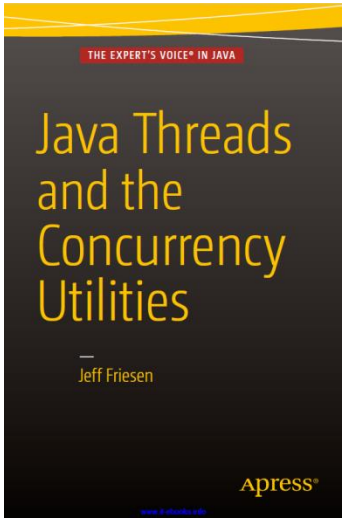
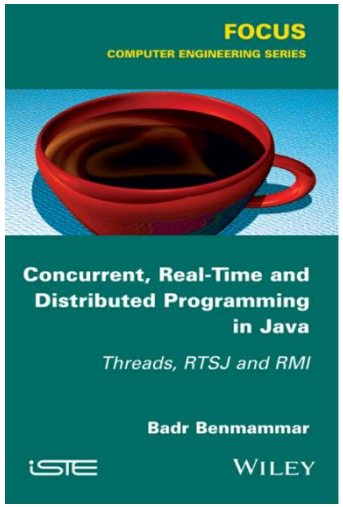
Programação Paralela e Concorrente

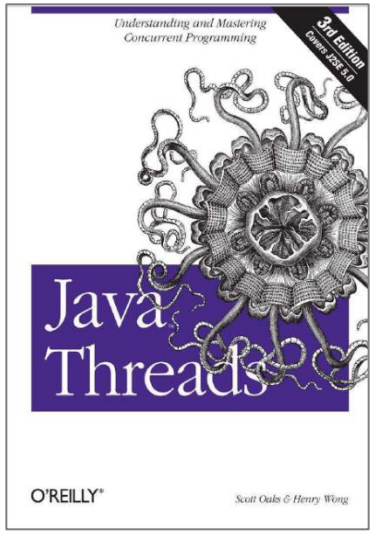
Unidade 2 - Processamento de Threads

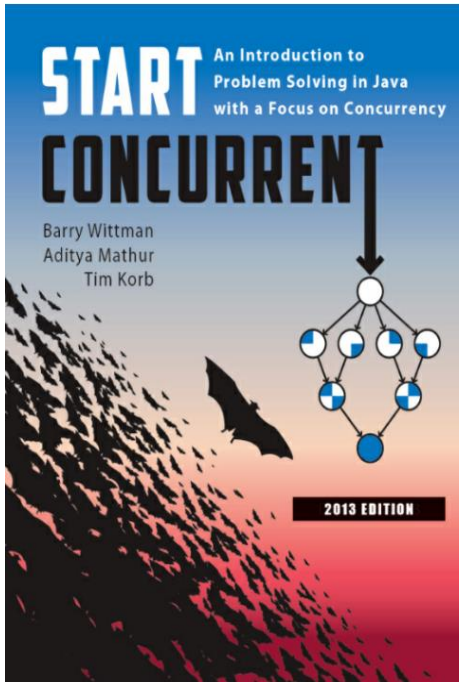
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

Bibliografia





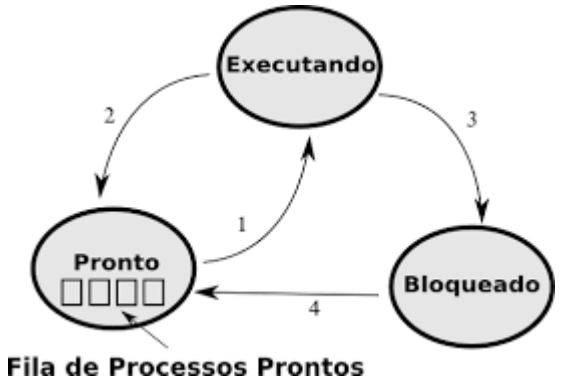




Qual a diferença entre Programa e Processo?

Programa e Processo

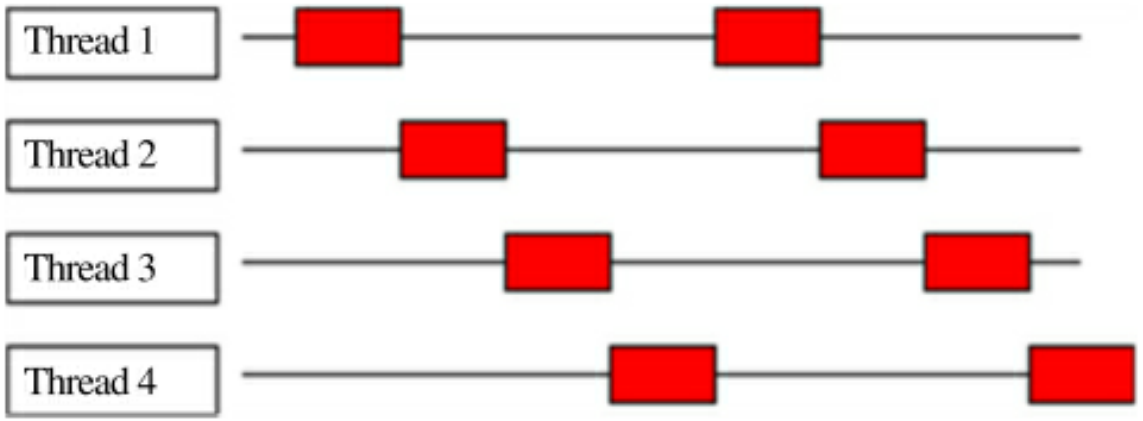
- **Sistemas operacionais** se encarregam de **alocar** os recursos necessários para que programas sejam executados;
- **Programas** são entidades **passivas**;
- Para que programas possam ser executados, o sistema operacional cria, em tempo de inicialização do programa, estruturas chamadas **processos**, para controle de execução.



