

Programação Orientada a Objetos

Prof. Paulo Henrique Pisani

<http://professor.ufabc.edu.br/~paulo.pisani/>

Tópicos

- Classes
- Objetos
- Mensagens
- Encapsulamento
- UML

Conceitos gerais

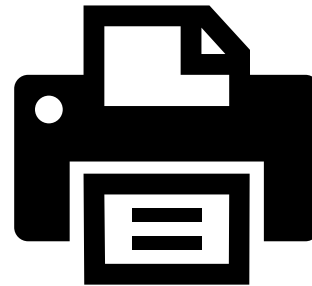
Motivação

- Tenta solucionar problemas da programação estruturada (PE):
 - Em PE, é difícil criar uma conexão forte entre dados e funcionalidades;
 - Manutenção, Produtividade, Reuso de código.
- Exemplo:
 - Validação de campos;
 - Acesso a diferentes gerenciadores de banco de dados.























Programação Orientada a Objetos



- Paradigma de programação baseado no princípio de “imitar o mundo real”:
 - Nossa vida está repleta de **objetos físicos** (carro, livro, impressora, etc) e **abstratos** (pergunta, esquema tático, plano de aulas, etc);
 - Objeto é tudo aquilo que tem **atributos** e **comportamentos**.



Programação Orientada a Objetos

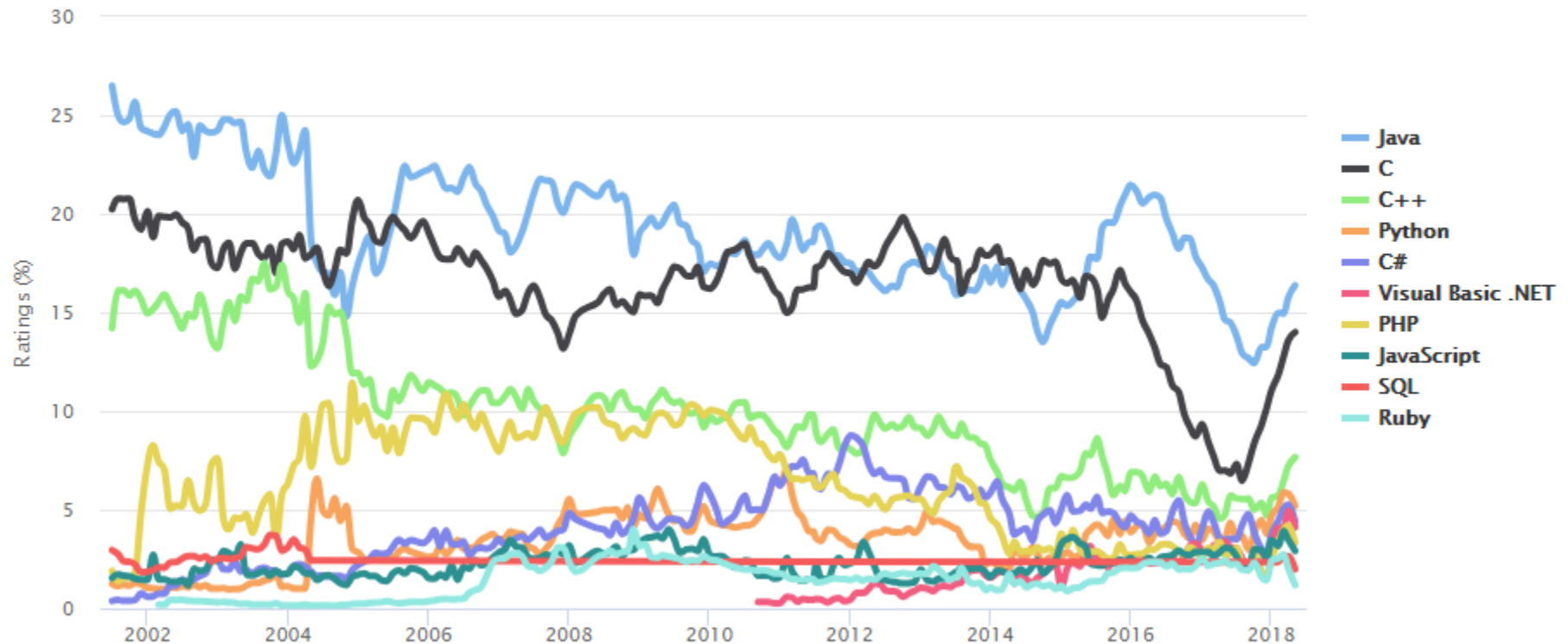
Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7

<https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages> (28/05/2018)

