# Aula 01: 15/03/2017

## Tópicos

- apresentação de MAC0323
- introduzir orientação a objetos em Java através de um programa simples

## MAC0323

### Bibliografia principal

- página Estrutura de Dados de Paulo Feofiloff (**PF**) `https://www.ime.usp.br/~pf/estruturas-de-dados/` notas de aula baseadas no livro *Algorithms* de Sedgewick & Wayne
- Livro *Algorithms* de Sedgewick & Wayne (**SW**). Website do livro: `http://algs4.cs.princeton.edu`

## Introdução (por Paulo Feofiloff)

Leitura: Introdução, PF

Estruturas de dados servem para organizar os dados de um problema de modo que eles possam ser processados mais eficientemente.

Eficiência tem a ver com escalabilidade: como o tempo de processamento cresce quando a quantidade de dados (ou seja, o tamanho) do problema aumenta? O tempo cresce de maneira moderada? Cresce de maneira explosiva?

**Exemplo**: Qual a diferença entre encontrar uma determinada palavra em uma lista de 10 palavras e encontrar a palavra em uma lista de 10000 palavras? O tempo de processamento é apenas dez vezes maior? É mil vezes maior? Ou é um milhão de vezes maior?

Não importa muito como encontrar um palavra em uma lista com 10 palavras. É como nadar no raso. Tanto faz como fazemos. O que importa mesmo é quando o tamanho do problema cresce.

## Biblioteca de programas

Para instalar o ambiente de programação veja a página Biblioteca de programas, PF.

Instale o ambiente de programação do livro no seu computador.

Programas que utilizaremos:

- `javac`: compilador java
- `java`: máquina virtual do java

  ```
  % file /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
  /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java: ELF 64-bit LSB shared object, x86-64, vers
  ```

- `checkstyle`: encontra os erros de estilo nos arquivos `.java`
- `findbugs`: encontra os eventuais erros no arquivo `.class`

- `DrJava`: IDE extremamente leve que é usada para escrever programas em Java. Foi projetada especialmente para estudantes e possui uma interface intuitiva e a habilidade de executar código Java **interativamente**.

## Tipos abstratos de dados (ADTs)

Leitura: https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/adt.html

**ADT**: conjunto de itens junto com conjunto operações eles.

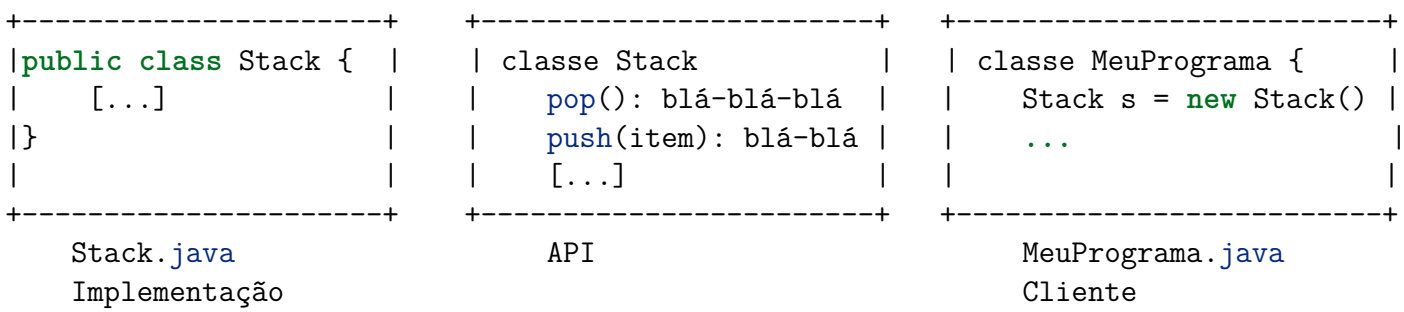
**API** (*applications programming interface*): especificação das operações.

Exemplo de ADT: pilha. As operações são: inserção de um item (`push()`) e remoção de um item (`pop()`).

**Cliente**: usuário do ADT. Não está nem ai para como a ADT foi implementada. Interessado apenas no que ela faz e possivelmente no consumo de tempo e espaço.

## Implementação/interface/cliente

Interface é contrato entre o cliente e a implementação.




## Anatomia de um programa em Java

Pelo menos em MAC0323

```
public class MinhaClasse {
    // 1 atributos de estado variáveis

    // 2 contrutor, invocado quando fazemos new MinhaClasse(...)
    public MinhaClasse(...) {
        [...]
    }

    // 3 métodos publicos, fazem parte da API
    public bollean isEmpty() {
        [...]
    }

    public String toString() {
        [...]
    }
```

```
    // 4 métodos privados, métodos auxiliares
    private int metodoAuxiliar(...) {
        [...]
    }

    // 5 unit test
    public static void main(String[] args) {
        [...]
    }
}
```

**Exemplo**

API:

```
    public class Counter
    --------------------
           Counter(String id)   cria um contador de nome id
      void increment()          incrementa o contador
      int  tally()              numero de incrementos
    String toString()           representação de um contador como string
```

```
// https://www.ime.usp.br/~pf/sedgewick-wayne/stdlib/documentation/index.html
import edu.princeton.cs.algs4.StdOut;

public class Counter {
    private final String name; // variável de instância
    private int count;         // variável de instância
    private static int noCounters = 0; // variável de classe

    // construtor
    public Counter(String id) {
        name = id;
        count = 0; // supérfluo
        noCounters++;
    }

    public void increment() {
        count++;
    }

    public int tally() {
        return count;
    }

    // retorna um string usado por print...
    public String toString() {
        return count + " " + name;
    }
```

```
    // retorna o número de contadores
    public static int size() {
        return noCounters;
    }

//
    public static void main(String[] args) {
        Counter pares = new Counter("pares");
        Counter impares = new Counter("ímpares");

        pares.increment();
        pares.increment();
        impares.increment();

        StdOut.println(pares + " " + impares);
        StdOut.println(pares.tally() + impares.tally());
        StdOut.println("número de contadores: " + Counter.size());
    }
}
```

