Curso Danki Code – **Modulo de Desenvolvimento de GAMES em Java**Software: **Eclipse IDE**Por Victor Gabriel F. Theodoro

**Conceitos Básicos Para O Funcionamento do Código:**

public static void main(String[] args) {

System.***out***.println("Hello Word!");

}

**Explicação:**

**Public**: Este é um **modificador de acesso** que indica que o método ‘main’ é acessível de qualquer lugar

**Static**: Indica que o método ‘main’ **pertence á classe em si e não a uma instância especifica dessa classe**. Isso significa que você pode chamar o método ‘main’ sem criar um objeto da classe que o contém

**Void**: Esse é o **tipo de retorno do método** ‘main’. O termo “Void” significa que o método não retorna nenhum valor

**Main**: Este é o nome do método. O método ‘**main**’ é o **ponto de entrada a execução do programa em Jav**a. Quando você executa um programa em Java, o sistema **procura por esse método e começa a execução a partir dele**

**(String[] Args)**: O método ‘main’ aceita um argumento, que é uma matriz de strings chamada ‘args’. **Esta é uma convecção em Java para passar argumentos da linda de comando para o programa**. No entanto, neste exemplo, o argumento não está sendo utilizado

**System.out.println**: Esta linha **imprime a mensagem determinada no console**. ‘System.out’ é um **objeto que representa a saída padrão** (normalmente o console). O método ‘**println**’ é usado para **imprimir uma linha de texto**, neste caso, imprime a Sting “Hello Word!” seguida por uma quebra de linha

**Variáveis:**  
**- int:** Armazena valores inteiros de 32 bits, variando de -2^31 a (2^31 - 1). **float:** Armazena valores de ponto flutuante de 32 bits, representando números de  
ponto flutuante. **- double**: Armazena valores de ponto flutuante de 64 bits, representando números de ponto flutuante. **char:** Armazena caracteres Unicode de 16 bits. **- boolean:** Armazena valores **verdadeiro** ou **falso**. (True or False) **Classe:** Tipo de referência que pode ser usado para criar objetos.  
**Interface:** Tipo de referência que define um conjunto de métodos que uma classe deve implementar.  
**- Array:** Um tipo especial de referência que representa uma coleção de elementos do mesmo tipo, acessíveis por um índice.  
**- String:** Uma sequência de caracteres em Java, é imutável.

**Exemplo de declaração de variáveis e seus valores e os utilizando no console:**

public static void main(String[] args) {

String nome = "Victor"; //Palavras e Letras

int idade = 17; //Números Inteiros

boolean tipo = true; //Verdadeiro ou Falso

double peso = 80.5;

System.***out***.println("Hello "+nome);

System.***out***.println(idade+" Years Old");

System.***out***.println(nome+" weighs "+peso);

}

**Resultado no Console:**Hello Victor

17 Years Old

Victor weighs 80.5

**Condições**:  
Estruturas Condicionais são **usadas para tomar decisões com base em condições** booleanas. As principais estruturas condicionais são **‘if’, ‘else if’, ‘else’, e ‘switch case’**

**Exemplos de como são utilizadas as condições:**

**IF: Else If:  
Texto

Descrição gerada automaticamente Texto

Descrição gerada automaticamente**

**Switch Case:**

**Texto

Descrição gerada automaticamente**

**Operadores:**

**Operadores Aritméticos:**

+ Adição.  
- Subtração.  
\* Multiplicação.  
/ Divisão.  
% Resto da divisão (módulo).

**Operadores Relacionais:**

== Igual a.  
!= Diferente de.  
< Menor que.  
> Maior que.  
<= Menor ou igual a.  
>= Maior ou igual a.

**Operadores Lógicos:**

&& (**E**): Retorna verdadeiro se **ambas as expressões forem verdadeiras**.  
|| (**OU**): Retorna verdadeiro **se pelo menos uma expressão for verdadeira**.  
! (**NÃO**): **Inverte o valor booleano da expressão**.

