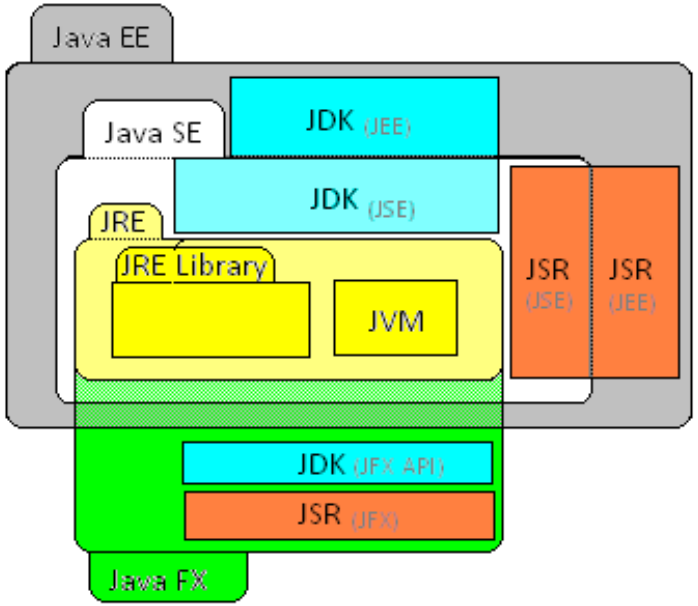
AULA 04 - REVISÃO JAVA

Disciplina de Backend - Professor Ramon Venson - SATC 2024

Plataforma Java

- Desenvolvida pela Sun até 2010
- Propriedade da Oracle Corporation
- Conjunto de softwares e padrões
  - Linguagem
  - Development Kits
  - Máquina Virtual



Linguagem Java

- Criada por James Gosling (1991)
- Baseada em classes
- Orientada a Objetos
- Compilada para byte-code usando JAVAC

Java Virtual Machine (JVM)

- Código fonte aberto em 2007
- Roda programas compilados
- Ambiente completo é chamado de Java Runtime Environment
  - Inclui bibliotecas e a JVM
- Especificação com várias implementações
  - Hotspot
  - GraalVM

Java Development Kit (JKD)

- Compilador, JVM, Debuggers, Docs, Base Packages

Gerenciamento de Memoria

- Garbage Collector deleta memória não **referenciada**
- Processo automático
- Libera o programador de alocar e desalocar recursos da memória

```
Object meuObjeto = new Object(); // novo objeto é alocado na memoria
meuObjeto = null; // Objeto desalocado da memória e aguarda deleção
```

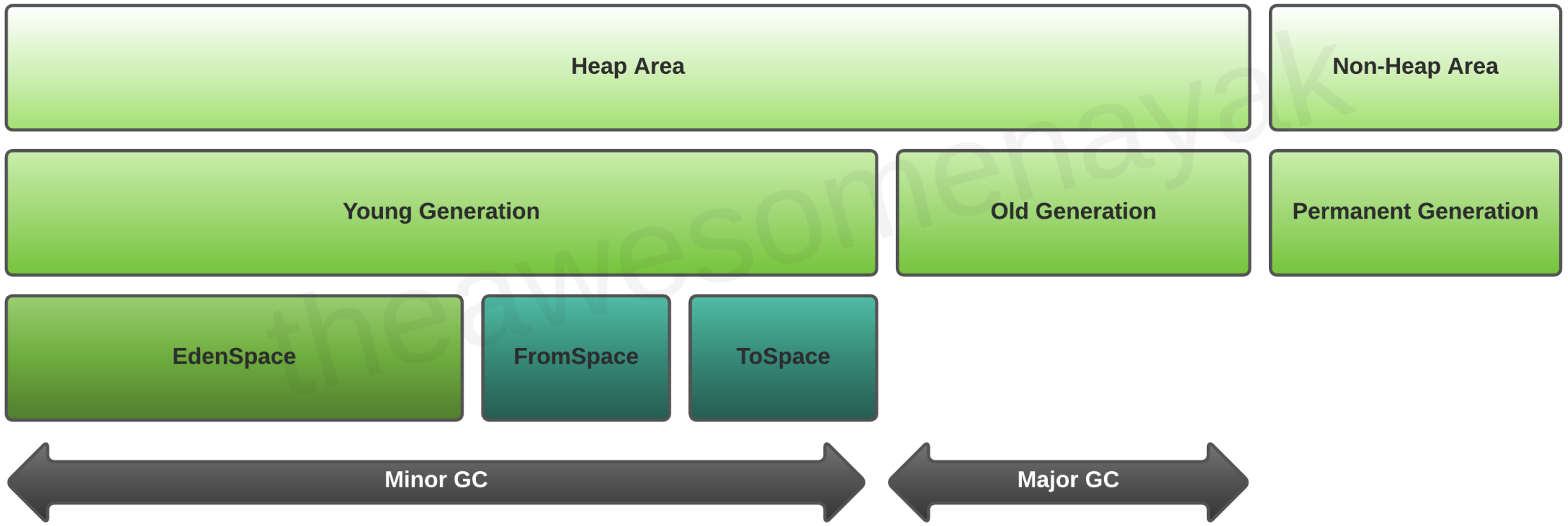
Como seria na linguagem `C++`:

```
int *ptr; // Inicia um ponteiro
ptr = (int*) malloc(sizeof(int)); // aloca um ponteiro de memória do tamanho de um inteiro
free(ptr1); // libera area de memória
```

