uma instância da classe Math, seria possível utilizá-lo somente da seguinte forma:

```
Math m = new Math();
double d = m.sqrt(42.24);
```

Em Java todos os métodos devem ser declarados dentro de uma classe. Entretanto, se você declarar um método ou uma variável como static, você pode acessá-lo utilizando o nome da classe. Nenhuma instância é requerida. Veja como o método sqrt() é declarado na classe Math:

```
class Math {
    public static double Sqrt(double d) { ... }
    ...
}
```

Tenha em mente que um método estático não é chamado em um objeto. Quando você define um método estático, ele não tem acesso a variáveis de instância declaradas na classe, somente a variáveis estáticas. Além disso, ele somente poderá invocar de forma direta outros métodos que também são estáticos. Métodos NÃO estáticos requerem a criação de um objeto.

3. Compreendendo as palavras chave class, public, static e main

A palavra chave **class** é utilizada para declarar a especificação de uma nova classe. A definição dessa classe inicia com "{" e se encerra com "}". Todos os elementos que estiverem dentro das chaves são membros da classe.

A linha `static void main()` inicia o método **main()**. É nessa linha que o programa iniciará sua execução (todas as aplicações em Java iniciam sua execução a partir de **main()**).

A palavra chave **static** permite que **main()** seja utilizado sem que um objeto do tipo da sua classe seja criado. Isso é necessário porque **main()** é chamado quando o programa começa a ser executado.

4. Casting

A conversão entre tipos de dados pode ser feita explicitamente utilizando-se um "cast". Entretanto, em alguns casos, conversões implícitas são permitidas: Exemplo:

```
int i = 10;
float f = 0;
f = i;  // Conversão Implícita. Nenhuma informação será perdida.
f = 0.5F;
i = (int)f;  // Conversão Explícita. Informações serão perdidas.
```

Um cast explícito invoca o operador de conversão de um tipo para o outro. Caso o operador de conversão definido não exista, o cast irá falhar.

Exemplo: Convertendo um double para um int. Sem o cast, o programa não irá compilar.

```
class Main {
  public static void main(String[] args) {
    double x = 1234.7;
    int a;
    a = (int)x;  // cast de "double" para "int"
    System.out.println(a);
  }
}
```

Saída: 1234

5. Números Aleatórios

O método random() da classe Math gera um valor double entre 0.0 e um valor próximo a 1.0. Entretanto, esse valor produzido pode diferir do que é necessário para uma aplicação específica. Por exemplo, um programa que simule o lançamento de uma moeda pode requerer somente “0” para cara ou “1” para coroa. Um programa que simule a rolagem de um dado de seis lados requer inteiros aleatórios no intervalo de 1 a 6.

Uma solução é utilizar o operador * em conjunto com random.

```
(int) (Math.random() * 6)
```

O código acima produz inteiros entre 0 e 5. Isso é chamado de escalonamento. Como o objetivo é gerar números entre 1 e 6, o valor 1 é adicionado ao resultado anterior.

```
1 + (int) (Math.random() * 6);
```

Exemplo: O código abaixo gera um número, aleatoriamente, maior ou igual a 1 e menor ou igual a 6.

```
int nro;
nro = 1 + (int) (Math.random() * 6);
System.out.println("Número Sorteado: " + nro);
```

6. Construtor

O construtor é um método especial. Ele possui o mesmo nome da classe e pode conter parâmetros. Entretanto, ele não pode retornar um valor (nem mesmo void). Cada classe deve conter, ao menos, um construtor (é possível conter mais do que um).

Se você não especificar um construtor, o compilador irá gerar o construtor padrão de forma automática. Ele não aceita parâmetros. Para criar um construtor, basta adicionar na classe um método público com o mesmo nome da classe, e que não retorne valor.

Quando a palavra **new** é utilizada, o construtor é chamado e ele realiza as inicializações requeridas nos objetos. Veja o exemplo a seguir:

Exemplo:

```
class Circulo {
  private double raio;

  public Circulo() {
    raio = 0.0;
  }

  public double calcularArea() {
    return 3.141592 * raio * raio;
  }
}
```

Observe que a classe Circulo possui um construtor:

```
public Circulo() {
  raio = 0.0;
}
```

Exemplo:

```
public class Carro {
  private String modelo;
  private int ano;

  // Construtor com parâmetros
  public Carro(String modelo, int ano) {
```

```
    this.modelo = modelo;
    this.ano = ano;
  }

  // Métodos getters
  public String getModelo() {
    return modelo;
  }

  public int getAno() {
    return ano;
  }
}
```

6.1. Regras importantes para o Construtor

- O construtor não possui tipo de retorno (nem mesmo void).
- É possível especificar mais de um construtor para a mesma classe, por meio da sobrecarga de construtores.
- Construtores podem chamar outros construtores utilizando a palavra reservada this().

