# Convenções e Conceitos

- Nome de classe: sempre maiúsculo

- Nome de métodos: sempre minúsculo

- NÃO é orientado a identação, mas é sempre bom manter a organização

- Aspas duplas é String e simples é char

**Classes**

- As classes são organizadas em pacotes

Ex: A classe Scanner existe dentro do pacote java.util

*Por padrão, a única classe que é automaticamente importada em todo arquivo Java é java.lang(contém classes fundamentais como String, System, Integer, etc)*

**Criar pacotes e classe**

- Criar packeges: clica com o botão direito no arquivo e escolha new java packege

- Criar classe: clica com o botão direito no arquivo e vai em New Java File – Class

- Bloco de comentário: /\* e \*/

- Comentário simples: //

**Importante:**

println 🡪 imprime o valor com \n ou seja pula pra linha de baixo

print 🡪 só imprime o valor

printf 🡪 É com o format

O Scanner em Java é uma classe fundamental usada para **obter entrada de dados de várias fontes**, principalmente do teclado (entrada padrão, System.in) ou de arquivos. Usa delimitador (por padrão é o espaço em branco)

**Explicando a linha principal**

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

* O java sempre procura esse método para começar a executar o programa

**static**: Quando um método ou variável é static, ele pertence a classe em si e não a uma instancia (objeto) específica da classe

**Por que main é static?** Porque a JVM precisa chamar main para iniciar seu programa sem ter que criar um objeto da sua classe primeiro.

**void:** é uma palavra-chave que indica que um método não retorna nenhum valor

**Convenção**

private final static String WELCOME\_MESSAGE = "Olá mundo"; *//por ser privado nenhuma outra classe pode acessar, só essa. o final é como uma constante, o valor não pode ser alterado*

**Importante:** Por convenção em Java, constantes (final static) são nomeadas em **letras maiúsculas** com sublinhados para separar palavras (ex: WELCOME\_MESSAGE, MAX\_VALUE

# Keywords

- Inteiro: byte, short, int, long

- Ponto Flutuante: float, double

- Caractere: char

- Booleano: Boolean

# Operadores

**Lógicos e Atribuição**

|| 🡪 Ou

&& 🡪 E

!= 🡪 Diferente de

! 🡪 Not (vai ser ao contrário)

**Sobre operadores Aritméticos:**

🚨No Java se tentar dividir dois números inteiros, ele irá retornar um inteiro, mesmo que o resultado seja decimal. Por isso, é importante receber a entrada como Float

- O java também aceita valor+= ou valor-=

- Para incrementar ++valor **importante:** se declarar o sinal a esquerda valor-- ele só decrementa na próxima linha, se for a direita --valor ele já retorna na mesma linha

Operadores Bitwise (Bit-a-Bit)

* São usados para manipular e testar bits individuais de números inteiros
* Trabalham com a representação binária do número
* Pouco usado
* Mais usado em sistema embarcado

- **xor** 🡪 é representado por ^. Quando os números são iguais ele retorna 0 e se for diferente retorna 1

- **complement** 🡪 é representado por ~. Ele funciona como o NOT, ou seja, inverte todos os valores

- **left shift operator** 🡪 é representado por <<. Desloca o número para esquerda e preenche espaço vazio com 0.

Ex: 9 tem o binário 1001

Se deslocar duas posições então ele completará: 100100

9 << 2

- **rigth shift operator** 🡪 é representado por >>.Funciona da mesma forma que o left, só que desloca para a direita e ao completar o número ele verifica, se for negativo (ex: -8) ele completa com **1**, se for positivo ele completa com **0**

Ex: -8 tem binário 111111111111111111111000

Se deslocar duas posições ele completará 11111111111111111111111110 porque é negativo

-8 >> 2

Agora, 8 tem binário 1000

Se deslocar duas posições ele completará 0010 restando só 10

8 >> 2

**- Unsigned rigth shift operator** 🡪 é representado por >>>. Igual ao rigth, mas ele sempre completa com 0, independente se for negativo ou positivo

# Estrutura Condicional

**Operador Ternário** 🡪 ? :

- Funciona em uma linha fazendo atribuição, para maiores códigos de lógica é melhor usar o bloco if / else

var canDrive = (age >= 18) || (age >= 16 && isEmancipated);

        System.out.println(canDrive ?

                name + ", você pode dirigir" :

                name + ", você não pode dirigir");

**Importante:** Em java não podemos atribuir sinal de igual = entre Strings, deve usar .equals e IgnoreCase se quiser levar em conta maiúsculas e minúsculas

**Estrutura de repetição do e while:**

- **While:** primeiro ele verifica se a condição já foi atendida, depois ele executa;

Então se primeiro eu passar

var nome = “exit”

e depois no while fizer

while (!nome.equalsIgnoreCase(“exit”))

NÃO vai funcionar, pois a condição já foi atendida antes do while executar

*obs: pode funcionar com o while true e depois passar a condição no if*

**- Do:** primeiro ele executa e depois verifica, então nesse exemplo acima ele VAI funcionar

var nome = “exit”;

do{

System.out.println(“informe um nome”)

if (name.equalsIgnoreCase("exit")) break;

}

# Orientação a objetos

- Variáveis estáticas NÃO são acessíveis pelo this (pois pertencem a classe e não ao objeto)

- Se quiser que um valor não seja alterado, deve passa-lo no construtor e atribuir final

- Ao trabalhar com setter usamos this

**Records**

- São classes imutáveis

- só é possível declarar atributos estáticos

- tem métodos (getters automáticos)

- Adicionamos os atributos no construtor

- Todo atributo declarado no record é privado

**Para estruturar a classe:**

Getter:

- public tipo getAlgumaCoisa()

Métodos e setter:

- public void addWater()

- public void setClean(boolean clean)

Duvida: quando usar float e quando usar double

Pq a petmachine não tem construtor

**Importante:** para conseguir acessar um método sem criar um objeto ele deve ser estático

**Em Java, o uso de** this **é opcional** **quando não há ambiguidade entre variáveis.**

**Não** há ambiguidade:

private int speed;

public void accelerate() {

speed += 1; // aqui "speed" já se refere ao atributo da classe

}

Quando há ambiguidade:

public void setSpeed(int speed) {

this.speed = speed; // agora é necessário usar "this" pra diferenciar

}

**Importante:** O return; (sem valor) é usado em métodos void pra **encerrar a execução do método ali**. Isso evita que o resto do método continue.

* quando o if é só um aviso mas o resto do método **ainda pode continuar**, não precisa usar return.

**Overloading**: São duas versões do mesmo método

# ATENÇÃO!!

**Else if e mais de um if seguido**

**else if:** só é testado se o anterior for falso

**mais de um if:** todos são testados. É ótimo para quando mais de uma sentença é verdadeira