**Operadores de Atribuição:** (são utilizados para atribuir valores a variáveis em programação)

= Atribuição simples.  
+= Adição e atribuição.  
-= Subtração e atribuição.  
\*= Multiplicação e atribuição.  
/= Divisão e atribuição.  
%= Módulo e atribuição.

**Operadores de Incremento/Decremento:**

++ **Incremento** (aumenta o valor por 1).  
-- **Decremento** (diminui o valor por 1).

**Operador Ternário:**

? : Operador ternário, **uma forma concisa de expressar uma instrução if-else**.

**== Igualdade Solta:** Realiza uma **comparação de igualdade**, ele **converte os operandos**, se necessário, para um tipo comum antes de fazer a comparação. Exemplo:  


**=== Igualdade Estreita:** Realiza uma **comparação de igualdade**, porém, **Compara os** **valores e os tipos sem fazer conversões**. Exemplo:  


**Laços de Repetição:**

laços de repetição, também conhecidos como **estruturas de controle de fluxo iterativo**, são utilizados para **executar  
um bloco de código várias vezes enquanto uma condição específica é atendida**. Existem três tipos principais de laços de repetição em Java: **for, while e do-while**.

**Exemplos de como são utilizadas:**

**For: While:**

Texto

Descrição gerada automaticamente Texto

Descrição gerada automaticamente

**Do-While:**

Texto

Descrição gerada automaticamente

**Controle de Fluxo em Laços:**

**‘break’** Termina imediatamente o laço. **‘continue’** Pula para a próxima interação do laço.

Texto

Descrição gerada automaticamente Texto

Descrição gerada automaticamente

**Classes:**

Uma classe em Java é uma estrutura fundamental que **define um tipo de objeto**. Ela serve como um **modelo para criar instâncias (objetos)** desse tipo. Uma classe **encapsula dados (atributos)** e **comportamentos (métodos)** **relacionados a um conceito específico**. Os objetos criados a partir de uma **classe compartilham as mesmas características**, mas cada instância pode ter valores diferentes para seus atributos. **Exemplo:**

Texto

Descrição gerada automaticamente

**Tipos de Classes:**

**Object**:  
- **Classe base de todas as classes em Java**.  
- Fornece métodos como **equals(), hashCode(), e toString().**  
- Muitas classes em Java herdam ou sobrescrevem métodos desta classe.

**String**:  
- Representa uma **sequência imutável de caracteres.**  
- Oferece métodos para manipulação de strings, **como charAt(), length(), substring(), etc.**

**Scanner**:  
- **Facilita a leitura de entrada do usuário a partir do console**.  
- Oferece métodos como **next(), nextInt(), nextLine(), etc.**

**ArrayList**:  
- **Implementação da interface List** que usa uma matriz redimensionável para armazenar elementos.  
- Fornece métodos como **add(), get(), size(), etc.**

**Math**:  
- Contém **métodos para operações matemáticas**.  
- Inclui métodos como **sqrt(), pow(), random(), etc.**

**----------------------------------------------------------------------------------------**

**Public:**- public é um modificador de acesso que indica que uma classe, método, ou variável **é acessível por qualquer classe**.  
- No contexto de uma classe, uma classe marcada como public **pode ser acessada de qualquer outro lugar no programa**.

**Private**:  
- private é um modificador de acesso que indica que uma classe, método ou variável **é acessível apenas dentro da própria classe**.  
- Uma classe private **não pode ser acessada diretamente por outras classes**.

**Protected:**Um membro (método ou variável) marcado como protected é acessível **dentro da mesma classe, por subclasses (mesmo que estejam em pacotes diferentes) e por outras classes no mesmo pacote**.

