SCC0504 - Programação Orientada a Objetos

Paradigmas, Motivação, Visão Geral de Java

Prof.: Leonardo Tórtoro Pereira

leonardop@usp.br

Paradigmas

Paradigmas de Programação [1]

➔ *"Um paradigma de programação é uma abordagem para programar um computador baseada em uma teoria matemática ou um conjunto coerente de princípios"*

-Peter Van Roy, *Programming Paradigms for Dummies: What Every Programmer Should Know.*

Paradigmas de Programação [2, 7]

- ➔ É um modelo mental, usado para entender o mundo (no caso, como computadores resolvem problemas)
- ➔ Um paradigma é um jeito de fazer algo, e não algo concreto. Portanto, **não** é certo falar algo como "*paradigma de linguagem de programação*".

Paradigmas de Programação [2]

- ➔ É comum que as linguagens tornem fácil de programar seguindo um paradigma específico, e por isso dizemos, por exemplo, que “Java é uma linguagem de programação orientada a objetos”.
- ➔ Mas **não** existe “*Paradigma de linguagem OO*”.

Paradigmas de Programação [1]

- ➔ Cada paradigma suporta um conjunto de conceitos que é ótimo para determinados problemas.
- ➔ Os problemas maiores muitas vezes são otimizados ao usarmos diversos paradigmas (multiparadigma).

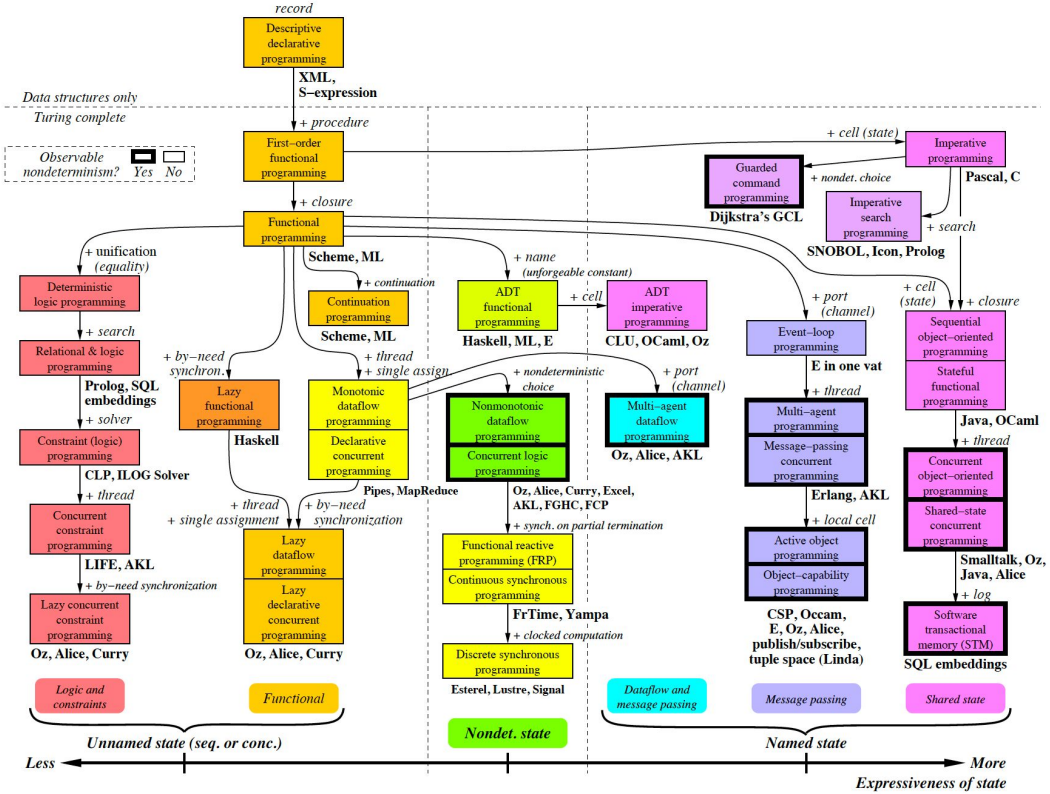


Figure 2. Taxonomy of programming paradigms

Taxonomia dos paradigmas de programação.
Fonte: Van Roy, "Programming Paradigms for Dummies," página 13, figura 2, em https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf.