7. Estrutura de Pacotes

Java possibilita a utilização de pacotes para organizar as classes criadas pelo desenvolvedor. A própria biblioteca padrão do Java é distribuída em pacotes, incluindo: java.lang, java.util, java.net, etc.

Além da organização das classes de seu projeto, a utilização de pacotes garante que classes com o mesmo nome não criem conflitos entre si, devido à convenção para nomenclatura dos pacotes. Imagine que três programadores trabalhando em um mesmo projeto resolvem criar uma classe chamada “Utilidades”. Sem a utilização de pacotes, não seria possível dizer à JVM qual das classes referenciar.

A *Oracle* recomenda que se utilize o nome de domínio da empresa na Internet escrito ao inverso para elaborar a estrutura dos pacotes. Como exemplo, considere um domínio na Internet cujo nome é “nomedominio.com.br”. Dessa forma, a estrutura de pacotes seria nomeada como “br.com.nomedominio”. Caso um projeto denominado “meuprojeto” fosse criado, a estrutura ficaria: “br.com.nomedominio.meuprojeto”.

Um pacote em Java é um diretório que agrupa classes relacionadas, estruturando o projeto em subpastas. A estrutura de pacote “br.com.nomedominio.meuprojeto” seria organizada na estrutura de pastas “br/com/nomedominio/meuprojeto”.

7.1. Definindo Pacotes

Para especificar o pacote ao qual uma classe pertence, a palavra chave package deve ser incluída na primeira linha do código fonte. Exemplo:

```
package br.com.edilson.exercicio;

public class Funcionario {

}
```

7.2. Importação de Classes

Uma classe pode utilizar todas as demais classes de seu próprio pacote, e também todas as classes públicas de outros pacotes. Para acessar as classes públicas de outros pacotes, existem duas maneiras: a primeira é por meio do nome completo da classe que você pretende utilizar. Exemplo:

```
package br.com.edilson.exercicio;

public class Teste {
```

```
    java.util.Scanner entrada = new java.util.Scanner(System.in);
}
```

Uma maneira mais simples é utilizar a declaração import. Por meio dela, é criado um atalho para se referir à classe de um determinado pacote. Uma vez que o import é utilizado, não há mais a necessidade de se utilizar o caminho completo da classe. Exemplo:

```
package br.com.edilson.exercicio;

import java.util.Scanner;

public class Teste {
    Scanner entrada = new Scanner(System.in);
}
```

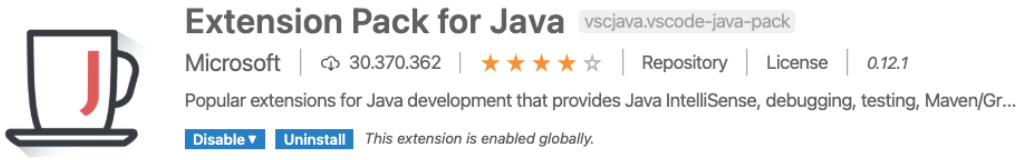
Se o objetivo é importar todas as classes de um determinado pacote, um * deve ser utilizado ao final do comando import. Exemplo:

```
import java.util.*;
```

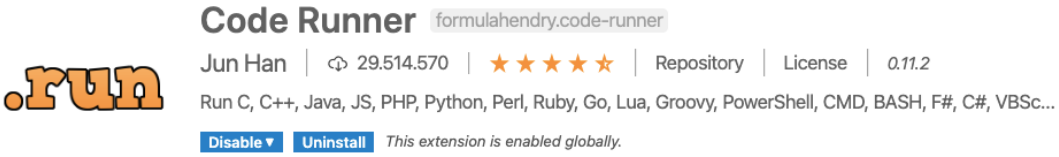
8. Configuração do Java no VS Code

Primeiro, é preciso baixar e instalar o Visual Studio Code, assim como baixar, instalar e configurar o JDK (Java Development Kit). O VS Code não possui suporte nativo para o Java. Por isso, é necessário instalar extensões específicas:

1) Abra o VS Code e no painel lateral esquerdo, clique no ícone de extensões (ou utilize o atalho Ctrl + Shift + X).
2) Pesquise por “Extension Pack for Java” e realize a instalação da extensão criada pela Microsoft e Red Hat.



3) Pesquise por “Code Runner” e realize a instalação da extensão criada por Jun Han.