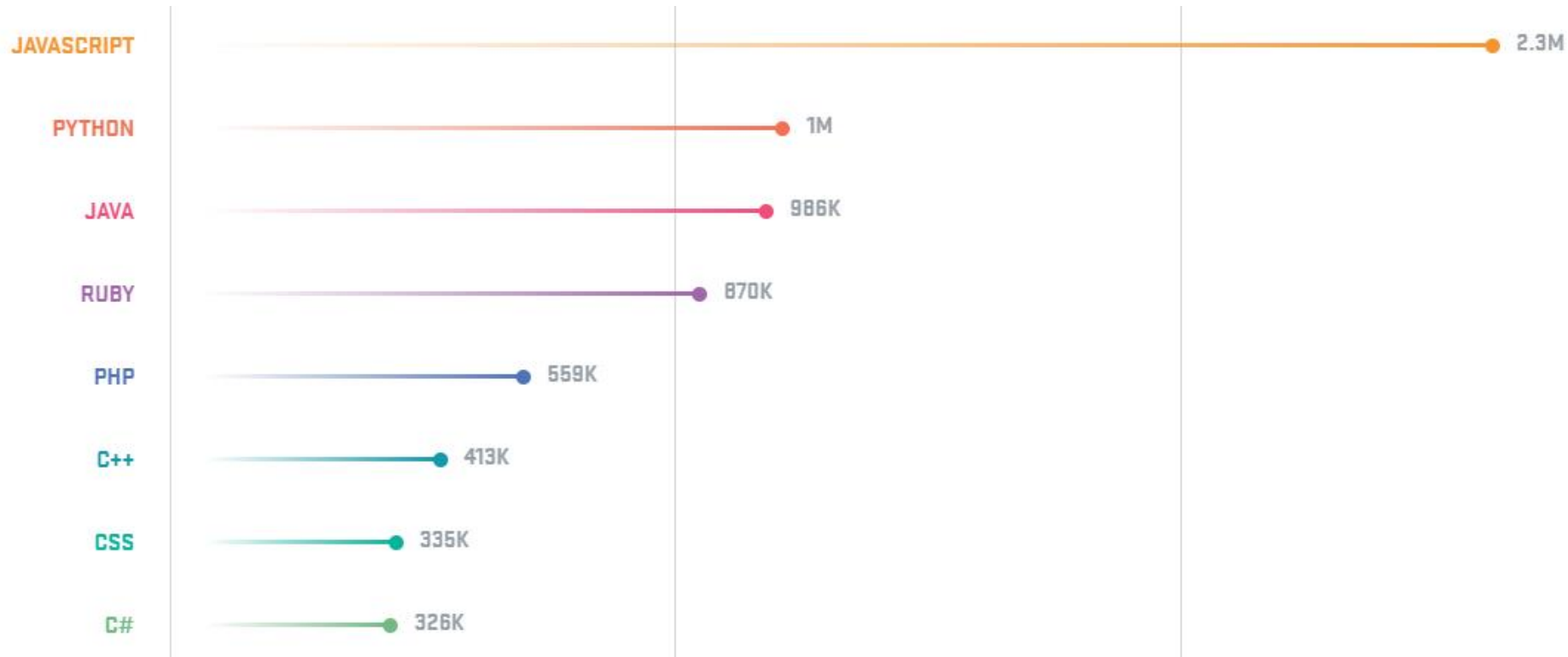
Programação Orientada a Objetos

TIOBE Programming Community Index

Source: www.tiobe.com



Programação Orientada a Objetos



Programação Orientada a Objetos (POO)

- Os programas são organizados por elementos chamados **classes**;
- Conceitos importantes em POO:
 - Abstração
 - Encapsulamento
 - Herança
 - Polimorfismo

Pilares da OO

Classe

- É uma **abstração** de um objeto:
 - É um modelo para criar um objeto;

Exemplo

```
public class DiscoVoador {  
  
}
```

Classe

- É uma **abstração** de um objeto:
 - É um modelo para criar um objeto;
 - Na abstração, desconsideramos detalhes irrelevantes.
- Uma classe **encapsula** comportamentos.

Exemplo

```
public class DiscoVoador {  
    public String cor;  
    public void darPartida() {  
        // Implementacao  
    }  
    public void irParaFrente(int metros) {  
        // Implementacao  
    }  
}
```

Classe

- É uma **abstração** de um objeto:
 - É um modelo para criar um objeto;
 - Na abstração, desconsideramos detalhes irrelevantes.
- Uma classe **encapsula** comportamentos.

Exemplo

```
public class DiscoVoador {  
    public String cor;  
    public void darPartida() {  
        // Implementacao  
    }  
    public void irParaFrente(int metros) {  
        // Implementacao  
    }  
}
```

Atributo

Métodos

UML (Unified Modeling Language)

- Linguagem de modelagem unificada: **linguagem visual** para **modelagem** de softwares utilizando orientação a objetos.
- Por que **modelar** software?
 - Planejamento da estrutura de um sistema antes de contruí-lo: prever evolução, manutenção.
 - Paralelo com a construção de uma casa:
 - Troca de fiação;
 - Manutenção em encanamentos.

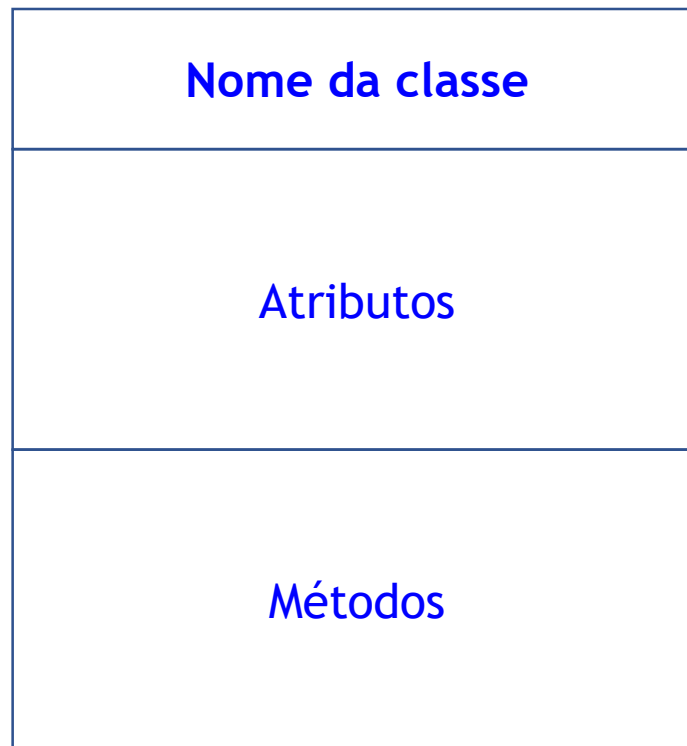
UML

- **A UML é composta por diversos diagramas:** casos de uso, classes, objetos, pacotes, sequência, comunicação, máquina de estados, atividade, componentes, implantação, tempo, estrutura composta, etc;
- Cada diagrama é usado para modelar o sistema sob uma determinada ótica.
- Neste curso, veremos o **diagrama de classes**.