Qual a diferença entre Processo e Thread?

Processo e Thread

- Um processo tem um fluxo de execução principal denominado **thread**;
- Em um **mesmo** processo pode-se iniciar **múltiplos** threads, criando-se assim unidades de processamento concorrentes dentro do processo.
- **Threads** compartilham recursos e espaço de endereçamento de memória do processo a que pertencem.

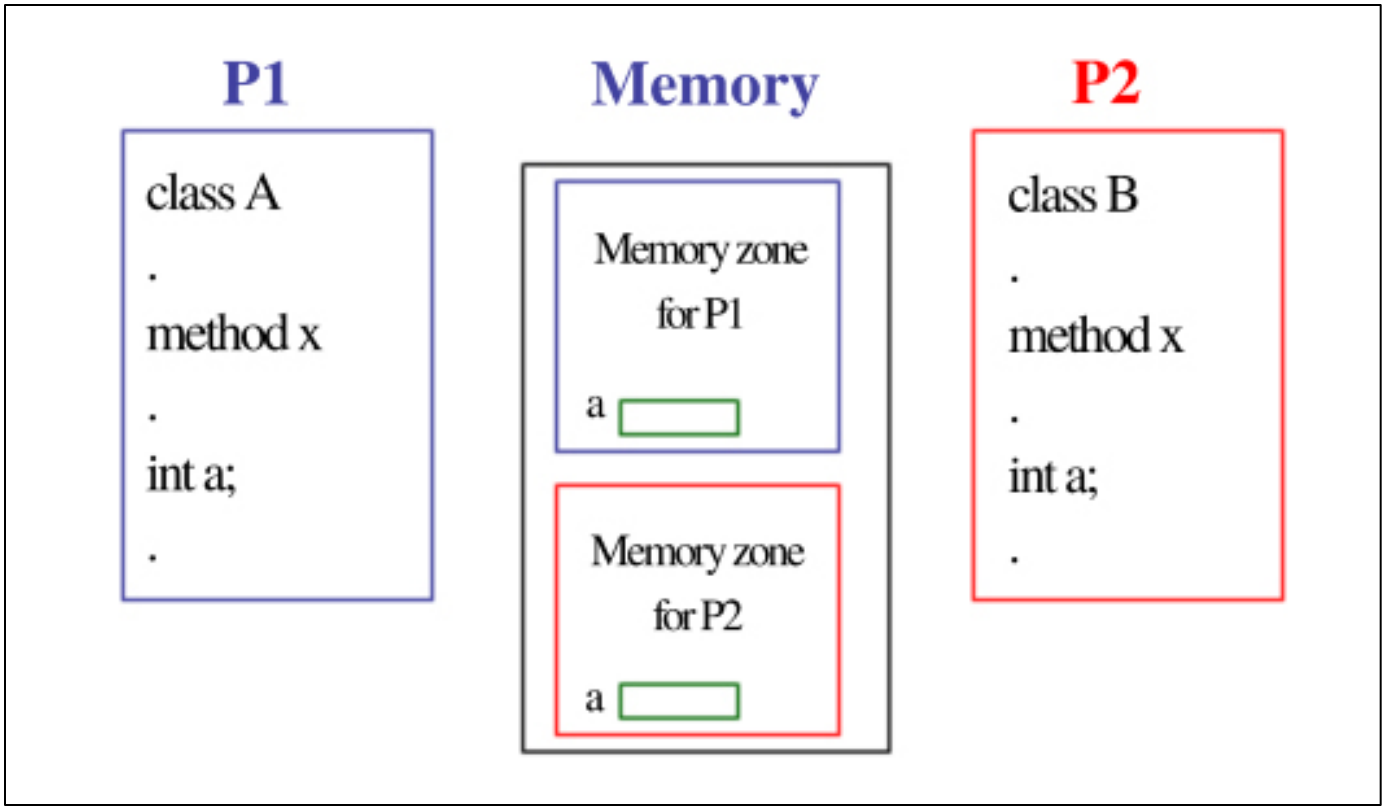


Processos x Threads

- Sistemas operacionais cuidam da **alocação** dos **recursos** (memória, processador, I/O) para que o processo possa ser executado;
- Adicionalmente, asseguram que processos sejam **isolados** de outros.