Cada JVM implementa o Garbage Collector de acordo com a especificação. Essa especificação tem como ciclo base 3 etapas:

- Marcação de Objetos Vivos
- Deleção de Objetos Mortos
- Compactação do restante da memória

É possível escolher diferentes estratégias de GC usando a algumas flags na execução do programa (ex.: `-XX:+UseParallelGC`)



Tipos de dados (Datatypes)

- Primitivos
  - `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` e `char`
- Não-primitivos
  - `Object`

Imports e Packages

Packages são usados no Java para organizar diferentes conjuntos de classes. Cada classe possui seu package definido logo no início do documento.

```
// package [nome_do_pacote]
package com.organizacao.modelos;
```

Para referenciar uma classe ou um conjunto delas (packages), é preciso declarar suas importações diretamente no arquivo onde serão utilizadas.

```
// import [nome_do_pacote]
import com.organizacao.modelos.Pessoa;
import com.organizacao.servicos.*;
// ...

public static void main(String[] args) {
    Pessoa pessoa = new Pessoa();
}
```

Sem a declaração de `import` é necessário especificar o caminho completo do package de uma classe.

```
// sem imports
// ...

public static void main(String[] args) {
    com.organizacao.modelos.Pessoa pessoa = new com.organizacao.modelos.Pessoa();
}
```

Porque usar packages?

- Encapsula um conjunto de classes
- Evita conflitos de nome e garantem encapsulamento e proteção

Sintaxe

- Declarações
- Métodos
- Booleanos
- Strings
- Arrays

Declarações

Declarações de variáveis são realizadas de maneira **tipada** (quando é necessário informar que tipo de dado vamos armazenar em uma variável)

Para declarações de variáveis dentro de classes, podemos utilizar também o modificador de acesso (`public` , `protected` , `private`). À essas variáveis damos o nome de **Atributos**.

```
public int numero;
public String nome;
protected boolean estaChovendo;
private String[] chamada;
```

Métodos

Métodos também possuem modificadores de acesso

```
public void imprimir() {

}
public String retornaNome(String nome) {
    return nome;
}
public int calcula(int a, int b) {
    return a + b;
}
```

Booleanos

Representa tipos de dados que só podem assumir dois valores (`true` e `false`). Tipicamente ocupa um byte de memória.

```
boolean estaChovendo = true;
boolean vaiChover = estaChovendo || (probabilidadeChuva > 90);
```

Utiliza-se de operadores de comparação (como `>`, `<` e `=`) operadores booleanos:

- `||` - OU
- `&&` - E
- `!` - NEGAÇÃO

Strings

Strings são sequencias de caracteres utilizados para representar texto. São representadas pelas `"` (Aspas duplas) e podem ser concatenadas (unidas) usando o operador `+` (soma).

```
String nome = "Wesley"
String sobrenome = "Safadão"
String artista = nome + " " + sobrenome;
```

Strings são imutáveis, significando que após geradas não serão modificadas.