UML - Diagrama de Classe

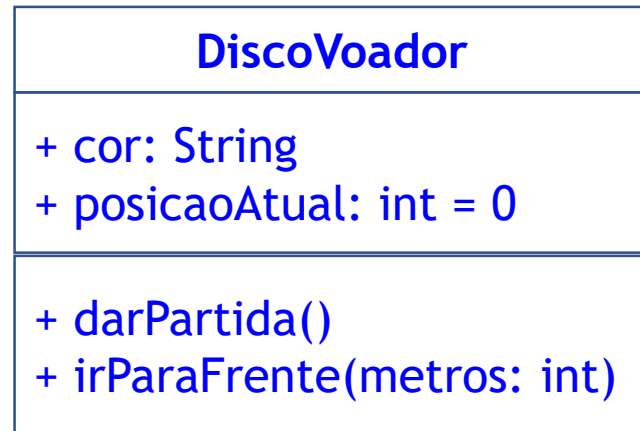
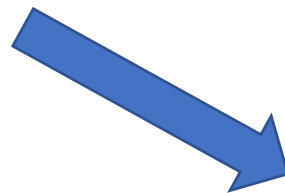
- Define a estrutura das classes adotadas pelo software: **atributos, métodos e relacionamentos**;
- Um dos mais utilizados e importantes diagramas da UML (serve como apoio para os demais diagramas).

UML - Diagrama de Classe



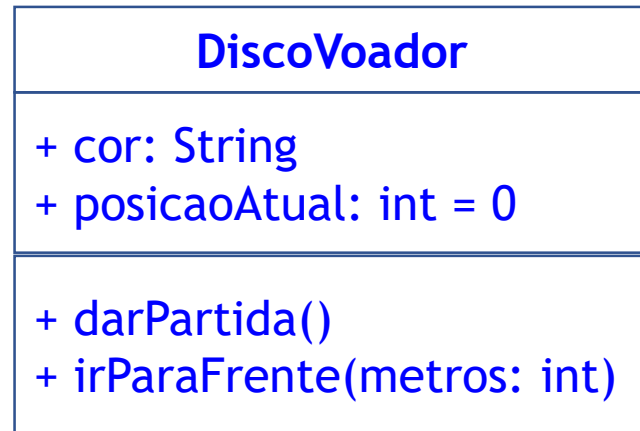
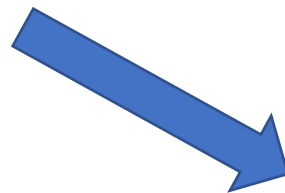
UML - Diagrama de Classe