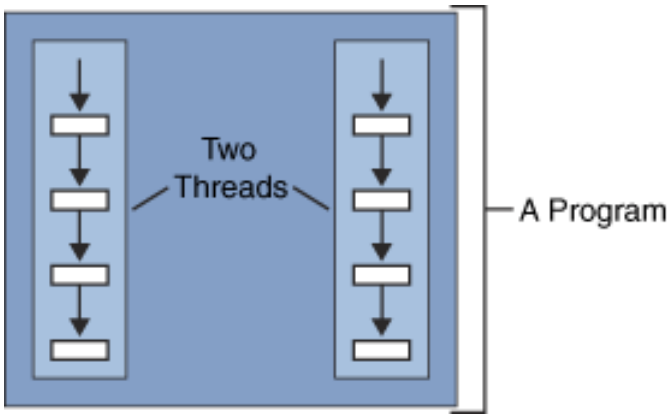
Isolamento de Processos

- Considere uma variável **a** definida em duas diferentes classes;
- Ao se executar concorrentemente as duas classes, as áreas de memória para esta variável **a** são completamente **isoladas**.



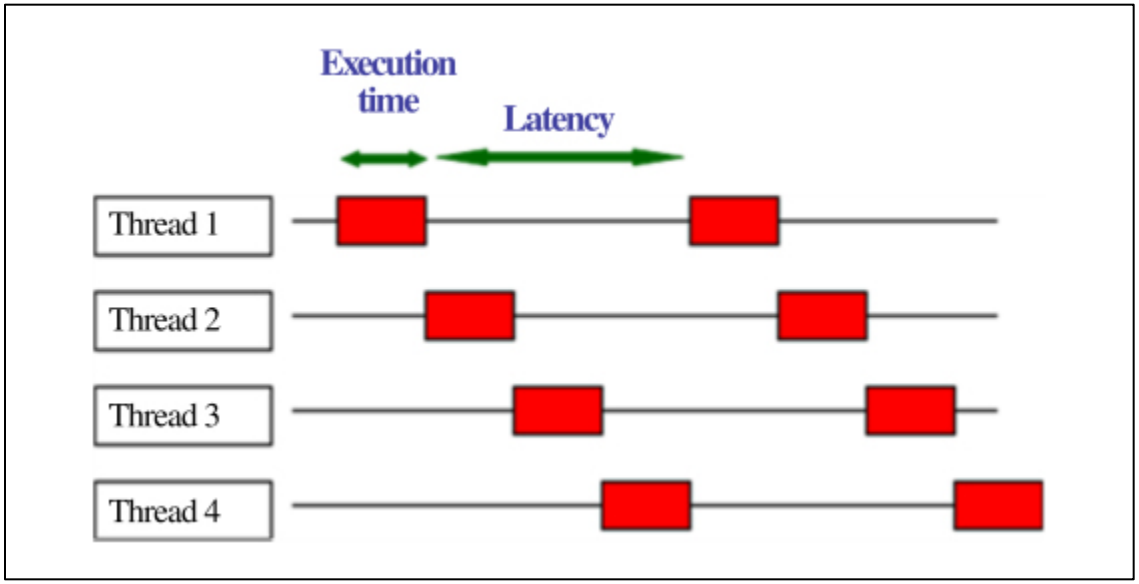
Processos x Threads

- A maioria dos sistemas operacionais oferecem uma distinção entre:
  - Processos com separação forte uns dos outros (**Heavy-weight**)
  - Processos que **compartilham** espaço de **memória** (**Threads – Light-weight**)

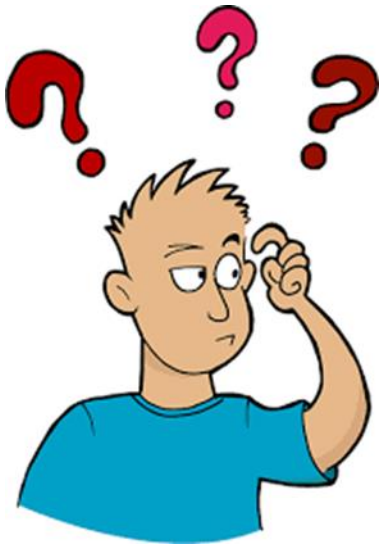


Threads

- Um **thread** corresponde a um trecho de código que possui a capacidade de ser executado concorrentemente com outros processos;



Quando uma linguagem pode ser considerada como Concorrente?



Linguagem Concorrente

- Permite a **criação** de **threads**;
- Permite o **compartilhamento** de dados entre **threads**;
- Permite o controle de **sincronização** entre **threads**.