O pacote “Extension Pack for Java” contêm as seguintes extensões:

- Language Support for Java™ by Red Hat: Suporte para sintaxe Java.
- Debugger for Java: Suporte para depuração de código Java.
- Java Test Runner: Ferramenta para executar testes em Java.
- Maven for Java: Suporte a projetos Maven.
- Visual Studio IntelliCode: Sugestões de código com base em inteligência artificial.

De forma opcional, você instalar a extensão “vscode-icons”, adicionando ícones personalizados para arquivos e pastas.

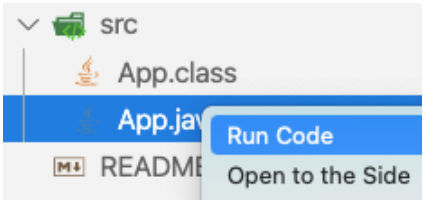
9. Como criar um projeto Java no VS Code

1) Abra o VS Code.
2) Acesse a paleta de comandos por meio do menu “View -> Command Palette” ou do atalho “Ctrl + Shift + P”.

3) Digite “Java: Create Java Project” e selecione essa opção.
4) Selecione a opção “No Build Tools” para criar um projeto Java básico, sem a configuração de Maven ou Gradle.
5) Especifique o nome do projeto.

– A pasta src é uma abreviação para “source”, que significa “fonte”. Nesta pasta serão organizados os pacotes e classes que compõem o projeto.
– A pasta lib é uma abreviação para “libraries”, que significa “bibliotecas”. Nesta pasta serão adicionadas as dependências externas (bibliotecas externas ou arquivos .jar) que o projeto possa depender.
– O arquivo README.md é um arquivo de texto em formato Markdown, que contém informações sobre o projeto, como descrição, instruções de instalação ou uso.

6) Para executar o programa, você pode clicar com o botão direito do mouse sobre o arquivo e escolher “Run Code”.



Uma alternativa é abrir o arquivo e executá-lo diretamente por meio do botão “Run Code”, incluído na parte superior direita do arquivo.

10. Sugestões de leituras

- Livro: Use a Cabeça (Capítulos 4 e 9).
- Livro: Java Como Programar (Capítulos 3 e 8).
- Livro: Java para Iniciantes (Capítulos 6 e 7).

11. Exercícios

11.1. Crie uma classe Filme. A classe deve conter os atributos titulo, ano, valor e métodos para recuperar o título do filme, o ano no qual foi lançado e também seu preço.

11.2. Crie uma classe Aluno que contenha os atributos nome, matrícula e idade. Adicione um construtor para inicializar esses atributos e métodos get e set para acesso dos dados. Organize as

classes em pacotes. Inclua uma regra de validação para que a idade do aluno não possa ser negativa.

11.3. Escreva um programa que leia as seguintes informações para dois clientes: nome, fone e crédito disponível. Você deve criar também um método para verificar se o cliente ainda tem direito a comprar na loja (baseando-se em um crédito positivo), e em caso afirmativo, o valor que pode ser gasto, de acordo com o crédito disponível.

11.4. Escreva um programa que efetue o controle da reserva dos laboratórios de informática. Estão disponíveis dois laboratórios, que devem ser cadastrados utilizando-se uma descrição para identificá-los. O processo de reserva funciona da seguinte forma: É possível solicitar a reserva de um laboratório somente para um dos dias da semana (segunda a sábado) em questão e a reserva é válida para o dia todo. Toda reserva deve conter o nome do responsável pelo laboratório naquele período. Ao liberar os laboratórios para novas reservas, um relatório deve exibir a porcentagem de utilização de cada um dos laboratórios durante a semana.

11.5. Foi realizada uma pesquisa de audiência para canais de TV em várias casas de uma cidade, num determinado dia. Para cada casa consultada foi fornecido o nome do canal (por exemplo, Globo, SBT, Band, Record) e o número de pessoas que estavam assistindo àquele canal. Se a televisão estivesse desligada, nada era anotado, ou seja, essa casa não era considerada para a pesquisa. Faça um programa que:

a) Leia um número indeterminado de dados (nome do canal e o número de pessoas que estavam assistindo);
b) Calcule e mostre a percentagem de audiência de cada canal.

11.6. Uma empresa de transporte possui ônibus com 8 lugares (4 nas janelas e 4 no corredor). Faça um programa que utilize dois vetores para controlar as poltronas ocupadas no corredor e na janela. Considere que 0 (zero) representa poltrona desocupada e 1 (um) representa poltrona ocupada.

O programa deve controlar a venda de passagens da seguinte maneira:

- O cliente informa se deseja poltrona no corredor ou na janela e o programa deve informar quais poltronas estão disponíveis para a venda;
- Quando não existirem poltronas livres no corredor, nas janelas ou, ainda, quando o veículo estiver completamente cheio, deve ser exibida uma mensagem.