Paradigmas Imperativos

Alguns dos Maiores Paradigmas [2]

- Programação Imperativa
  - Fluxo de controle é explícito:
    - Comandos mostram **como** a computação é feita, passo a passo
  - Cada passo afeta o **estado** global da computação

Alguns dos Maiores Paradigmas [2]

```
   result = []
    i = 0
start:
    numPeople = length(people)
    if i >= numPeople goto finished
    p = people[i]
    nameLength = length(p.name)
    if nameLength <= 5 goto nextOne
    upperName = toUpper(p.name)
    addToList(result, upperName)
nextOne:
    i = i + 1
    goto start
finished:
    return sort(result)
```

Alguns dos Maiores Paradigmas [2]

- Programação Estruturada
  - Tipo de programação imperativa
  - Fluxo de controle é definido por laços, condições e subrotinas aninhados ao invés de **gotos**
  - Geralmente, variáveis são locais aos seus blocos
  - Pascal, C, Algol 60...

Alguns dos Maiores Paradigmas [2]

```
result = [];
for i = 0; i < length(people); i++ {
    p = people[i];
    if length(p.name)) > 5 {
        addToList(result, toUpper(p.name));
    }
}
return sort(result);
```

Alguns dos Maiores Paradigmas [2]

- Programação Orientada a Objetos
  - Baseada no envio de mensagens a objetos
  - Objetos respondem realizando operações (métodos)
  - Mensagens podem ter argumentos
  - Simula-67, Smalltalk, C++, Java, C#, Ruby, Python

Alguns dos Maiores Paradigmas [2]

- Programação Orientada a Objetos
  - A ideia de existirem vários objetos, cada um com sua própria memória local e conjunto de operações, dá uma sensação bem diferente do processador monolítico e memória única compartilhada das linguagens não OO.

Alguns dos Maiores Paradigmas [2]

- Programação Orientada a Objetos
  - Um dos aspectos mais visíveis das linguagens com implementações mais puras do paradigma de OO é que condições e laços tornam-se mensagens, e seus argumentos são blocos de código executável

Alguns dos Maiores Paradigmas [2]

➔ O mesmo exemplo em *Smalltalk*:

```
result := List new.
people each: [:p |
  p name length greaterThan: 5 ifTrue: [result add (p name upper)]
]
result sort.
^result
```

➔ Que pode ser encurtado para:

```
^people filter: [:p | p name length greaterThan: 5] map: [:p | p name upper] sort
```

Alguns dos Maiores Paradigmas [2]

- Programação Orientada a Objetos
  - Muitas linguagens populares que são consideradas OO (como Java e C++) na realidade usam elementos de OO e misturam com código semelhante ao paradigma imperativo.

Alguns dos Maiores Paradigmas [2]

➔ *length* e *toUpper* são métodos ao invés de funções de alto-nível, mas o *for* e o *if* são estruturas de controle:

```
result = []
for p in people {
    if p.name.length > 5 {
        result.add(p.name.toUpper);
    }
}
return result.sort;
```

Paradigmas Declarativos

Alguns dos Maiores Paradigmas [2]

- Programação Declarativa
  - Fluxo de controle é implícito
  - É dito apenas como o resultado deve parecer,
    - E **não** como obtê-lo
  - Sem laços, atribuições, etc.
  - O motor que interpreta o código deve pegar a informação, usando a abordagem que desejar.

Alguns dos Maiores Paradigmas [2]

```
select upper(name)
from people
where length(name) > 5
order by name
```

Alguns dos Maiores Paradigmas [2]

- Programação Funcional
  - Fluxo de controle é feito combinando chamadas de funções
    - Ao invés de atribuir valores às variáveis
  - Foco em execução de séries de funções matemáticas
  - JavaScript, Haskell

Alguns dos Maiores Paradigmas [2]

```
sort(
  fix(λf. λp.
    if(equals(p, emptylist),
      emptylist,
      if(greater(length(name(head(p))), 5),
        append(to_upper(name(head(p))), f(tail(p))),
        f(tail(people)))))(people))
```

Alguns dos Maiores Paradigmas [3]