Sincronização por Competição

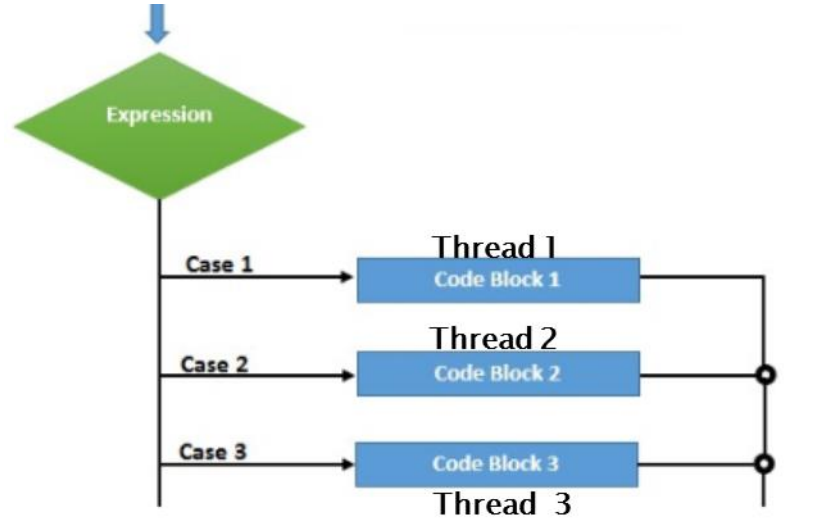
- **Controle** da ordem de execução das tasks que ocorre quando mais de um thread estiver acessando o **mesmo recurso**.

Sincronização por Cooperação

- Controle da ordem de execução das tasks que ocorre quando um thread **espera** um outro terminar a execução antes de iniciar a sua.

Threads com a Linguagem Java

- Aplicações **Java** são executadas por meio de **threads**;
- Threads, como vimos, são trechos de códigos **independentes** da aplicação;
- Ao se trabalhar com aplicações **multithreads**, cada **thread** pode executar códigos que seguem fluxos de execução independentes;
- Por exemplo, um thread **T1** pode executar uma das instruções **case** de um comando **switch** e outro thread **T2** pode executar outro case;



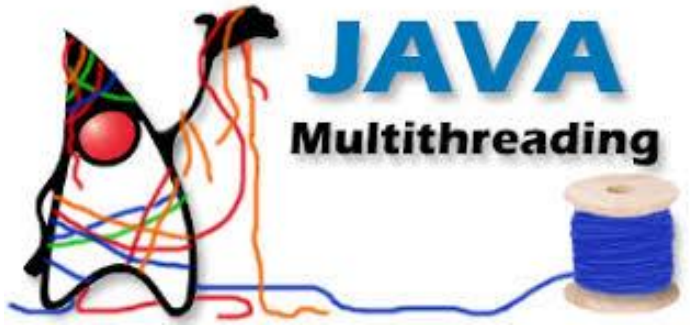
Threads com a Linguagem Java

- Toda aplicação Java tem ao menos **1 thread** (**default**) que executa o método **main();**

- Todavia, em tempo de execução, a aplicação pode também criar **novos** threads para executar tarefas **time-intensive** em modo background de forma a manter responsividade aos usuários;

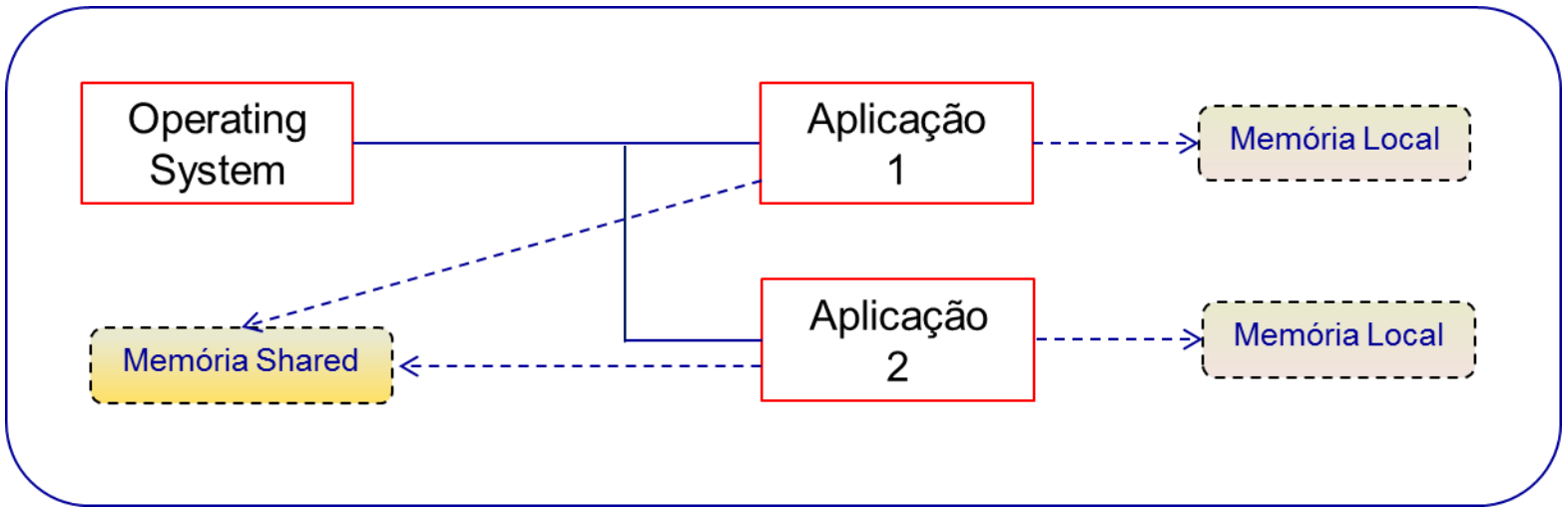
- Esses **threads** executam sequências de código encapsuladas em objetos que são conhecidos por ***runnables***;