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```



UML - Diagrama de Classe

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```

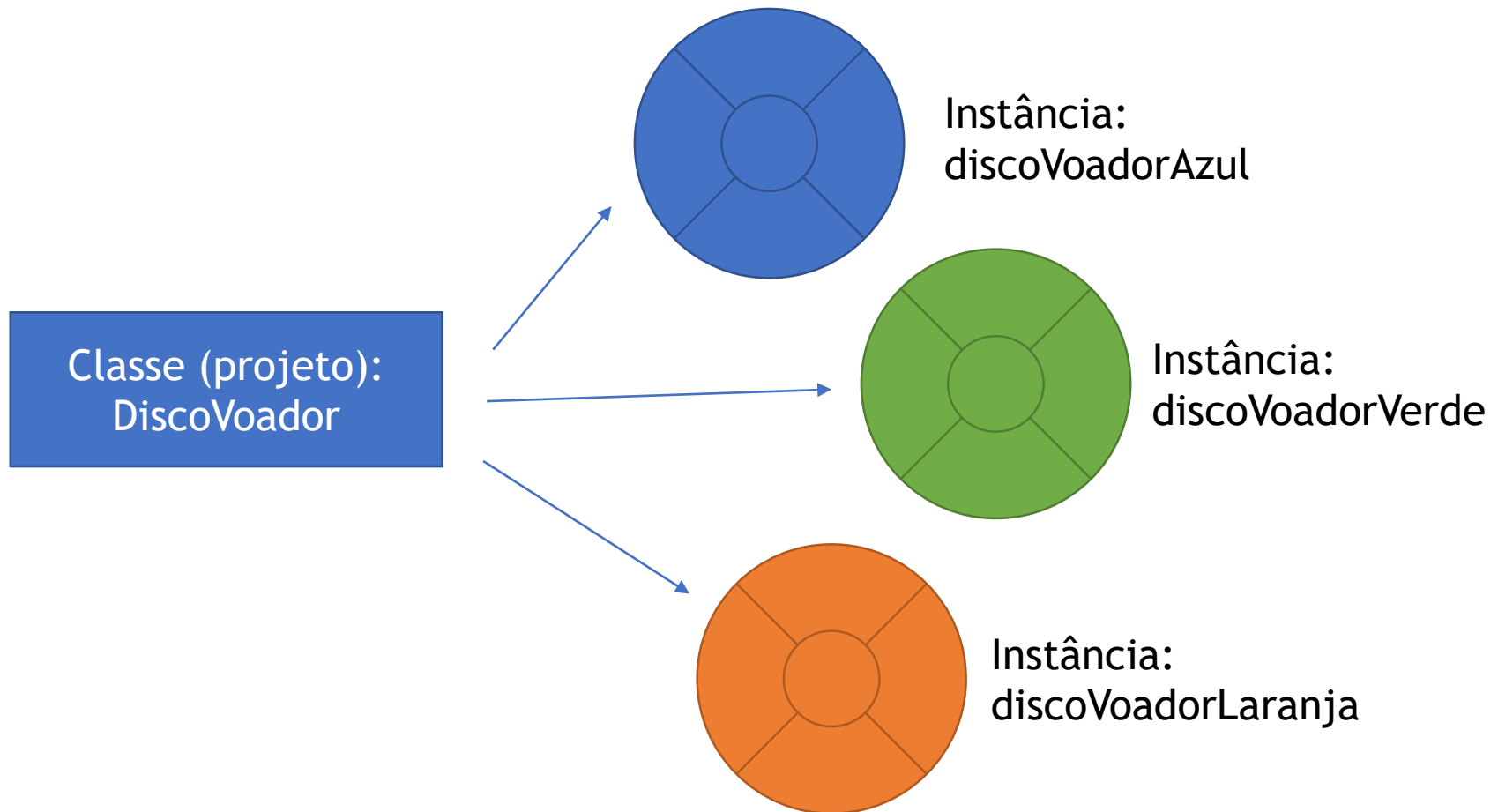


Abstração



Vamos instanciar um disco voador?

- Um objeto é uma **instância** de uma **classe**.



Vamos instanciar um disco voador? (agora no Java)

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```

Vamos instanciar um disco voador? (agora no Java)

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```

Importante! cada arquivo .java pode ter apenas uma classe pública (e o arquivo deve ter o nome da classe)

Vamos instanciar um disco voador? (agora no Java)

Arquivo: TesteObjetos.java

```
public class TesteObjetos {  
    public static void main(String[] args) {  
        DiscoVoador discoVoadorAzul = new DiscoVoador();  
        discoVoadorAzul.cor = "Azul";  
  
        DiscoVoador discoVoadorVerde = new DiscoVoador();  
        discoVoadorVerde.cor = "Verde";  
  
        DiscoVoador discoVoadorLaranja = new DiscoVoador();  
        discoVoadorLaranja.cor = "Laranja";  
    }  
}
```

new: Instancia um objeto da classe

Já instanciamos objetos antes...

```
Scanner leitor = new Scanner(System.in);
```

```
int[] vetor;  
vetor = new int[8];
```

```
double[] vetor2 = new double[10];
```

Tipos de dados (com Wrapper)

- Java possui classes para seus tipos primitivos



byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Tipos de dados (com Wrapper)

- Boxing:

```
Integer num1 = new Integer(10);  
Integer num2 = new Integer(2 * num1.intValue());  
System.out.println("num1="+num1.intValue() + "  
    num2="+num2.intValue());
```

- Autoboxing:

```
Integer num1 = 10;  
Integer num2 = 2 * num1;  
System.out.println("num1="+num1 + " num2="+num2);
```

Atributos

- Dados que diferenciam um objeto de outro (e.g. cor)

Arquivo: TesteObjetos.java

```
public class TesteObjetos {  
    public static void main(String[] args) {  
        DiscoVoador discoVoadorAzul = new DiscoVoador();  
        discoVoadorAzul.cor = "Azul";  
  
        DiscoVoador discoVoadorVerde = new DiscoVoador();  
        discoVoadorVerde.cor = "Verde";  
  
        DiscoVoador discoVoadorLaranja = new DiscoVoador();  
        discoVoadorLaranja.cor = "Laranja";  
    }  
}
```

Atributos

```
Conta contaCorrente = new Conta();  
contaCorrente.nomeTitular = "Fulano";  
contaCorrente.agencia = 200;  
contaCorrente.numeroConta = 6000;
```

```
DiscoVoador disco = new DiscoVoador();  
disco.capacidade = 40;  
disco.peso = 8000;  
disco.dono = "Alien";  
disco.preco = 1800.50;
```

Métodos

- Realizam instruções relacionadas ao objeto.

Arquivo: TesteObjetos.java

```
public class TesteObjetos {  
    public static void main(String[] args) {  
        DiscoVoador discoVoadorAzul = new DiscoVoador();  
        discoVoadorAzul.cor = "Azul";  
        discoVoadorAzul.darPartida();  
  
        DiscoVoador discoVoadorVerde = new DiscoVoador();  
        discoVoadorVerde.cor = "Verde";  
        discoVoadorVerde.darPartida();  
    }  
}
```

Métodos

```
Conta contaCorrente = new Conta();  
contaCorrente.depositar(500);  
contaCorrente.depositar(100);  
double saldo = contaCorrente.getSaldo();
```

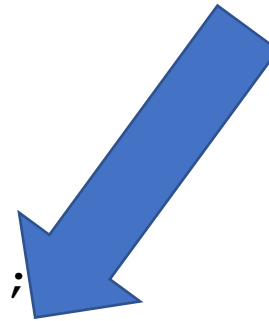
```
Pilha pilha1 = new Pilha();  
pilha1.empilha(4);  
pilha1.empilha(8);  
pilha1.empilha(3);  
int valor = pilha1.desempilha();  
valor = pilha1.desempilha();
```

Métodos

Mensagens -> Chamadas de métodos no Java

```
Conta contaCorrente = new Conta();  
contaCorrente.depositar(500);  
contaCorrente.depositar(100);  
double saldo = contaCorrente.getSaldo();
```

```
Pilha pilha1 = new Pilha();  
pilha1.empilha(4);  
pilha1.empilha(8);  
pilha1.empilha(3);  
int valor = pilha1.desempilha();  
valor = pilha1.desempilha();
```



Palavra-chave **this**

this

- Ponteiro para o próprio objeto;
- Apenas **métodos de instância** (não static) podem usá-lo.