- Programação Lógica
  - Modelo abstrato de computação
  - Resolve problemas lógicos como quebra-cabeças, séries, etc.
  - Existe uma base de conhecimento que, quando alimentada de perguntas e outra base de conhecimento, produz resultados
  - Prolog

Alguns dos Maiores Paradigmas [3]

sum of two number in prolog:

predicados

```
sumoftwonumber(integer, integer)
```

cláusulas

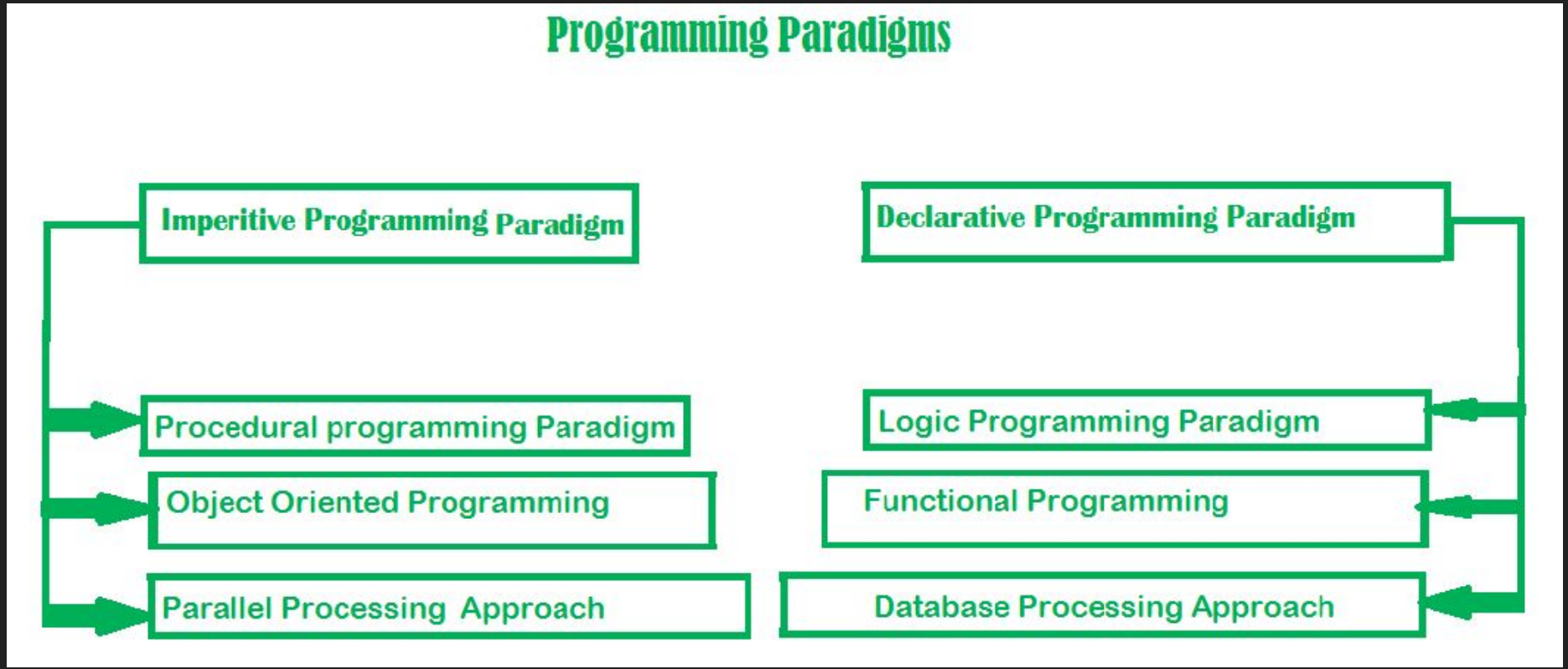
```
sum(0, 0).
 sum(n, r):-
      n1=n-1,
      sum(n1, r1),
      r=r1+n
```

Alguns dos Maiores Paradigmas [3]

- Orientada a Dados/Bases de Dados
  - Baseado em dados e sua movimentação
  - Definidos por dados ao invés de séries de passos
  - Funções de criação de arquivos, entrada, atualização e busca de dados
  - SQL