- A linguagem Java suporta threads por meio do classe **java.lang.Thread** e da interface **java.lang.Runnable**;



Dois Programas Singled-Threaded

- Como o Sistema Operacional é **multitasking**, os dois programas rodam concorrentemente no ambiente;
- No entanto, cada processo **não** conhece detalhes do outro;
- Por default, os dados de cada processo são **isolados**, podendo haver uma área compartilhada que pode ser acessada pelos dois processos.



Programas Java Multi-Threaded

- Thread principal inicia a execução do código **main**();
- Demais threads são iniciados numa posição definida pelo **programador**;
- Cada thread efetua processamento **sequencial**;
- A execução de um thread é **independente** dos outros. No entanto, existem mecanismos de cooperação entre eles;
- Variáveis locais em um thread **não** são visíveis em outros threads;
- Variáveis instâncias e objetos podem ser compartilhados entre os diversos threads, mas é necessária a **permissão** para este compartilhamento.

Como se cria threads em Java ?

Criação de Threads em Java - Método 1

- Criar nova classe derivada de **java.lang.Thread**;

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.lang

Class Thread

java.lang.Object
    java.lang.Thread

All Implemented Interfaces:

Runnable

Direct Known Subclasses:

ForkJoinWorkerThread

```
public class Thread
extends Object
implements Runnable
```

Criação de Threads em Java - Método 1

- A classe **Thread** e suas subclasses fornecem mecanismos para a criação e controle de threads;
- Conforme **API** abaixo, a classe **Thread** implementa a interface **Runnable**;

```
public class Thread
extends Object
implements Runnable
```

Mas, o que é mesmo Interface em Java?

Interfaces são conceitos relacionados à Classes Abstratas ! ! !

Classes Abstratas

- São classes em que **não** se pode criar objetos diretamente a partir delas;

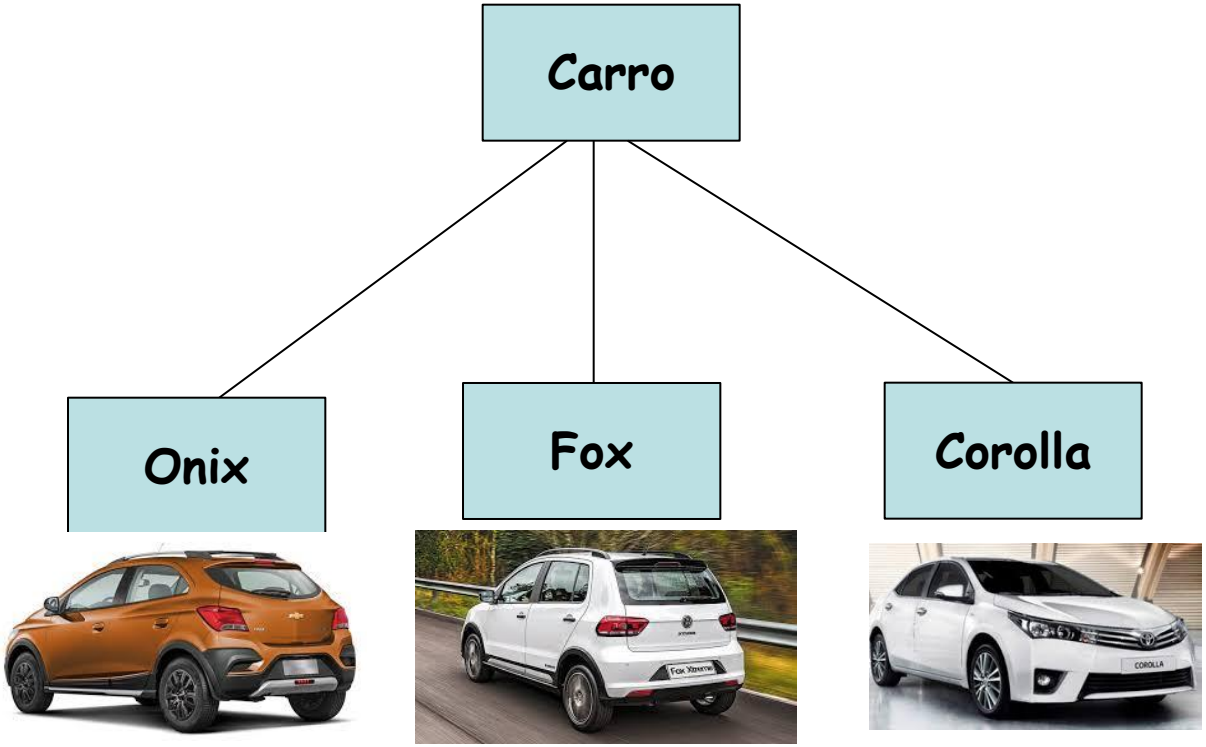
Mas, se não se pode criar objetos a partir de classes abstratas, então para que elas servem ?