Discutiremos este tópico mais a frente no curso...

this

- Voltando ao exemplo do disco voador:

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```

posicaoAtual é um membro de instância de DiscoVoador -> podemos usar this

this

- Voltando ao exemplo do disco voador:

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        this.posicaoAtual += metros;  
    }  
}
```

posicaoAtual é um membro de instância de DiscoVoador -> podemos usar this

this

- Pode ser útil para lidar com parâmetros que possuem o mesmo nome de atributos:

```
public class DiscoVoador {  
  
    public String cor;  
  
    public void mudarCor(String cor) {  
        this.cor = cor;  
    }  
}
```

Modificadores de acesso e Métodos modificadores

Modificadores de acesso

- Determinam a visibilidade de atributos, métodos e classes (ou seja, define quem podem acessar/modificar os atributos e utilizar os métodos):
 - `private`
 - `protected` Veremos na aula de herança...
 - `public`
 - Sem modificador (package)

Modificadores de acesso

	Própria classe	Pacote	Subclasses	“Mundo”
public	S	S	S	S
protected	S	S	S	N
(sem modificador)	S	S	N	N
private	S	N	N	N

<https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>

Modificadores de acesso

- Todos os atributos de DiscoVoador são public.

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```

Modificadores de acesso

- Todos os atributos de DiscoVoador são public.

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```



```
DiscoVoador disco = new DiscoVoador();  
disco.irParaFrente(300);  
disco.irParaFrente(200);  
disco.posicaoAtual = 100;  
disco.posicaoAtual = 2000;
```


Modificadores de acesso

- Todos os atributos de DiscoVoador são public.

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    public int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```

Atributo não está
devidamente encapsulado!



```
DiscoVoador disco = new DiscoVoador();  
disco.irParaFrente(300);  
disco.irParaFrente(200);  
disco.posicaoAtual = 100;  
disco.posicaoAtual = 2000;
```

Modificadores de acesso

- Todos os atributos de DiscoVoador são public.

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    private int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```



```
DiscoVoador disco = new DiscoVoador();  
disco.irParaFrente(300);  
disco.irParaFrente(200);  
disco.posicaoAtual = 100;  
disco.posicaoAtual = 2000;
```

Não compila!

Modificadores de acesso

- Todos os atributos de DiscoVoador são public.

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    public String cor;  
    private int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
}
```

Não compila! Assim evita que o programador altere atributos de forma incorreta



```
DiscoVoador disco = new DiscoVoador();  
disco.irParaFrente(300);  
disco.irParaFrente(200);  
disco.posicaoAtual = 100;  
disco.posicaoAtual = 2000;
```

Modificadores de acesso

- Todos os atributos de DiscoVoador são public.

Arquivo: DiscoVoador.java

```
public class DiscoVoador {
```

```
    public String cor;
```

```
    private
```

```
    public
```

```
}
```

```
public
```

```
    posicaoAtual += metros;
```

```
}
```

```
}
```

Não compila! Assim evita que o programador altere atributos de

Uso correto dos conceitos de POO ajuda a evitar erros de programação!

```
DiscoVoador disco = new DiscoVoador();  
disco.irParaFrente(300);  
disco.irParaFrente(200);  
disco.posicaoAtual = 100;  
disco.posicaoAtual = 2000;
```

Modificadores de acesso

- Todos os atributos de DiscoVoador são public.

Arquivo: DiscoVoador.java

```
public class DiscoVoador {
```

Não compila! Assim evita que o programador altere atributos de

Por exemplo, poderia haver alguma restrição de distância percorrida (que estaria sendo ignorada com a atribuição direta)

```
    posicaoAtual += metros;

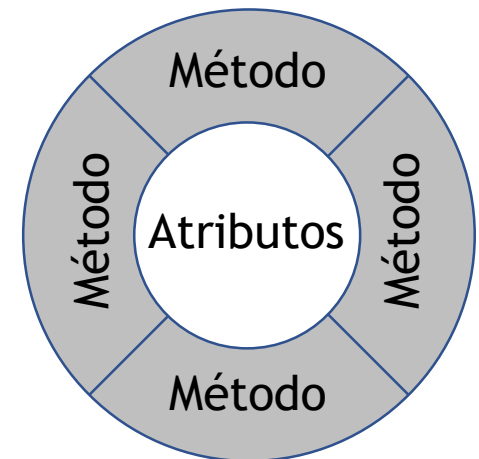
}

    public void irParaFrente(int metros) {
        if ((metros >= 0) && (metros <= 100))
            posicaoAtual += metros;
    }
}
```

~~disco.posicaoAtual = 100;~~
~~disco.posicaoAtual = 2000;~~

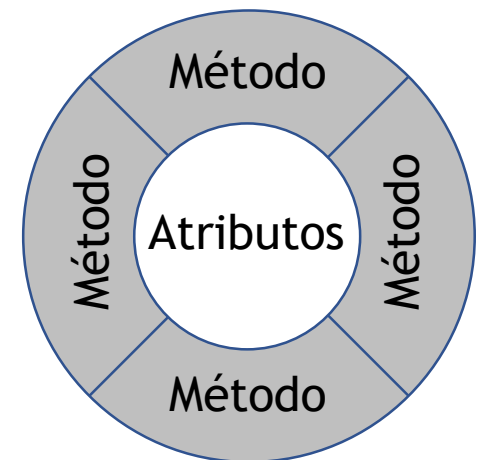
Métodos modificadores (get/set)

- Todos os atributos são modificados apenas por métodos da classe:
 - Portanto, todos os atributos deixam de ser públicos. **Fortalece o encapsulamento**
- Vantagens:
 - Melhora a legibilidade do código;
 - Facilita a manutenção e reutilização.