Alguns dos Maiores Paradigmas [3]

```
CREATE DATABASE databaseAddress;
CREATE TABLE Addr (
    PersonID int,
    LastName varchar(200),
    FirstName varchar(200),
    Address varchar(200),
    City varchar(200),
    State varchar(200)
);
```



Fonte:
https://www.geeksforgeeks.org/introduction-of-programming-paradigms/

O Paradigma Orientado a Objetos

O Paradigma Orientado a Objetos [4]

- Objetos do mundo real são vistos como entidades separadas com seus próprios estados
  - Modificados apenas por métodos (mensagens)
- Como objetos operam independentemente, são encapsulados em módulos
  - Contém ambientes e métodos locais
  - Comunicação com objetos é feita por mensagens

O Paradigma Orientado a Objetos [4]

- Objetos são organizados em classes
  - Recebem métodos e variáveis de suas classes
- Maiores benefícios são reusabilidade e extensibilidade de código
- Uma nova classe pode ser *derivada* de outra classe através da **herança**
  - Herda todas as características da classe *base*
    - Estrutura e comportamento

O Paradigma Orientado a Objetos [4]

- Além disso, as classes *derivadas* podem adicionar estados e comportamentos
- Também poder *sobrescrever* comportamentos de alguns métodos da classe *base*
  - Um método diferente para uma mesma mensagem
- A *herança* é permitida mesmo sem acesso ao código da classe *base*

O Paradigma Orientado a Objetos [6]

- No paradigma imperativo os dados são passivos e os procedimentos ativos
- No OO, dados são combinados com procedimentos para dar origem a objetos, que são ativos
  - Você “pede” para objeto fazer uma ação

Vantagens e Desvantagens de OO

Vantagens e Desvantagens de OO [4]

- A *herança* é um dos maiores benefícios da OO
- Reúso e extensão de códigos relativamente fáceis
  - Sem necessitar mudar o código fonte
- Modelar um programa como uma coleção de objetos de diferentes classes traz muita modularidade
  - Especialmente ao fazer classes serem extensões ou modificações de outras classes

Vantagens e Desvantagens de OO [4]

- Encapsulamento e ocultação de informação são inerentes ao paradigma
  - É recomendado que o estado de objetos seja acessado e manipulado apenas por seus métodos
  - Separa a interface (como objetos da classe são acessados) da implementação (o codigo com os métodos da classe)

Vantagens e Desvantagens de OO [10]

- ➔ É interessante quando vários programadores trabalham no mesmo projeto e não precisam entender cada componente
- ➔ Existe muito código que pode ser compartilhado e reusado
- ➔ É esperado que o projeto mude frequentemente e seja estendido com o tempo
- ➔ Seções diferentes podem se beneficiar de diferentes fontes como fontes de dados ou *hardware*

Vantagens e Desvantagens de OO [1, 3, 5, 10]

- ➔ Ideal para problemas com muitas abstrações de dados organizados em uma hierarquia
- ➔ Segurança dos dados
- ➔ Herança
- ➔ Flexível e com abstração
- ➔ Extremamente popular

Vantagens e Desvantagens de OO [10]

- Modular
  - Provê estrutura modular limpa
  - Bom para definir tipos de dados abstratos com detalhes de implementação ocultos
  - Unidade tem uma interface claramente definida

Vantagens e Desvantagens de OO [10]

- ➔ Escalável
  - Adicionar mais desenvolvedores no projeto é relativamente mais simples
  - Não precisam entender o código completo
    - Apenas a seção na qual vão trabalhar
  - Cada módulo pode ter diferentes recursos de *hardware*

Vantagens e Desvantagens de OO [10]

- ➔ Manutenibilidade
  - ◆ Fácil de manter e modificar o código existente
    - Novos objetos podem ser criados com pequenas diferenças dos já existentes

Vantagens e Desvantagens de OO [10]

