Linguagem de Programação II

Métodos e Variáveis de Classe e de Instância

Universidade do Estado do Rio de Janeiro-UERJ
Instituto de Matemática e Estatística-IME
Ciência da Computação
Professor: Alexandre Sztajnberg

Atributos Estáticos

Classe Carro

```
private static String cor ="Preto";

private int numPortas = 4;

public Carro(){};

public Carro (int np){this.numPortas = np;}

public void setCor(String novaCor)
        {this.cor = nvaCor;}

public String toString () {
   return "Cor: " + cor +
          " portas: " + numPortas;
}
```

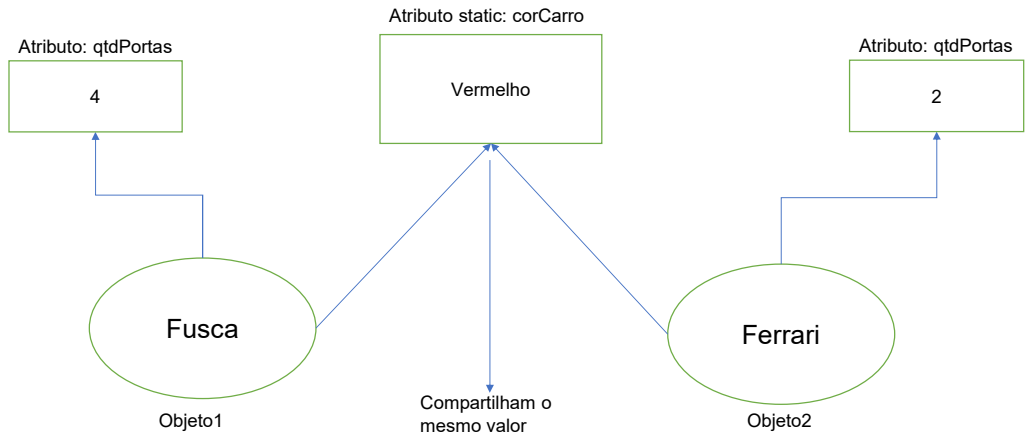
Objetos

```
Carro fusca = new Carro();
Carro ferrari = new Carro(2);

ferrari.setCor("Vermelho");

System.out.println(fusca);
System.out.println(ferrari);
```

Static em Atributos



Métodos Estáticos

- Os métodos estáticos não precisam de objetos para ser chamado, eles podem ser invocados através do nome da classe .
- Isto acontece pois os métodos estáticos são associados a classe como um todo diferente dos métodos não estáticos que são associados a instâncias de classes, os objetos.
- Métodos estáticos X Métodos não estáticos
  - Estáticos:
    - `NomeClasse.nomeMétodo();`
  - Não estáticos
    - `NomeClasse objeto = new NomeClasse();`
    - `Objeto.nomeMétodo();`

Código Métodos Estáticos

```
public class MetodoEstatico {
    public static void main(String[] args){
        Imprimir.imprimirNome("Alexandre");
    }
}
```

```
public class Imprimir {
    public static void imprimirNome(String nome){
        System.out.println("Imprimindo : " + nome);
    }
}
```

Resposta

```
Imprimindo : Alexandre
```

Não precisei instanciar um objeto somente usei o nome da classe seguido de um ponto para chamar o método estático

Variáveis Final

- As variáveis declaradas como final não podem ter seus valores trocados até o final da execução do programa, elas passam a ser constantes
- Os rótulos de variáveis constates em CamelCase são definidos totalmente em maiúsculo.
- Sua inicialização poderá ser feita junto a declaração ou em métodos construtores (blank FINAL variable)
  - Se uma classe possuir vários métodos construtores, o atributo FINAL deverá ser inicializado em todos os métodos construtores.
- Caso sejam aplicadas a um vetor ou objeto apenas o nome destes ficam fixos mas seu conteúdo ainda pode ser mudado, ou seja, o final só impede que estes sejam instanciados novamente
- Podem ser usados nos parâmetros de métodos.

Código da Variável Final

```
public class VariavelFinal {
  public static void main(String[] args){
    CalculoArea calculo = new CalculoArea();
         calculo.PI = 5.68;
     }
}
```

```
public class CalculoArea {
    private final Double PI = 3.141592;
    public double calculoArea(Double raio){
        Double resp = (4/3)*PI* (Math.pow(raio,
2));
        return resp;
    }
}
```