Classes Abstratas

- São utilizadas par se representar grupos de objetos que têm características comuns, mas que, em alguns detalhes específicos, possuem algumas diferenças;
- Classes abstratas estão diretamente relacionadas ao conceito de Polimorfismo em Java.

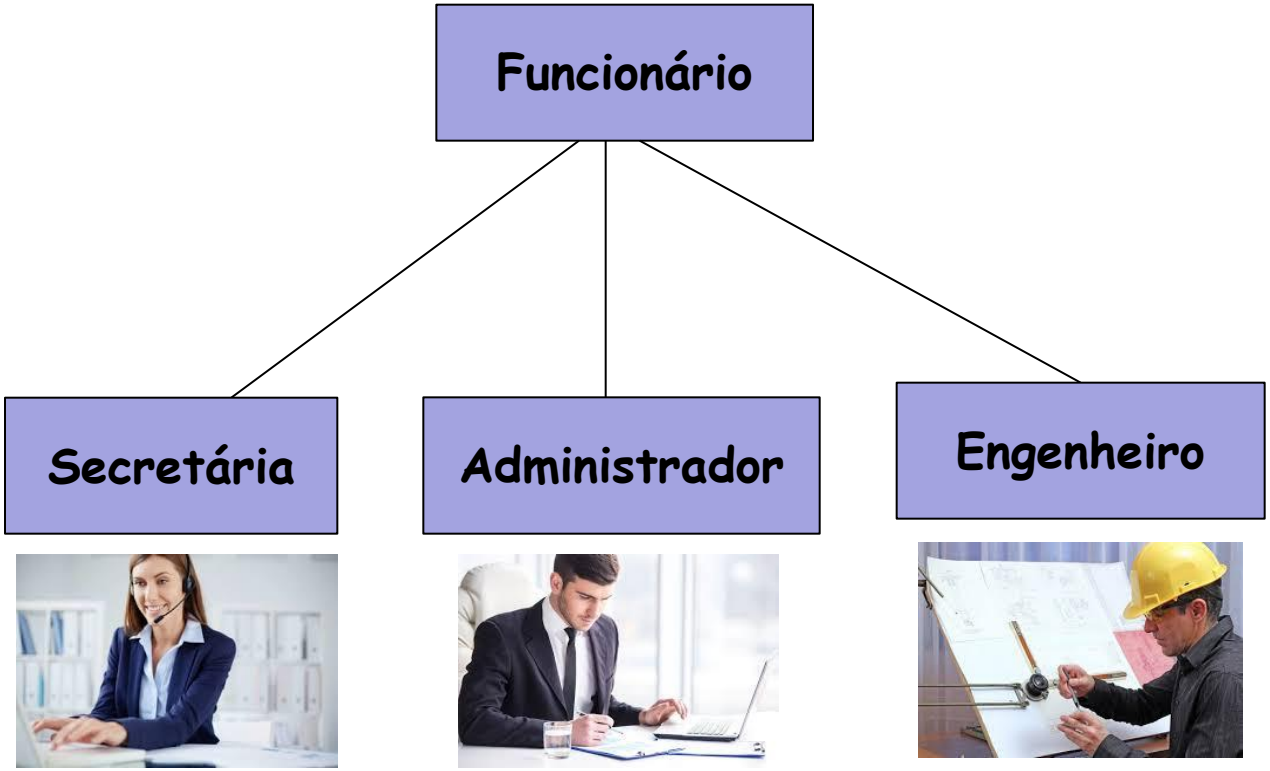
Classes Abstratas

- Se formos à uma Loja de Veículos para comprar um veículo, não compramos um **carro**, mas sim compramos um **Fox**, um **Onix**, um Corolla, ou seja, algum modelo específico.



Classes Abstratas

- Um dono de uma empresa não contrata um funcionário. Ele contrata uma secretária, um administrador, um engenheiro, ou alguém com alguma profissão em específico.



Classes Abstratas

- Na verdade, em nosso mundo real, só existem objetos;
- Classes abstratas, na verdade, são **abstrações** que usamos para representar **agrupamentos** de objetos, ou para **classificá-los**;
- Assim, classes abstratas são **superclasses** que podem ser utilizadas como base (modelo) para se definir outras classes;
- Classes abstratas podem possuir métodos concretos (com implementação) ou métodos abstratos (sem implementação);
- possuem apenas assinatura dos métodos por uma razão bem simples: os métodos irão se comportar de forma diferente nas subclasses.

Métodos Abstratos

- Métodos que **não** possuem código de implementação;
- São apenas declarados (**assinaturas**) e definidos com a keyword **abstract**;
- Classes abstratas possuem ao menos 1 método abstrato;
- Métodos abstratos possuem apenas assinatura dos métodos por uma razão bem simples: os métodos irão se comportar de forma diferente nas subclasses.

Como se definem Classes Abstratas em Java ?

Classes Abstratas

- São definidas com a keyword **abstract**.