## Cliente de Counter

Com leitura de arquivo e entrada padrão.

```
import edu.princeton.cs.algs4.In;
import edu.princeton.cs.algs4.StdOut;

public class ParesImpares {
    public static void main(String[] args) {
        In in = new In(args[0]);
        Counter pares = new Counter("pares");
        Counter impares = new Counter("impares");

        while (!in.isEmpty()) {
            int valor = in.readInt();
            StdOut.printf("%d\n", valor);
            if (valor % 2 == 0) pares.increment();
            else impares.increment();
        }

        StdOut.println("Relatório: ");
        StdOut.println(pares);
        StdOut.println(impares);
    }
}
```

## Modificadores

`public`: métodos e variáveis acessíveis a todos os clientes. Especificados na API.

`private`: métodos e variáveis auxiliares, clientes não têm acesso.

`static`: são da classe, cada instância da classe tem sua própria cópia dos demais métodos e variáveis.

## Objetos

### Classes

Classes são formadas por atributos que podem ser variáveis ou funções que são chamadas de métodos.

A primeira letra em um nome de uma classe deve ser maiúscula e a comunidade java usa camelCase.

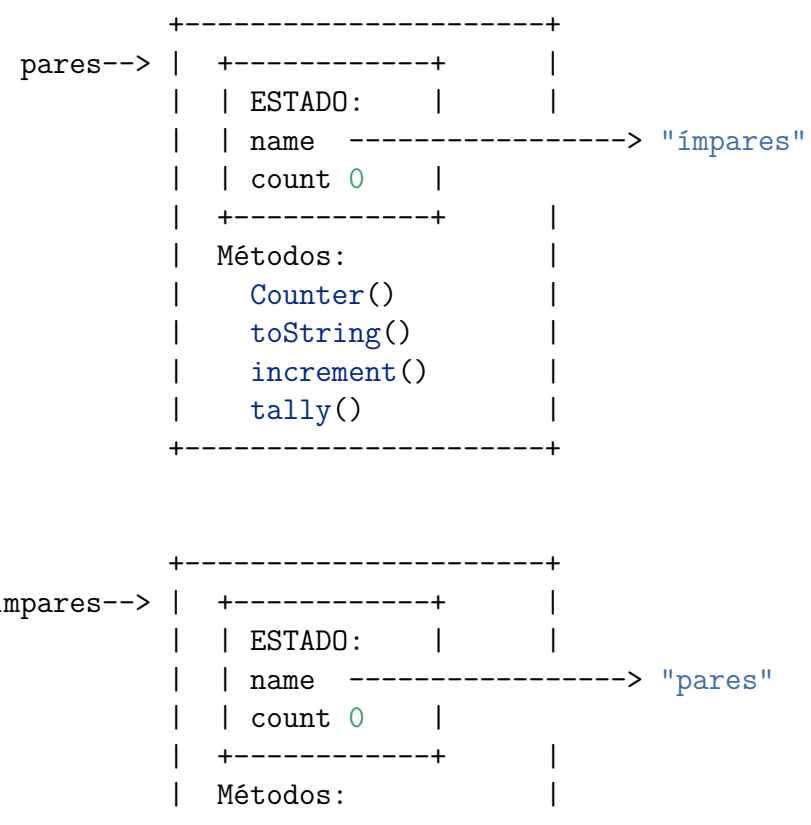
### Objetos

Um objeto contém as informações/valores de um ADT (classe) definido pelo programador.

### Esquema geral da classe Counter

Visão geral que será explicada pouco a pouco. Atributos de uma classe podem ser de estado (variáveis) ou funções (métodos).

```
Counter pares = new Counter("pares");
Counter impares = new Counter("ímpares");
```



```
|    Counter()         |
|    toString()        |
|    increment()       |
|    tally()           |
+----------------------+
```

Utilizamos *dot-notation*: `pares.increment(), impares.tally(),...`

### Métodos

Métodos são funções associadas com uma determinada classe.

```
pares.increment();
```

Métodos são como funções, mas há duas diferenças:

- métodos são definidos dentro de uma classe
- a sintaxe para executar um método é diferente Por exemplo, função `imprima()`, aqui está um objeto para você imprimir. Enquanto, `r.imprima()` sugere objeto `r`, imprima a si mesmo. Essa mudança de perspectiva pode ser polida, mas não é óbvio que seja útil.

Algumas vezes mover a responsabilidade de uma função para um objeto faz com que seja possível escrever um código mais versátil que é mais fácil de ser reutilizado e mantido. Blá-blá-blá. E com a palavra, a turma de engenharia de software.

No momento o que está escrito acima é longe de óbvio...

### Construtores

O método especial `public NomeDaClasse(...)` é responsável por construir e retornar um objeto.

Chamado quando um objeto é criado (= *instanciado* é um nome mais bonito) através da palavra `new`.

### Imprimindo um objeto

O método especial `toString` cria e retorna um `String` que diz como o objeto deve ser impresso por `StdOut.print(), StdOut.println(), StdOut.printf(),...`.