**Default:**Um membro **sem modificador de acesso (às vezes chamado de "default" ou "pacote-private") é acessível apenas dentro do mesmo pacote.**

**Static**:  
- static é usado para **criar membros de classe (métodos e variáveis) que pertencem à classe em vez de instâncias específicas da classe.  
-** Membros estáticos podem ser acessados **diretamente pela classe, sem criar uma instância da classe**.

**Métodos:**

Um método em Java é **um bloco de código que executa uma ação específica**. Métodos são definidos dentro de uma classe e **representam o comportamento dessa classe**. Eles podem receber parâmetros, realizar cálculos, e retornar valores. Métodos são chamados em objetos ou diretamente na classe, dependendo de serem estáticos ou de instância.  
**Exemplo do uso de Métodos:**

**Texto

Descrição gerada automaticamente**

**Tipos de retorno:**

**void:**

Indica que o método **não retorna nenhum valor.**  
Exemplo:

Texto

Descrição gerada automaticamente

**Tipos Primitivos:**  
Métodos podem retornar tipos primitivos, como **int, double, char, boolean, etc.**Exemplo:

Texto

Descrição gerada automaticamente

**Objetos ou Tipos de Referência:**  
Métodos podem retornar **objetos de classes específicas**.  
Exemplo:

Texto

Descrição gerada automaticamente

**Arrays:**  
Métodos podem retornar **arrays de qualquer tipo**.  
Exemplo:

Texto

Descrição gerada automaticamente

**Void vs Outros Tipos:**  
- Se um método tiver **void** como tipo de retorno, **não precisa incluir uma instrução return**, ou pode usar return; para encerrar o método.  
- Métodos com outros tipos de retorno **precisam usar return para fornecer um valor correspondente ao tipo de retorno.**

**Heranças:**

Herança em Java é um conceito fundamental da **programação orientada a objetos**, permitindo que **uma classe herde atributos e métodos de outra classe**. Aqui está uma explicação resumida dos principais pontos:

**Superclasse (ou classe pai):**  
É a classe que **contém os atributos e métodos que podem ser herdados**.  
Também é chamada de classe base ou classe mãe.

**Subclasse (ou classe filha):**É a classe que **herda os atributos e métodos de uma superclasse**.  
Pode ter seus próprios atributos e métodos, além dos herdados.

**Palavra-chave extends:**Utilizada para **estabelecer uma relação de herança entre classes**.  
A subclasse declara extends seguido pelo nome da superclasse.

**Texto

Descrição gerada automaticamente**

**PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)**

- Melhor Organização do Programa;  
- Lógica mais “fácil” de se trabalhar;  
- Uso de classes, com métodos e atributos;  
- Profissionalismo;  
- Facilidade na manutenção do código.

Um **objeto** seria a mesma coisa que uma **classe**. Por exemplo:

- Criamos um Objeto/Classe – Inimigo (Supondo que estamos fazendo um jogo)  
- Dentro dessa classe iriamos ter: Vida, Velocidade  
- Com os métodos: adicionarVida(), perderVida()

Com isso, depois precisamos apenas **puxar esses métodos criados dentro dos objetos para a parte principal do código** onde serão utilizadas. Como havia dito antes, isso seria o conceito de herança.

**Atributos:**  
São como **variáveis que definem as características** de uma classe.  
Por exemplo, em uma classe ‘Carro’, podemos ter atributos como ‘modelo’, ’ano’, ’cor’, etc.  
Exemplo:

Texto

Descrição gerada automaticamente

**Método Construtor:**  
É um método especial **usado para inicializar objetos**. Ele tem o mesmo nome da classe e é chamado automaticamente quando um objeto é criado. O construtor **pode ser utilizado para definir valores iniciar para os atributos do objeto**.  
Exemplo:

Texto

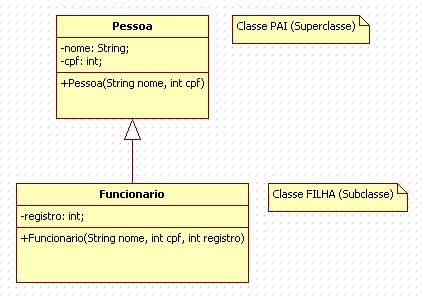
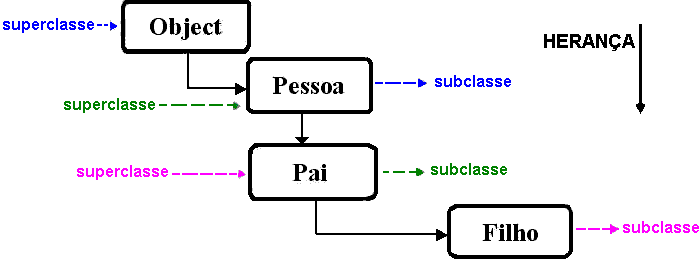
Descrição gerada automaticamente

**Método ‘Super’:**  
O método ‘super‘ é utilizado para **chamar o construtor da classe pai(superclasse) dentro de uma subclasse**. Isso é útil quando queremos **estender o comportamento do construtor da superclasse na subclasse**.

Por exemplo, se temos uma classe ‘Animal’ com um construtor, e uma subclasse ‘Cachorro’ que estende ‘Animal’, podemos chamar o construtor de ‘Animal’ dentro do construtor de ‘Cachorro’ usando ‘super()’

Texto

Descrição gerada automaticamente



**Instanceof:**

O operador ‘instanceof’ é utilizado para **verificar se um objeto é de uma instancia de uma determinada classe ou de uma classe que implementa uma interface**. Ele retorna ‘true’ se o objeto for **uma instância da classe especificada** ou de uma das suas subclasses, e ‘false’ caso contrário. Isso permite **verificar o tipo do objeto antes de realizar determinadas operações com ele**.

Exemplo:

Texto

Descrição gerada automaticamente

Neste exemplo, estamos verificando se obj é uma instância da classe String. Como obj contém uma instância de String, o resultado da verificação será true, então a mensagem "O objeto é uma instância de String." será impressa.

**Package:**Packages são iguais a **pastas que utilizamos para organizar todas nossas classes e códigos**.  
Por padrão na **Industria profissional de desenvolvimento em Java**, nomeamos os Packages da seguinte forma: **(domínio).(nome-da-empresa).(tipo-de-classe)**Por exemplo: **com.theodoro.entidades**Texto

Descrição gerada automaticamente

**Sintaxe básica para exibir gráficos na tela:**

public class Game extends Canvas implements Runnable{ //criando o metodo Canvas para criar gráficos.

public static JFrame *frame*; //chamando o método JFrame.

private final int WIDHT = 10;

private final int HEIGHT = 120;

private final int SCALE = 3;

//definindo o tamanho da tela.

public Game() {

this.setPreferredSize(new Dimension(WIDHT\*SCALE,HEIGHT\*SCALE));

//definindo do Game os tamanhos pré-definidos anteriormente.

*frame* = new JFrame();

*frame*.add(this); //chamando o tamanho definido

*frame*.setResizable(false); //impedindo que o usuario possa redimensionar a janela.

*frame*.pack(); //calcular as dimensões e exibi-las na tela.

*frame*.setLocationRelativeTo(null); //centralizando a janela.

*frame*.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

//define que quando clicar para fechar a janela ele de fato feche.

*frame*.setVisible(true); //para deixar a janela visivel

}

public static void main(String args[]) {

Game game = new Game(); //colocando o obj game no método principal

}

**BUFFER:**

É uma **área de armazenamento temporário usada para guardar dados antes de serem processados ou exibido**s. Em computação gráfica, é usado para **armazenar pixels de imagens antes de serem mostrados** na tela.

Ele é usado para **manipulação eficiente de gráficos e para garantir uma renderização suave** e precisa das imagens na tela, **as otimizando**.