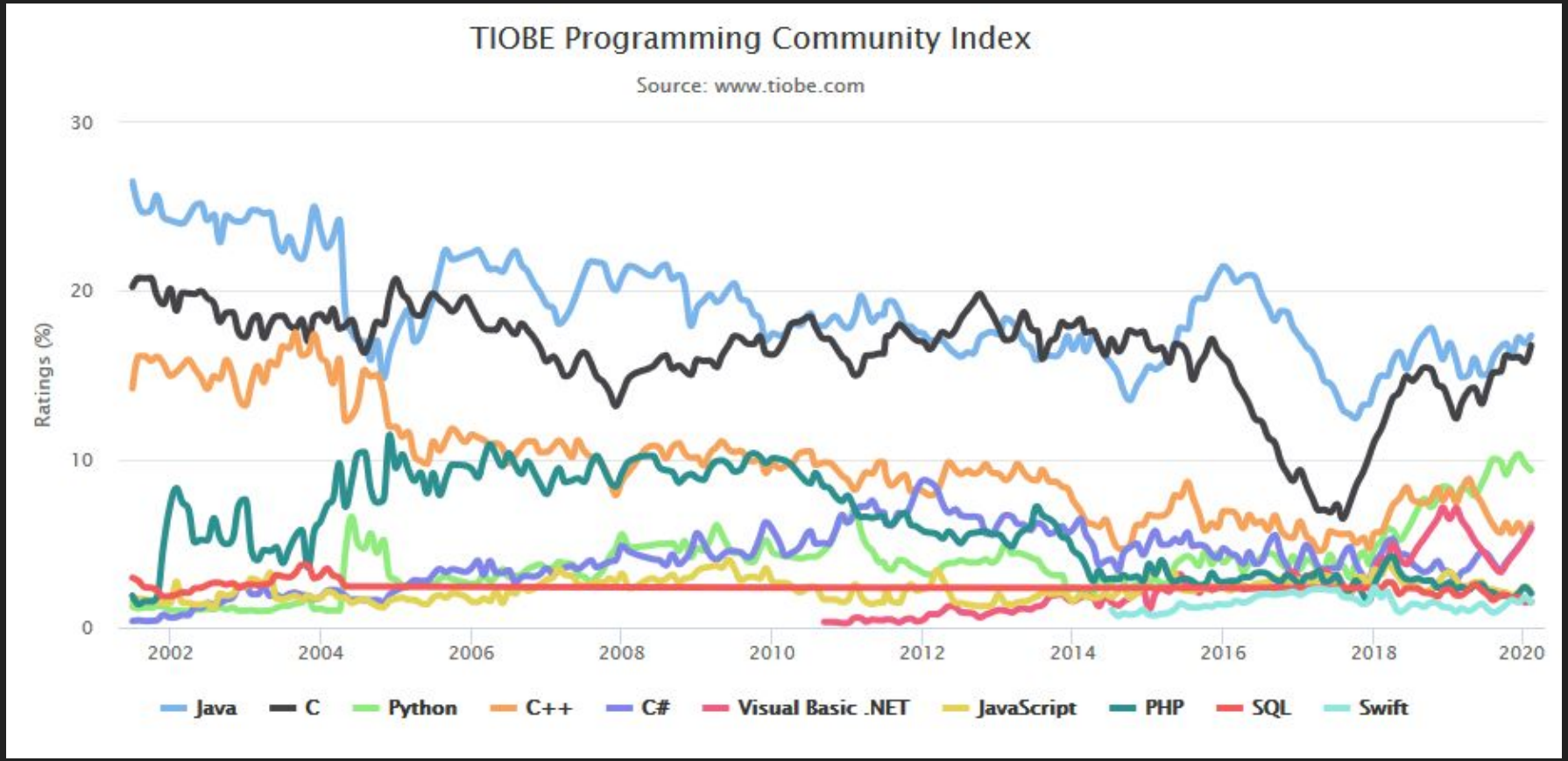
- ➔ Extensibilidade
  - ◆ Bom *framework* para estender projetos com bibliotecas
    - Componentes podem ser facilmente adaptados e modificados por um programador
  - ◆ Costuma ser muito útil em GUIs

Vantagens e Desvantagens de OO [10]

- Reusabilidade do código
  - Cada módulo trabalha independente dos seus vizinhos
  - Pode retirar uma seção e usar em outros projetos

Vantagens e Desvantagens de OO [1, 3, 5]

- ➔ Semântica complexa e difícil de pensar sobre
- ➔ Dificulta programação concorrente
- ➔ Menos eficiente que paradigma imperativo [8]
- ➔ Alguns estudos refutam uma maior produtividade entre POO e outras abordagens imperativas [9]
- ➔ Objetos podem interagir de maneiras complexas e não esperadas
- ➔ Não recomendado para projetos pequenos