Métodos de Strings

Assumindo que acabamos de criar uma nova string chamada `texto`, podemos utilizar os seguntes métodos:

- `texto.length` - retorna o tamanho do texto
- `texto.equals("teste")` - compara se o conteúdo de texto é igual a `teste`
- `texto.toLowerCase()` - retorna o conteúdo de texto em caixa baixa
- `texto.toUpperCase()` - retorna o conteúdo de texto em caixa alta
- `texto.replace("a", "b")` - substitui todos os caracteres `a` por `b`
- `texto.split("x")` - quebra a string em várias strings usando a letra `x`

Arrays

Arrays são estruturas capazes de armazenar múltiplos valores de um mesmo tipo sob uma mesma variável de referência.

```
// declara um novo vetor com 6 números inteiros
int[] numerosMegaSena = new int[6];
// numerosMegaSena ==> int[6] { 0, 0, 0, 0, 0, 0 }
```

Arrays em java possuem tamanho fixo definitivo ao serem instanciados. A primeira posição de um vetor sempre será 0.

```
numerosMegaSena[0]; // acessa a primeira posição do vetor anterior
numerosMegaSena[5]; // acessa a última posição do vetor anterior
numerosMegaSena[6]; // indexOutOfBounds - fora de posição
```

Também podemos utilizar as chaves para gerar um novo vetor com valores pré-definidos:

```
int[] numerosMegaSena = {4, 11, 19, 25, 33, 42};
```

Vetores de qualquer tipo de dado podem ser gerados, incluindo os não primitivos:

```
Object[] dados = {"matrix", 10, true};
```

> Os valores acima são respectivamente String, Integer e Boolean, que são, por definição, todos herdeiros da classe Object.

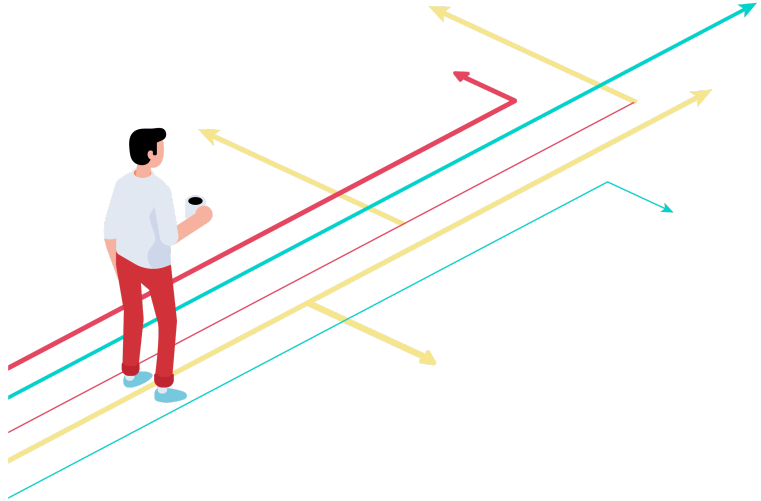
Métodos de Arrays

Assumindo que acabamos de criar um novo array chamado chamado `lista`, podemos utilizar os seguntes métodos:

- `lista.length` - retorna o tamanho do vetor
- `lista.equals(lista2)` - compara se o vetor `lista` é igual ao vetor `lista2`. Vetores são iguais se possuem mesmo tamanho, valores iguais e na mesma ordem.
- `Arrays.toString(lista)` - imprime o conteúdo do vetor em forma de texto
- `Arrays.fill(dados, 1)` - preenche todos as posições do vetor com o valor
- `Arrays.sort(dados)` - Ordena o vetor por ordem numerica ou lexicográfica

Estrutura de Decisão

Estruturas de decisão são responsáveis por definir o fluxo de execução de um código. Essas decisões geralmente criam diferentes rotas para aplicação e devem ser pensados com cuidado pois aumentam o número de testes necessários para um código (test covering)



IF-ELSE

Utilizado para definir o fluxo do código. Testes booleanos (`true` ou `false`) são usados como condição para executar ou não um bloco de código.

```
if (condicaoBooleana) {
    executeIssoSeVerdadeiro()
}
```

```
if (condicaoBooleana) {
    executeIssoSeVerdadeiro()
} else {
    executeIssoSeFalso()
}
```

Operador Ternario

Tem como objetivo retornar um valor a partir de uma condicional, em uma única operação;

```
int valorFinal = if (condicaoBooleana) ? valorSeVerdadeiro : valorSeFalso;
```

Estrutura de Repetição

Estruturas de repetição permitem executar blocos de código diversas vezes, geralmente com parâmetros diferentes à cada iteração.