Classes Abstratas - Exemplo

```
abstract class Animal {
    int distanciaPercorrida = 0;

    public abstract void fazerBarulho();

    public void andar() {
        distanciaPercorrida++;
    }

    public void treinar() {
        andar();
        fazerBarulho();
    }
}
```

Classes Abstratas - Exemplo

```
class Cachorro extends Animal {
    public void fazerBarulho() {
        System.out.println("Au-au!");
    }
}
```

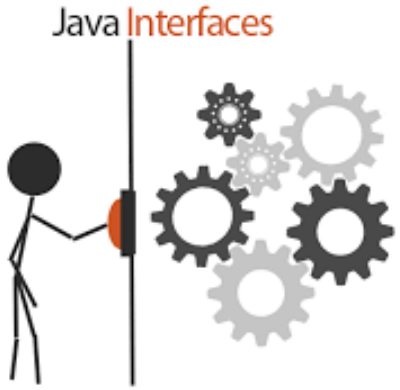
Classes Abstratas - Exemplo

```
class Gato extends Animal {
    public void fazerBarulho() {
        System.out.println("Miau!");
    }
}
```

Classes Abstratas - Exemplo

```
class Main {
    public static void main(String[] args) {
        Cachorro cao = new Cachorro();
        Gato gato = new Gato();
        cao.treinar();
        gato.treinar();
    }
}
```

Interfaces



Interfaces

- Imagine que você irá desenvolver uma aplicação no qual duas equipes irão desenvolver o software de forma **simultânea**;
- Cada equipe irá desenvolver seus códigos de forma **independente**.

Interfaces

- No entanto, deverá haver um "**contrato**" entre as equipes de tal modo que haja interação entre os códigos;
- Este **contrato** é conhecido por **interface**.

Interfaces

- **Interface** é uma forma de descrever o quê as classes devem fazer, **sem** especificar como elas devem fazê-lo;
- **Interfaces** empregam o conceito de **classe abstrata** ao **extremo**;
- É como se definisse uma classe abstrata no qual **todos** os **métodos** também sejam **abstratos**;
- Em **Java**, uma **interface** não é uma classe, mas um conjunto de **requisitos**, os quais devem ser implementados por alguma classe que aceite o **contrato**.

Interface

- Em Java, uma interface é uma definição de tipo, semelhante à classe, que pode conter apenas **constantes** e assinatura de métodos (**protótipos**);
- Numa interface não há corpo de definição de **método**.
- **Não** podem ser instanciadas, podem somente serem implementadas por classes ou ainda estendidas em outras interfaces.

Interface

- Todos os métodos de uma interface são automaticamente **public**;
- Por esta razão **não** há necessidade de incluir a keyword **public** quando estivermos declarando um método em uma interface;
- Tendo em vista que interfaces **não** são classes, **nunca** se pode usar o operador **new** para instanciar uma interface;
- Ou seja, **nunca** se instancia um objeto a partir de uma interface.

Como se define interface em Java ?

Exemplo – Interface

```
public interface XPTO {
            int func ( String a );
      }
```

- Isto significa que para qualquer classe que implementa a interface **XPTO** é requerido que se tenha a implementação do método **func** e este método deve ter um parâmetro **String** e retornar um **inteiro**.

Interfaces

- Uma **interface** é essencialmente uma **coleção** de **constantes** e **métodos abstratos**;
- Para se fazer uso de uma **interface**, deve-se **implementar** a interface na classe;
- Ou seja, deve-se definir a classe que implementa a interface e escrever o código para cada método declarado na **interface**.

Interfaces

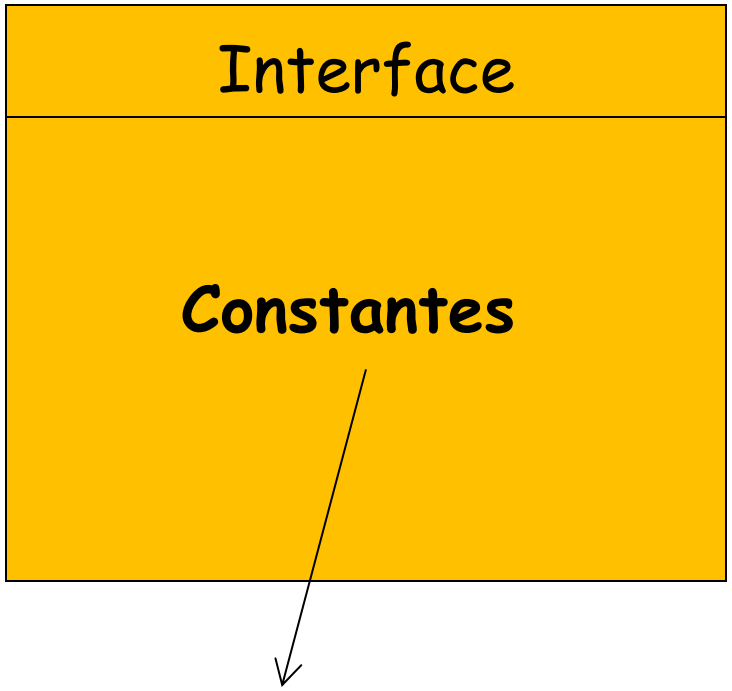
- Quando uma classe implementa uma **interface**, quaisquer constantes que foram definidas na interface são diretamente disponíveis na classe, como se fossem **herdados** de uma classe base.