Resposta

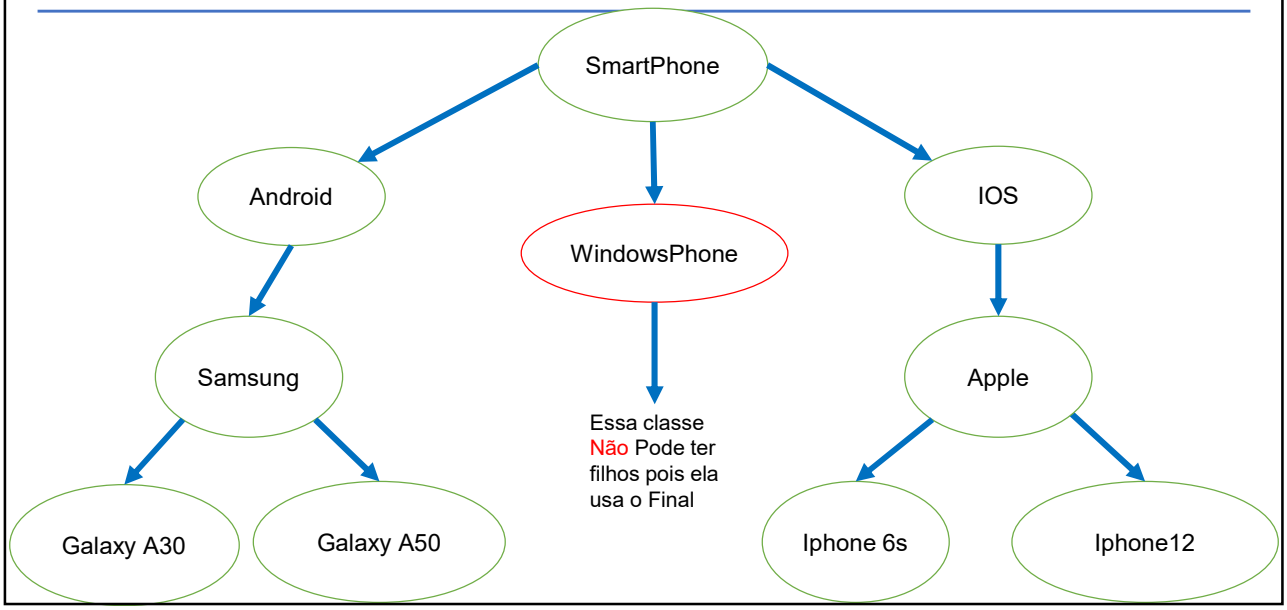
```
location: class VariavelFinal
VariavelFinal.java:4: error: PI has private access in
CalculoArea
         calculo.PI = 5.68;
                  ^
```

Deu um erro ao tentarmos modificar o valor de PI pois ele é uma variável final

Classes Final

- Uma classe definida com final significa que sua herança chegou ao fim, ou seja, não existem outras classes não podem se estender a ela.
- Ela é considerada uma classe folha, ou seja não possui filhos.
- Ela é considerada o grau máximo de especialização de classes não podendo ser mais especializada.
- O final pode ser usado para segurança, pois ao usa-lo sabemos que quando um objeto for criado ele pertence aquela classe e não a possíveis classes herdeiras
- Um exemplo de classe do pacote java.útil que usa o final é a classe String.

Classe Final



Código Classe final

```
public final class WindowsPhone extends Smartphone
 {
  public void vibrar(){
    System.out.println("WindownsPhone Vibrando");
    }
}
```

```
public class ClasseFinal {
  public static void main(String[] args){
      WindowsPhone w =new WindowsPhone();
      w.vibrar();
    }
}
```

Resposta

```
WindownsPhone Vibrando
```

O código roda sem erros

Código Classe final

```
public class LG extends WindowsPhone{
  @Override
  public void vibrar(){
     System.out.println("LG Vibrando");
    }
}
```

```
public class ClasseFinal {
  public static void main(String[] args){
       LG lg = new LG();

    }
}
```

Resposta

```
.\LG.java:1: error: cannot inherit from final WindowsPhone
public class LG extends WindowsPhone{
                        ^
1 error
```

O código Apresenta um erro ao fazermos herança da classe WindowsPhone para a classe LG

Métodos Final

- Os métodos declarados com final não podem ser sobre escritos por classes herdeiras, ou seja eles são declarados e não poderão mais mudar seu conteúdo.
- Essa propriedade confere segurança ao código escrito pois evita possíveis erros de sobre escrita, além de evitar que classes maliciosas sobre escrevam dados que são tratados nestes métodos
- Outra vantagem é o desempenho de execução, dado que as chamadas a métodos Final são substituídas por suas definições, isto é, pelo código contido na definição do método (técnica de inclusão de código inline).

Código do método final

```
public class MetodoFinal {
 public static void main(String[] args) {
        Smartphone a30 = new Smartphone();
        a30.tocar();
    }
}
```

```
public class Smartphone {
  public final void tocar(){
    System.out.println("tocando plim plim");
    }
}
```

Resposta

```
tocando plim plim
```

O código roda sem erros

Código do método final

```
public class MetodoFinal {
 public static void main(String[] args) {
        Smartphone a30 = new Smartphone();
        a30.tocar();
    }
}
```

```
public class Smartphone {
  public final void tocar(){
    System.out.println("tocando plim plim");
    }
}
```

Resposta

```
.\Samsung.java:3: error: tocar() in Samsung cannot override tocar() in
Smartphone
    public void tocar(){
                ^
  overridden method is final
1 error
```

O código Teve um erro pois o método é final e não pode ser sobreescrito.