Métodos modificadores (get/set)

- Adotam o nome:
 - `get<nome_atributo>`
 - `set<nome_atributo>`
- É uma boa prática que qualquer atribuição utilize os métodos modificadores:
 - Inclusive os métodos da própria classe.



Métodos modificadores (get/set)

- Aplicando no disco voador...

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    private String cor;  
    private int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
    public void setCor(String cor) {  
        this.cor = cor;  
    }  
    public String getCor() {  
        return this.cor;  
    }  
}
```


Métodos modificadores (get/set)

- Em UML

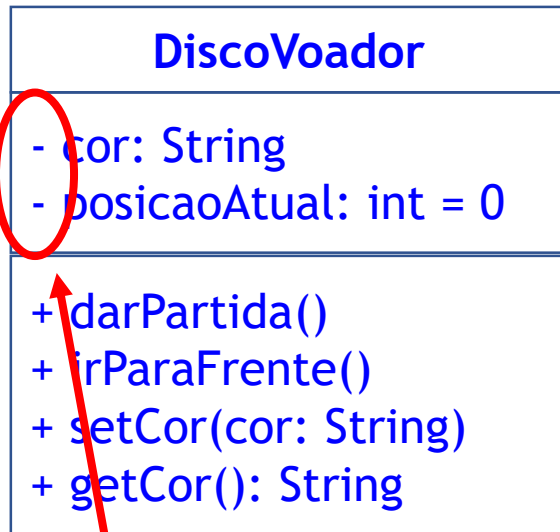
DiscoVoador
- cor: String - posicaoAtual: int = 0
+ darPartida() + irParaFrente() + setCor(cor: String) + getCor(): String

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    private String cor;  
    private int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
    private void setCor(String cor) {  
        this.cor = cor;  
    }  
    private String getCor() {  
        return this.cor;  
    }  
}
```

Métodos modificadores (get/set)

- Em UML



private

Arquivo: `DiscoVoador.java`

```
public class DiscoVoador {
    private String cor;
    private int posicaoAtual = 0;
    public void darPartida() {
        System.out.println("Motor ligado!");
    }
    public void irParaFrente(int metros) {
        posicaoAtual += metros;
    }
    private void setCor(String cor) {
        this.cor = cor;
    }
    public String getCor() {
        return this.cor;
    }
}
```

Métodos modificadores (get/set)

- Com métodos modificadores, podemos adicionar validações (que acabam sendo aplicadas a qualquer parte do programa que altere o atributo);
- Por exemplo, no DiscoVoador, é possível atribuir qualquer String a cor:
 - Contudo, um DiscoVoador, pode ser “Cinza” ou “Verde” apenas.

Métodos modificadores (get/set)

Arquivo: DiscoVoador.java

```
public class DiscoVoador {  
    private String cor;  
  
    ...  
  
    public void setCor(String cor) {  
        if ((cor == "Cinza") || (cor == "Verde")) {  
            this.cor = cor;  
        } else {  
            System.out.println("Cor invalida!.");  
        }  
    }  
    public String getCor() {  
        return this.cor;  
    }  
}
```

Métodos podem ser **private** também? Sim!

- Por exemplo, um método usado apenas pela classe não precisa ser exposto com **public**.

Método private para validar cor

```
private boolean validaCor(String cor) {  
    return (cor == "Cinza") || (cor == "Verde");  
}  
  
public void setCor(String cor) {  
    if (validaCor(cor)) {  
        this.cor = cor;  
    } else {  
        System.out.println("Cor invalida!.");  
    }  
}
```

Pacotes

Pacotes

- É um grupo de tipos relacionados (classes, interfaces, enumeradores);
- Usado para organizar tipos, evitar conflitos de nomeação, controlar acesso.
- Exemplos de pacotes:
 - java.lang
 - java.util (onde está a classe Scanner)
 - weka.core (pacote para aprendizado de máquina: <https://www.cs.waikato.ac.nz/ml/weka/>)
 - org.deeplearning4j: <https://deeplearning4j.org/>
 - org.apache.commons: <https://commons.apache.org>

Pacotes

- Vamos implementar um pacote chamado **estruturadados**;
 - Nomes de pacotes geralmente possuem todos os caracteres minúsculos;
- No pacote, haverá uma classe chamada **ListaFacil**.

Classe ListaFacil

Arquivo: `estruturasdedados/ListaFacil.java`

```
package estruturasdedados;
```

```
public class ListaFacil {  
  
    public void iniciaLista(int tamanhoMax) {  
        ...  
    }  
  
    public void adicionarItem(int novoItem) {  
        ...  
    }  
  
    public int lerItem(int indice) {  
        ...  
    }  
}
```

Usando a ListaFacil

Arquivo: TesteObjetos.java

```
import estruturasdedados.ListaFacil;

public class TesteObjetos {

    public static void main(String[] args) {
        ListaFacil lista = new ListaFacil();
        lista.iniciaLista(3);
        lista.adicionarItem(9);
        lista.adicionarItem(10);
        lista.adicionarItem(11);
        lista.adicionarItem(12); // Erro
        System.out.println(lista.lerItem(0));
        System.out.println(lista.lerItem(1));
        System.out.println(lista.lerItem(2));
        System.out.println(lista.lerItem(3)); // Erro
    }
}
```

Pacotes

- Vamos implementar outro pacote: **espaco**;
- Neste pacote, haverá duas classes:
DiscoVoador e **PortaDiscosVoadores**.