O que pode conter uma interface ?

O que pode conter uma interface ?

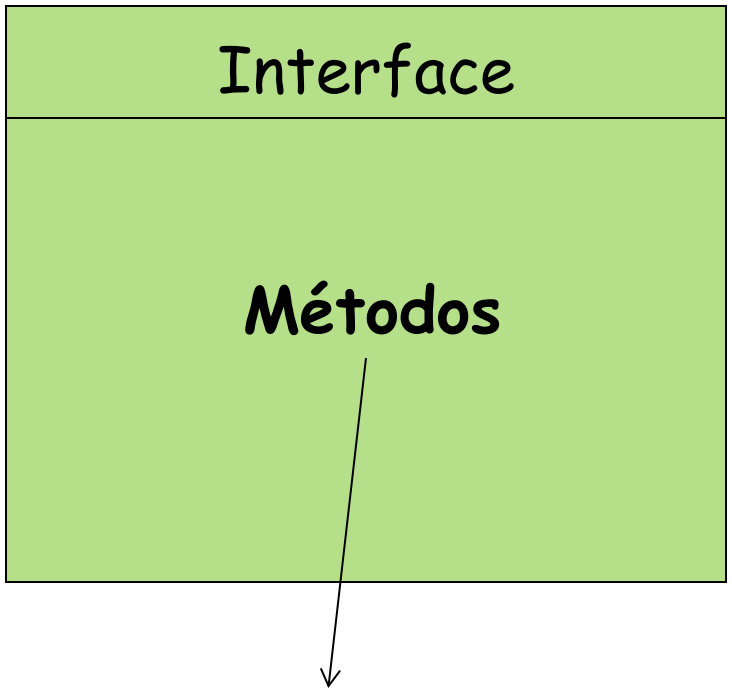
- Uma interface pode conter **constantes**, **métodos abstratos** ou **ambos**.

Interface com constantes



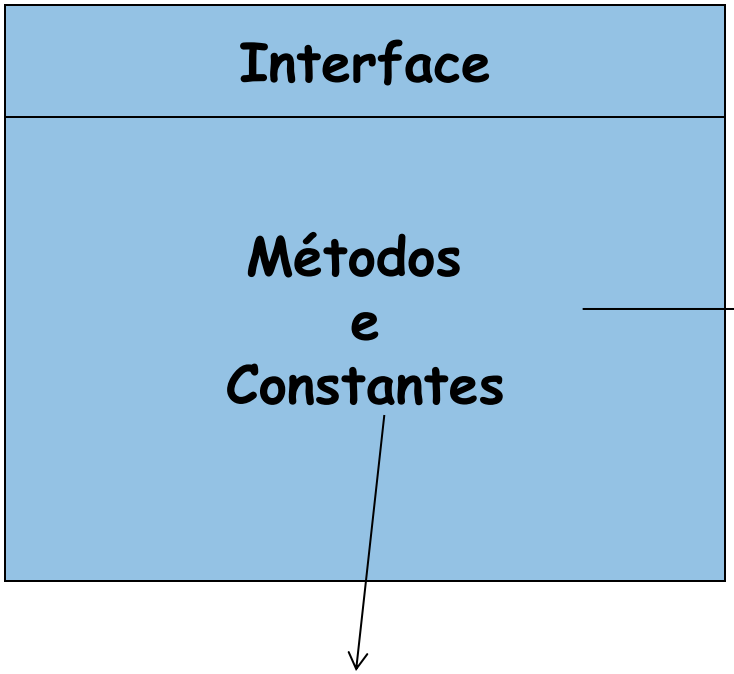
São sempre **public**, **static** e **final** por default;

Interface com métodos



São sempre **public** e **abstract** por default;

Interface com métodos e constantes



São sempre **public** e **abstract** por **default**;

São sempre **public**, **static** e **final** por default;

Como a interface trabalha ?

- Uma **interface** é definida como uma **classe**;
- Mas usa a **keyword interface** ao invés de **class**;
- O único atributo de acesso permitido em uma interface é **public**;
- Isto faz a interface acessível **fora** do package que a contém;
- Caso se omita o atributo de acesso public, a interface somente será acessível (visível) no **package** que a contém.

Exemplo

```java
public interface Shape {

        //implicitly public, static and final
        public String LABLE="Shape";

        //interface methods are implicitly abstract and public
        void draw();

        double getArea();
}
```

Exemplo

```java
public class Circle implements Shape {

        private double radius;

        public Circle(double r){
                this.radius = r;
        }

        @Override
        public void draw() {
                System.out.println("Drawing Circle");
        }

        @Override
        public double getArea(){
                return Math.PI*this.radius*this.radius;
        }

        public double getRadius(){
                return this.radius;
        }
}
```