Fonte: https://www.tiobe.com/tiobe-index/

Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0

Fonte: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019

Language Ranking: Trending

Rank	Language	Type	Score
1	Python	web, desktop, embedded	100.0
2	Java	web, mobile, desktop	94.9
3	C	mobile, desktop, embedded	91.4
4	C++	mobile, desktop, embedded	87.5
5	JavaScript	web	74.9
6	C#	web, mobile, desktop, embedded	74.1
7	Go	web, desktop	69.3
8	R	desktop	68.8
9	Ruby	web, desktop	67.1
10	Dart	web, mobile	67.1

Fonte: https://spectrum.ieee.org/ns/IEEE_TPL_2019/index/2019/1/1/1/1/1/5/1/75/1/50/1/100/1/50/1/75/1/75/1/20/1/50/1/40/1/50/

Language Ranking: Jobs

Rank	Language	Type	Score
1	Python		100.0
2	Java		97.5
3	C		96.0
4	C++		85.7
5	JavaScript		83.1
6	C#		77.2
7	HTML,CSS		75.6
8	Swift		69.4
9	Matlab		69.4
10	SQL		69.3

Fonte: https://spectrum.ieee.org/ns/IEEE_TPL_2019/index/2019/1/1/1/1/1/25/1/25/1/50/1/25/1/25/1/25/1/25/1/25/1/100/1/50/1/40/1/50/

Language Ranking: Open

Rank	Language	Type	Score
1	Python		100.0
2	Java		93.7
3	C		87.3
4	C++		85.6
5	JavaScript		85.3
6	C#		76.3
7	R		74.2
8	HTML,CSS		73.2
9	Swift		71.9
10	Dart		70.6

Fonte: https://spectrum.ieee.org/ns/IEEE_TPL_2019/index/2019/1/1/1/1/1/30/1/40/1/100/1/100/1/75/1/15/1/15/1/15/1/50/1/25/1/50/

Uma Introdução ao Código

O que um animal faz?

Abstração de um Animal [12]

```
abstract class Animal {
    // abstract methods
    abstract void move();
    abstract void eat();

    // concrete method
    void label() {
        System.out.println("Animal's data:");
    }
}
```

O que um pássaro faz?

O que um peixe faz?

Implementação das abstrações [12]

```
class Bird extends Animal {

    void move() {

        System.out.println("Moves by
        flying.");
     }
    void eat() {

        System.out.println("Eats
        birdfood.");
    }
}
```

```
class Fish extends Animal {
        void move() {

            System.out.println("Mov
            es by swimming.");
        }
        void eat() {

            System.out.println("Eat
            s seafood.");
        }
}
```

Testando... [12]

```
class TestBird {

        public static void
        main(String[] args) {

       Animal myBird = new Bird();


       myBird.label();

       myBird.move();

       myBird.eat();

    }

}
```

```
class TestFish {

        public static void
        main(String[] args) {

       Animal myFish = new Fish();


       myFish.label();

       myFish.move();

       myFish.eat();

    }

}
```

Resultado:

[Console output of TestBird]

Animal's data:

Moves by flying.

Eats birdfood.

[Console output of TestFish]

Animal's data:

Moves by swimming.

Eats seafood.

Como Proteger seus Dados?

Encapsulamento [12]

```
class Animal {
    private String name;
    private double averageWeight;
    private int numberOfLegs;

    // Getter methods
    public String getName() {
        return name;
    }
    public double getAverageWeight() {
        return averageWeight;
    }
    public int getNumberOfLegs() {
        return numberOfLegs;
    }
```

Encapsulamento [12]

```
    // Setter methods
     public void setName(String name) {
         this.name = name;
     }
     public void setAverageWeight(double averageWeight) {
         this.averageWeight = averageWeight;
     }
     public void setNumberOfLegs(int numberOfLegs) {
         this.numberOfLegs = numberOfLegs;
     }
}
```

Testando... [12]

```
public class TestAnimal {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();

        myAnimal.setName("Eagle");
        myAnimal.setAverageWeight(1.5);
        myAnimal.setNumberOfLegs(2);

        System.out.println("Name: " + myAnimal.getName());
        System.out.println("Average weight: " + myAnimal.getAverageWeight() + "kg");
        System.out.println("Number of legs: " + myAnimal.getNumberOfLegs());
    }
}
```