DiscoVoador

Arquivo: `espaco/DiscoVoador.java`

```
package espaco;
```

```
public class DiscoVoador {  
    private String cor;  
    private int posicaoAtual = 0;  
    public void darPartida() {  
        System.out.println("Motor ligado!");  
    }  
    public void irParaFrente(int metros) {  
        posicaoAtual += metros;  
    }  
    public void setCor(String cor) {  
        this.cor = cor;  
    }  
    public String getCor() {  
        return this.cor;  
    }  
}
```

PortaDiscosVoadores

```
package espaco;
```



Aqui NÃO precisa de import espaco.DiscoVoador,
pois estamos no mesmo pacote!

```
public class PortaDiscosVoadores {  
    private DiscoVoador[] discos = new DiscoVoador[3];  
  
    public void estacionarDisco(DiscoVoador disco) {  
    }  
  
    public DiscoVoador liberarDisco(int indice) {  
  
    }  
}
```

PortaDiscosVoadores

```
private int encontraIndiceLivre() {  
    for (int i = 0; i < discos.length; i++)  
        if (discos[i] == null)  
            return i;  
    return -1;  
}  
  
public void estacionarDisco(DiscoVoador disco) {  
    int i = encontraIndiceLivre();  
    if (i == -1) {  
        System.out.println("Nao ha vagas!");  
        return;  
    }  
    discos[i] = disco;  
    disco.estacionado = true;  
}
```

Usando o pacote “espaco”

Importa todas as classes do pacote “espaco”.

```
import espaco.*;
```

```
public class TesteObjetos {
```

```
    public static void main(String[] args) {
```

```
        DiscoVoador discoVoadorAzul = new DiscoVoador();
```

```
        discoVoadorAzul.setCor("Azul");
```

```
        System.out.println(discoVoadorAzul);
```

```
        PortaDiscosVoadores portaDiscos = new PortaDiscosVoadores();
```

```
        portaDiscos.estacionarDisco(discoVoadorAzul);
```

```
        System.out.println(portaDiscos.liberarDisco(0));
```

```
    }
```

```
}
```


Pacotes com javac

- Cada pacote fica em uma pasta/diretório:

/TesteObjetos.java

/estruturadedados/ListaFacil.java

/espaco/DiscoVoador.java

/espaco/PortaDiscosVoadores.java

- Compilação de classes em pacote:

Somente pacote estruturas de dados:

javac estruturadedados/ListaFacil.java

Somente pacote espaco:

javac espaco/*.java

Projeto todo:

javac *.java

Modificadores de acesso (em pacotes)

- Relembrando...

	Própria classe	Pacote	Subclasses	“Mundo”
public	S	S	S	S
protected	S	S	S	N
(sem modificador)	S	S	N	N
private	S	N	N	N

<https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>

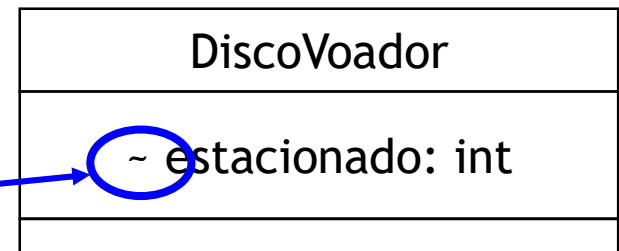
Modificadores de acesso (em pacotes)

- Vamos adicionar um atributo **boolean** **estacionado** (sem modificador de acesso) na classe DiscoVoador;

Arquivo: `espaco/DiscoVoador.java`

```
package espaco;  
public class DiscoVoador {  
    boolean estacionado = false;  
  
    ...  
}
```

package



Modificadores de acesso (em pacotes)

- A partir da classe PortaDiscosVoadores, vamos alterar o valor de **estacionado**.

Arquivo: `espaco/PortaDiscosVoadores.java`

```
package espaco;  
public class PortaDiscosVoadores {  
    ...  
  
    public void estacionarDisco(DiscoVoador disco) {  
        disco.estacionado = true;  
        ...  
    }  
    ...  
}
```

Modificadores de acesso (em pacotes)

- A partir do programa principal não conseguimos alterar o valor de estacionado!

Arquivo: TesteObjetos.java

```
public class TesteObjetos {  
    public static void main(String[] args) {  
        DiscoVoador discoVoadorAzul = new DiscoVoador();  
        discoVoadorAzul.estacionado = true;  
    }  
}
```

Não compila!

Modificadores de acesso (em pacotes)

- A partir do programa principal não conseguimos alterar o valor de estacionado!

Arquivo: TesteObjetos.java

```
public class  
    public
```

Qual o pacote da classe
TesteObjetos???

```
oVoador() ;
```

```
}
```

```
}
```

Modificadores de acesso (em pacotes)

- A partir do programa principal não conseguimos alterar o valor de estacionado!

Arquivo: TesteObjetos.java

```
public class  
    public
```

Pacote *default*

```
oVoador () ;
```

```
}
```

```
}
```

Modificadores de acesso

- Os modificadores de acesso contribuem para o **encapsulamento**:
 - Aumentam o controle sobre os atributos da classe;
 - Escondem detalhes (desnecessários) de como as classes são implementadas (atributos e métodos);
 - **Evitam erros dos programadores!**

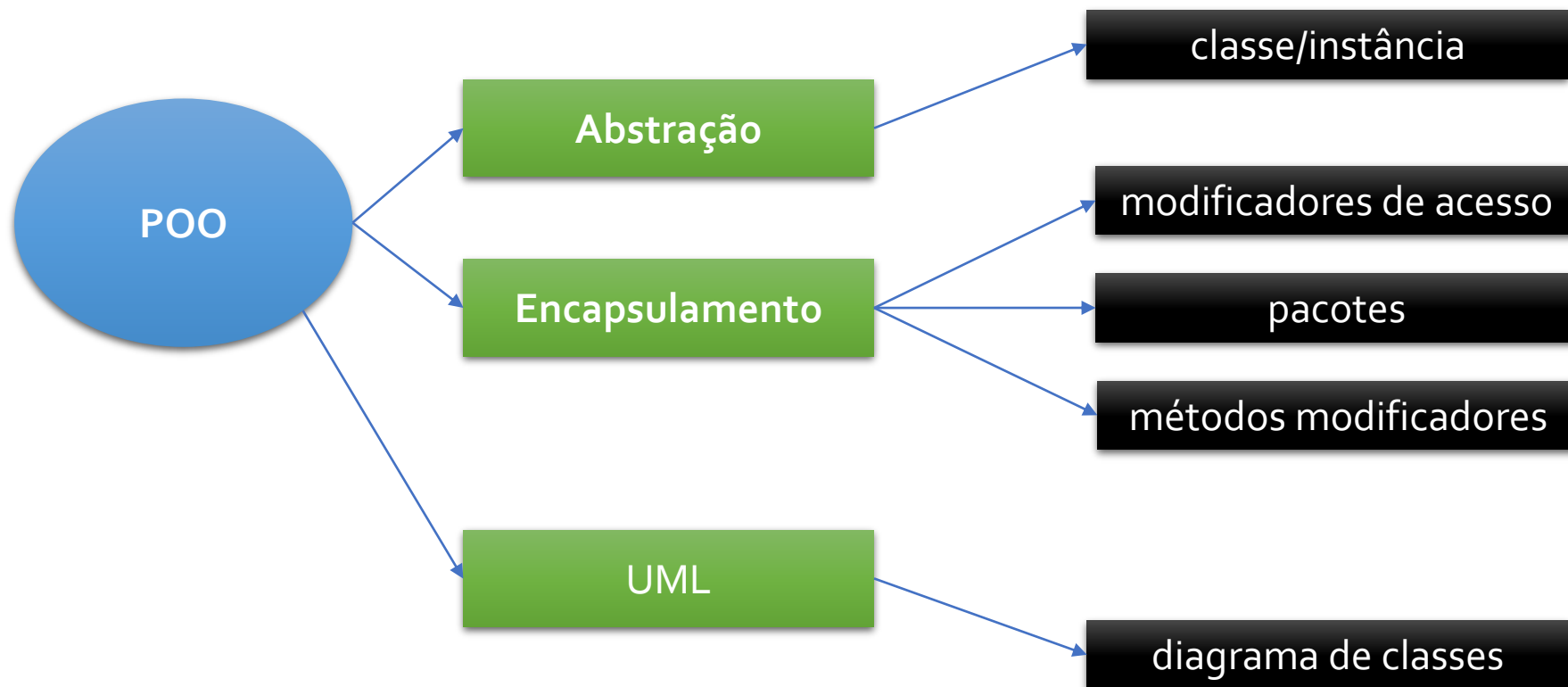
Exercício Extra 1

- Criar uma ListaFacil de DiscosVoador e usá-la em PortaDiscosVoadores.

Exercício Extra 2

- O professor ABC planejou montar um sistema para acompanhar a copa do mundo. Nesse sistema haverá algumas classes: TabelaCopaDoMundo, Jogo, Estadio;
- Modele essas classes em UML e implemente em Java:
 - Quais atributos e métodos cada classe possui?
 - Implemente classes adicionais se necessário.

Resumo da aula



Referências

- Slides do Prof. Monael P. Ribeiro:
<https://sites.google.com/site/poo2017q2/>
- Slides do Prof. Vladimir Rocha (aula de Conceitos Gerais)
- Java e Orientação a Objetos - FJ11, Caelum

Referências (projeto pedagógico)

- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML: guia do usuário. Rio de Janeiro, RJ: Campus, 2005.
- GUEDES, G. T. A. UML 2: uma abordagem prática. São Paulo, SP: Novatec, 2009.
- DEITEL, H. M.; DEITEL, P. J. Java: como programar. 6a edição. Porto Alegre, RS: Bookman, 2005.
- BARNES, D. J.; KOLLING, M. Programação orientada a objetos com Java. 4ª edição. São Paulo, SP: Editora Pearson Prentice Hall, 2009.

Referências (projeto pedagógico)

- FLANAGAN, D. Java: o guia essencial. 5ª edição. Porto Alegre, RS: Bookman, 2006.
- BRUEGGE, B.; DUTOIT, A. H. Object-oriented software engineering: using UML, patterns, and Java. 2ª edição. Upper Saddle River, NJ: Prentice Hall, 2003.
- LARMAN, C. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3ª edição. Porto Alegre, RS: Bookman, 2007.
- FOWLER, M. UML essencial: um breve guia para a linguagem-padrão de modelagem de objetos. 3ª edição. Porto Alegre, RS: Bookman, 2005.