As duas principais estruturas que utilizamos (em diversas linguagens) são:

- for
- while

for tradicional

Usado geralmente para iterar sobre uma quantidade definida de operações. É composto por declaração, condição booleana e uma operação pós-loop, porém todos os valores são opcionais.

```
int quantidadeDeIteracoes = 5;
for (int i = 0; i < quantidadeDeIteracoes; i++) {
    // faça isso
}
```

for-each

Usado para iterar sobre coleções iteráveis (especialmente vetores).

```
String[] cidades = {"içara", "forquilhinha", "maracajá"};
for (String c : cidades) {
    System.out.println(c);
}
```

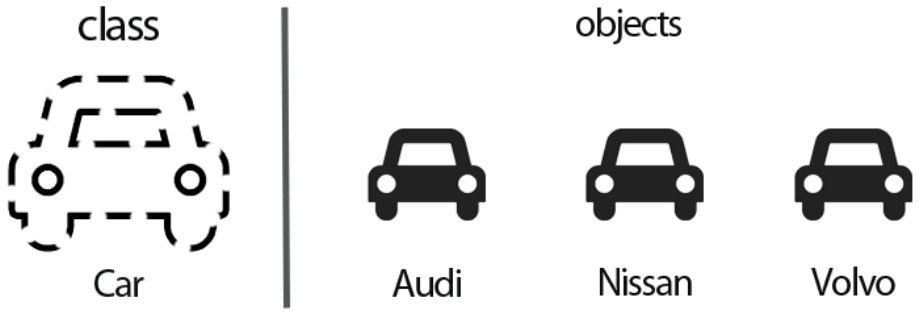
while

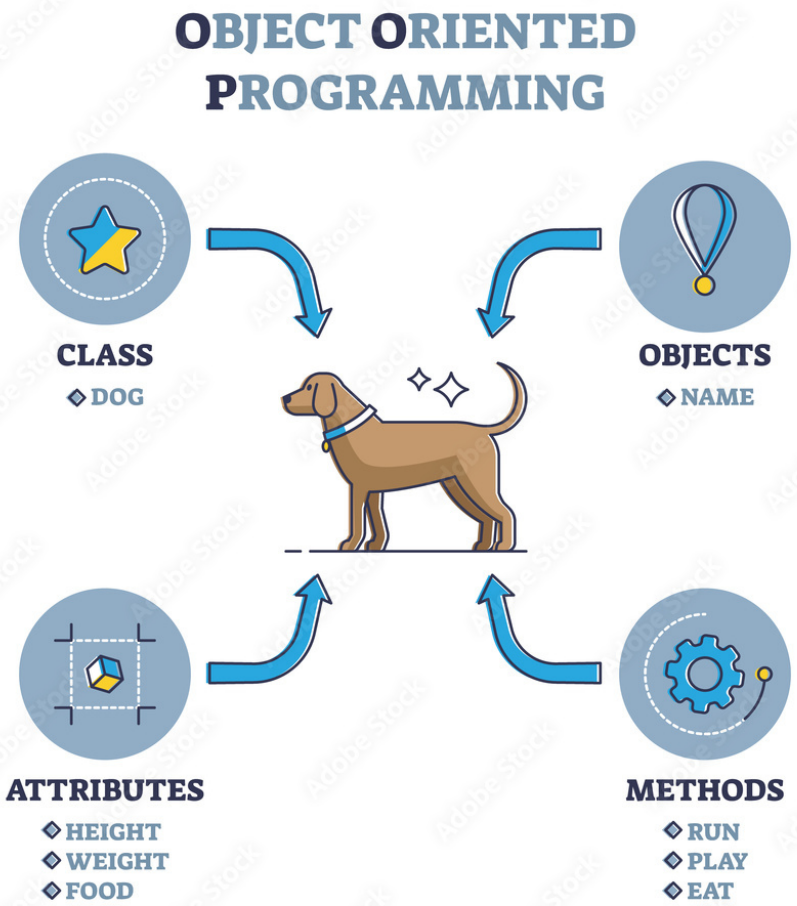
Usado geralmente para iterar quando a condição booleana deve ser manipulada de dentro do loop. Similar ao for-loop tradicional.

```
while(condicaoBooleana) {
    // repete enquanto a condição booleana for verdadeira
}
```

Orientação à Objeto

Orientação à Objeto (OOP - Oriented Object Programming) é um conceito de programação que tem como princípio a utilização de abstrações para modelar um sistema em torno da manipulação de objetos isolados.





Na Orientação à Objetos, todos os componentes de um sistema possuem atributos que o descrevem e métodos que descrevem funcionalidades.

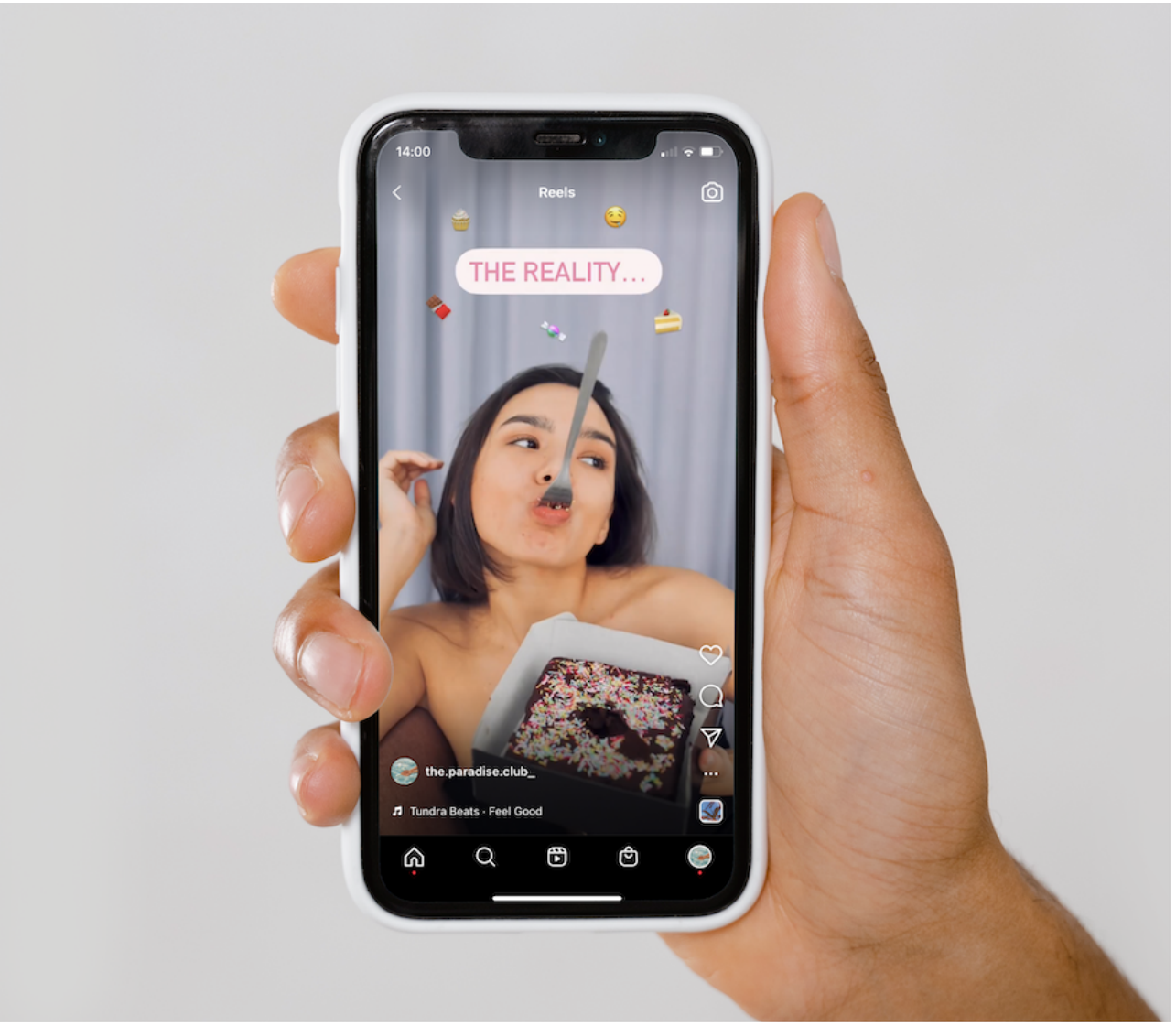
Em uma analogia com o mundo real, podemos utilizar qualquer tipo de entidade (um cachorro por exemplo) e descrever seus atributos e funcionalidades

Um cachorro pode ser descrito (*atributos*) como:

- Cor
- Raça
- Tamanho
- Peso

E suas funcionalidades (*métodos*):

- Correr
- Dormir
- Comer
- Brincar



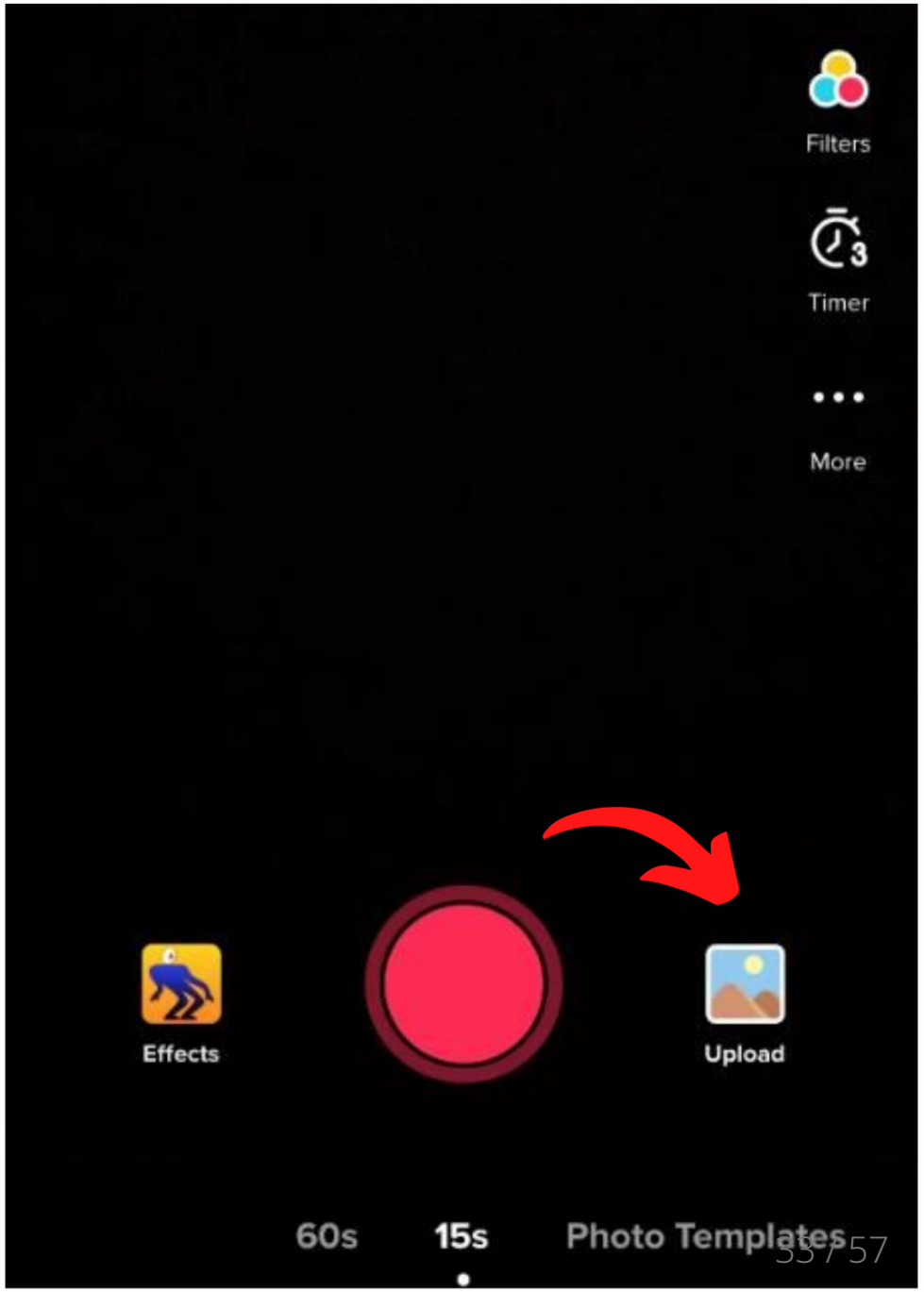
Além de objetos materiais ou pets, também podemos construir objetos a partir de modelos mais abstratos, como por exemplo um upload de vídeo.

Atributos:

- Título
- Usuário
- Curtidas
- Duração em Segundos
- Publicado

Funcionalidades:

- Exibir detalhes do vídeo
- Exibir comentário mais popular do vídeo
- Alterar o número de curtidas do vídeo



Classes

O Java utiliza uma abordagem orientada à objetos com suporte a classes. Isso significa que os objetos criados usando java primeiro precisam ser declarados em forma de classe.

A classe representa o formato pelo qual um ou vários objetos podem ser criados.

Usando a mesma analogia do cachorro do mundo real, uma classe é basicamente a **estrutura** que descreve um cachorro.

Os diferentes cachorros existentes são os **objetos únicos** criados a partir dessa estrutura.

Para criar uma classe utilizamos a seguinte sintaxe:

```
public class Funcionario {
    public String nome;
    public String codigo;

    public void baterPonto() {
        System.out.println("Ponto registrado com sucesso!");
    }

}
```

Repare que dentro da declaração da classe podemos declarar no mesmo nível diferentes variáveis (aqui chamadas de atributos) e funções (aqui chamadas de métodos).

Criando um novo objeto a partir de uma classe

Com a classe criada, vamos utilizar a palavra `new` para criar um novo objeto. É necessário que o tipo de dados seja o mesmo nome da classe:

```
public class Main {
    public static void main(String[] args) {
        Funcionario estagiario = new Funcionario();
        estagiario.nome = "Ronaldo Fenômeno";
        estagiario.baterPont();
    }
}
```

A variável `estagiario` nesse caso será do tipo `Funcionario` e conterá todos os atributos e métodos descritos na classe.

this

`this` é um atributo especial de uma classe, que representa um objeto dentro dele mesmo.

Sua definição pode parecer confusa, mas na prática o this é utilizado para eliminar a confusão entre os atributos da classe e os parâmetros usados em seus métodos.

```java
public class Funcionario {
    public String nome;

    public void editaNome(String nome) {
        /*
        this.nome refere-se ao atributo da classe
        nome refere-se ao parâmetro da função
        */
        this.nome = "Funcionario: " + nome;
    }
}
```

Herança

A herança permite à uma classe utilizar-se de todos os atributos e métodos de outra classe, funcionando como uma especialização da mesma.

```
public class Funcionario {
    public String nome;
    public String codigo;

    public void baterPonto() {
        //
    }
}
public class Professor extends Funcionario {
    String disciplina;
}
```

Tudo o que foi definido para uma classe é automaticamente herdado para a classe que à extende.

```
public class Main {
    public static void main(String[] args) {
        Professor ramon = new Professor();
        professor.nome = 'Ramon'
        professor.baterPonto();
    }
}
```

Polimorfismo

Polimorfismo é a capacidade de criar funções com o mesmo nome usando diferentes assinaturas ou sobrescrevendo as assinaturas das classes herdadas. O Java permite implementar polimorfismo no código usando:

- Sobrecarga
- Sobreposição

Sobrecarga

Polimorfismo por sobrecarga é a capacidade de criar métodos em uma mesma classe com diferentes parametros:

```
public class Funcionario {
    public void baterPonto() {

    }

    public void baterPonto(int hora, int minuto) {

    }
}
```

Sobreposição

Polimorfismo por sobreposição é a capacidade de re-criar métodos de uma classe heradada. Diferente da sobrecarga, o método sobreescrito deve manter os mesmo parâmetros do original.

```
public class Professor extends Funcionario {
    @Override
    public void baterPonto() {
        System.out.println("Apenas em horário de aula");
    }
}
```

> O método sobreposto acompanha a anotação `@Override`

Encapsulamento

Encapsulamento é a capacidade de esconder atributos, métodos e classes para outras classes e packages.

```
public class Funcionario {
    private String nome;
}

public class Main {
    public static void main(String[] args) {
        Funcionario ramon = new Funcionario();
        professor.nome = 'Ramon' // ERRO
    }
}
```

Atributos ou métodos definidos como `private` **não podem ser acessados de fora da classe**.

Atributos ou métodos definidos como `protected` **não podem ser acessados fora do mesmo package**.

Atributos ou métodos definidos como `public` **podem ser acessados de qualquer lugar**.

GETTERS e SETTERS

É comum utilizar-se dos chamados Getters e Setters para acessar e modificar o valor de atributos privados.

```
public class Funcionario {
    private String nome;

    public String getNome() {
        return nome;
    }

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

Os métodos get (getters) de um atributo geralmente inicial com a palavra `get`.

Os métodos set (setters) de um atributo geralmente iniciam com a palavra `set`.

Ambas as situações são apenas uma **convenção** bem conhecida na programação.

Abstração

A abstração é a capacidade de reduzir detalhes de uma implementação de forma que o código possa atingir menor complexidade e maior reusabilidade. O Java possui duas formas principais de atingir abstração:

- Classes abstratas
- Interfaces

Exceções

Exceções são eventos que ocorrem durante a execução do programa que alteram seu fluxo normal. No Java, podemos organizar exceções como:

- `Checked`: Exceções que são checadas em tempo de compilação e que exigem o tratamento ou uso da clausula `throws`. (ex.: `IOException`)
- `Unchecked`: Exceções que são geradas em tempo de execução (Ex.: `ArrayIndexOutOfBounds`)

Tratando exceções

Há duas formas básicas de tratar Exceções. A primeira é explicitamente ignorando seu tratamento no escopo:

```
/* Usando a clausula throws neste método, garantimos
    que não será necessário realizar nenhum tipo de
    tratamento aqui, porém o erro será disparado para
    qualquer local onde a funcao leArquivo seja invocada
*/
public void leArquivo(String caminho) throws Exception{
    File file = new File("example.txt");
    return file;
}
```

Exceções podem ser especificadas. Ao invés de utilizar a classe `Exception`, que reconhece qualquer tipo de exceção, podemos definir exatamente que tipo de exceção pretendemos tratar:

```
public void leArquivo() throws FileNotFoundException{
    File file = new File("example.txt");
    return file;
}
```

> `FileNotFoundException` é gerada quando o arquivo não está presente no sistema de arquivos

Para tratar uma exceção e evitar que ela seja transmitida para cima. Podemos utilizar o bloco `try-catch`:

```
public void leArquivo() {
    try {
        File file = new File("example.txt");
    } catch (FileNotFoundException exception) {
        System.out.println(exception);
    }
    return file;
}
```

Dessa forma, sempre que a exceção ocorrer, o código dentro da clausula `catch` será executado e o programa não será interrompido.

Gerando exceções

Para gerar uma nova excessão no código, também podemos utilizar a clausula throws, porem no local onde a `exception` acontece no código.

```
public int dividir(int dividendo, int divisor) {
    if (divisor == 0) {
        throw new ArithmeticException("Impossível dividir por zero!");
    }
    return dividendo / divisor;
}
```

Terminologias

- `Tempo de execução` - aquilo que acontece enquanto o programa roda
- `Tempo de compilação` - o que acontece quando o programa é compilado

Outros Pontos

- Construtores e Destrutores
- Generics
- Streams
- Collections
- Threads
- Logs
- Data Structures
- Files e HTTP Requests
- Build Tools

O que aprendemos hoje

- O que é a plataforma Java e seus componentes
- A sintaxe básica para a linguagem java
- Estruturas da linguagem e seu funcionamento