Resultado [12]

[Console output of TestAnimal]

Name: Eagle

Average weight: 1.5kg

Number of legs: 2

Herança

Herança [12]

```java
class Bird {
    public String reproduction = "egg";
    public String outerCovering = "feather";

    public void flyUp() {
        System.out.println("Flying up...");
    }
    public void flyDown() {
        System.out.println("Flying down...");
    }
}
```

```java
class Eagle extends Bird {

    public String name = "eagle";

    public int lifespan = 15;

}
```

Testando... [12]

```
class TestEagle {
    public static void main(String[] args) {
        Eagle myEagle = new Eagle();

        System.out.println("Name: " + myEagle.name);
        System.out.println("Reproduction: " + myEagle.reproduction);
        System.out.println("Outer covering: " + myEagle.outerCovering);
        System.out.println("Lifespan: " + myEagle.lifespan);
        myEagle.flyUp();
        myEagle.flyDown();
    }
}
```

Resultado [12]

[Console output of TestEagle]

Reproduction: another egg

Outer covering: feather

Lifespan: 15

Flying up...

Flying down...

Polimorfismo

Polimorfismo [12]

```
class Bird {
    public void fly() {
        System.out.println("The bird is flying.");
    }
    public void fly(int height) {
        System.out.println("The bird is flying " + height + " feet high.");
    }
    public void fly(String name, int height) {
        System.out.println("The " + name + " is flying " + height + " feet
high.");
    }
```

Testando... [12]

```
class TestBird {
    public static void main(String[] args) {
        Bird myBird = new Bird();

        myBird.fly();
        myBird.fly(10000);
        myBird.fly("eagle", 10000);
    }
}
```

Resultado [12]

[Console output of TestBird]

The bird is flying.

The bird is flying 10000 feet high.

The eagle is flying 10000 feet high.

Polimorfismo [12]

```
class Animal {
    public void eat() {
        System.out.println("This animal eats insects.");
    }
}
class Bird extends Animal {
    public void eat() {
        System.out.println("This bird eats seeds.");
    }
}
```

Testando... [12]

```
class TestBird {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        myAnimal.eat();

        Bird myBird = new Bird();
        myBird.eat();
    }
}
```

Resultado [12]

[Console output of TestBird]

This animal eats insects.

This bird eats seeds.

Referências

[1] https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf

[2] https://cs.lmu.edu/~ray/notes/paradigms/

[3] https://www.geeksforgeeks.org/introduction-of-programming-paradigms/

[4] http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html

[5] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.7366&rep=rep1&type=pdf

[6] https://www.cs.bham.ac.uk/research/projects/poplog/paradigms_lectures/lecture1.html

[7] http://www.cs.sjsu.edu/faculty/pearce/modules/lectures/languages/Paradigms.htm

[8] Potok, Thomas; Mladen Vouk; Andy Rindos (1999). "Productivity Analysis of Object-Oriented Software Developed in a Commercial Environment" (PDF). Software – Practice and Experience. 29 (10): 833–847. doi:10.1002/(SICI)1097-024X(199908)29:10<833::AID-SPE258>3.0.CO;2-P. Retrieved 21 April 2010.

Referências

[9] Potok, Thomas; Mladen Vouk; Andy Rindos (1999). "Productivity Analysis of Object-Oriented Software Developed in a Commercial Environment" (PDF). Software – Practice and Experience. 29 (10): 833–847. doi:10.1002/(SICI)1097-024X(199908)29:10<833::AID-SPE258>3.0.CO;2-P.

[10] https://teamtreehouse.com/community/when-to-use-oop-over-procedural-coding

[11] https://medium.com/from-the-scratch/oop-everything-you-need-to-know-about-object-oriented-programming-aee3c18e281b

[12] https://raygun.com/blog/oop-concepts-java/

https://www.tiobe.com/tiobe-index/

https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019

Links Adicionais

https://teamtreehouse.com/library/objectoriented-php-basics-2/why-objectoriented-programming/why-use-objectoriented-programming#teacher-s-notes

https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming