

Linguagem de Programação II

Prof. Dr. Gustavo Leitão

Ementa

- Introdução a Programação Orientada a Objetos
- Classes e Objetos. Atributos e Métodos.
- Encapsulamento.
- Alocação dinâmica e coletor de lixo.
- Composição.
- Herança.
- Polimorfismo
- Classes abstratas e interfaces.
- Modularização.
- Tratamento de Exceções.
- Classes Genéricas.
- Anotações.
- Depuração e Profiling
- Aplicações em estruturas e algoritmos presentes em EDB2.

Conteúdo

- Apresentação da disciplina e Introdução a OO
- Criação de classes, objetos e pacotes
- Introdução a IDE e Gerenciamento de bibliotecas
- Coleções, Javadoc e Libs
- Conceito de Herança
- Polimorfismo
- Classes Abstratas
- Conceito de Interfaces
- Classes Genéricas
- Design de classes com SOLID
- Anotações
- Tratamento de exceções
- Explorando OO em outras linguagens
- Padrões de projetos

Comunicação



<https://discord.gg/wqnvd3nm>



<http://meet.google.com/kmi-bmcr-mcs>



gustavo.leitao@imd.ufrn.br







Atividades práticas

[Classrooms](#) / IMD0040-LP2

IMD0040-LP2

imetrodigital

 **Assignments** 0  **Students** 0  **TAs and Admins** 1  **Settings**

Assignments



Create an assignment to get started.

Create an individual assignment to generate an assignment repository for each student to work from. Or, create a group assignment and have students work collaboratively in groups from team repositories.

Create an assignment

Learn more about [individual](#) and [group](#) assignments.



Need to teach Git & GitHub fundamentals?

The Classroom team has created an assignment for you to use to teach your students the fundamentals of Git & GitHub.

Use starter assignment

[Learn more](#)

Presença

- Entrega das atividades no prazo

Avaliações

- I Unidade - Trabalhos individuais ao longo do curso
- II Unidade - Trabalho individual + Avaliação online
- III Unidade - Trabalho final em grupo

Bibliografia



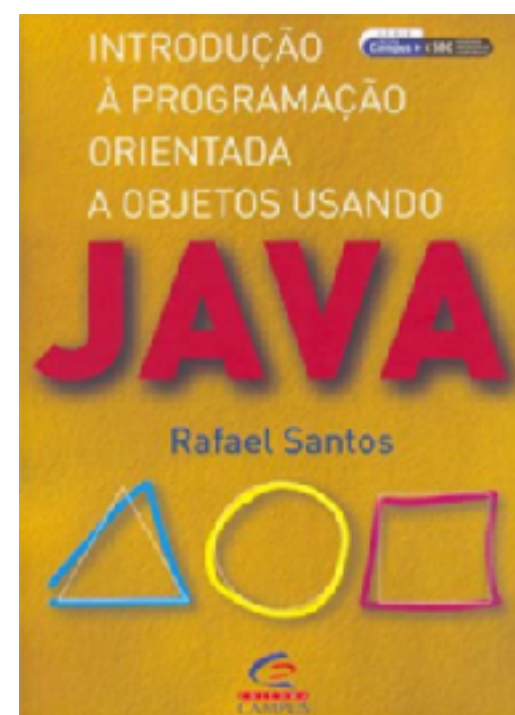
Orientação a Objetos: Aprenda seus conceitos e suas aplicabilidades de forma efetiva

Thiago Carvalho
Casa do Código



Programação orientada a objetos: Conceitos e técnicas

Sérgio Furgeri
Editora Érica



Introdução A Programação Orientada A Objetos Usando Java

Rafael Santos
Editora: Elsevier

Java e Orientação a Objetos

Acessível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/>

Introdução a POO

O que é Programação Orientada a Objetos?

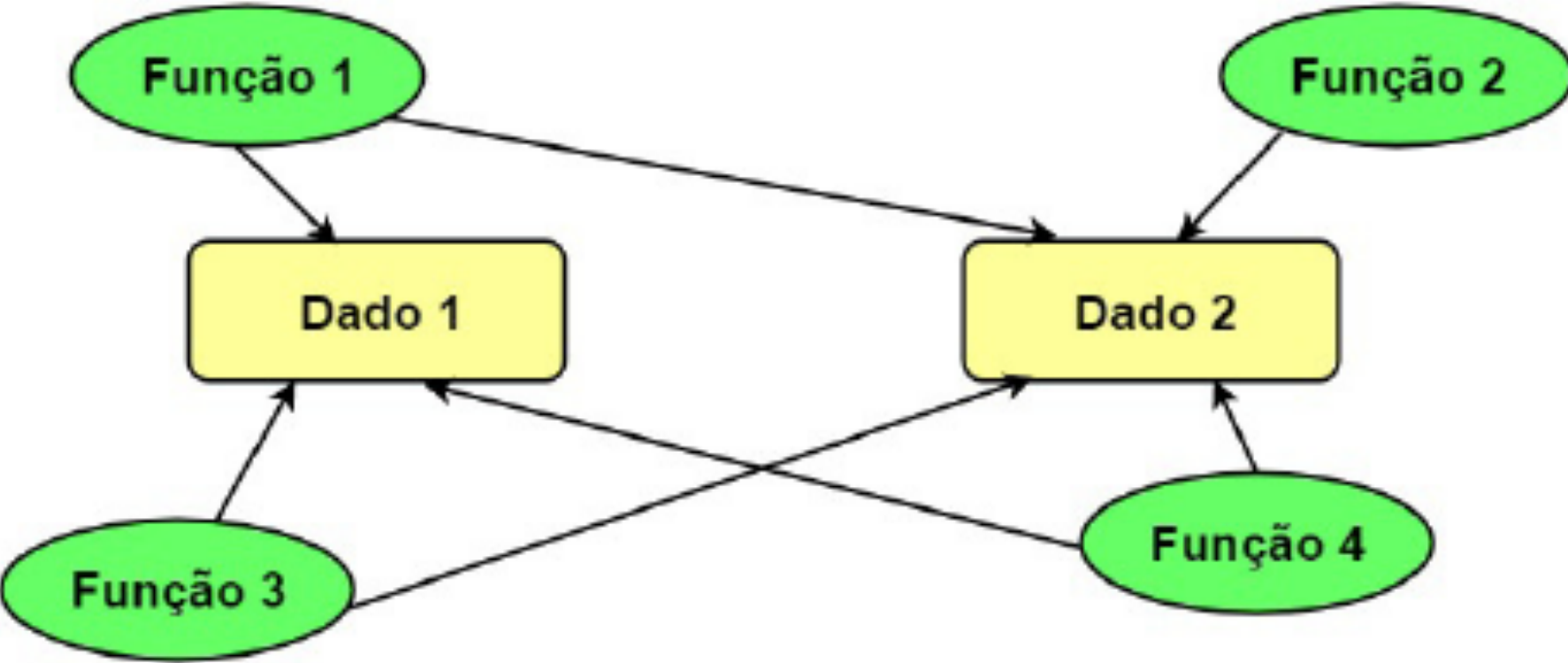
- É um paradigma de programação onde se usam classes e objetos criados a partir dos modelos descritos anteriormente, para representar e processar dados.
- Os objetos são criados a partir de Classes que são estruturas que buscam criar modelos simplificados do mundo real.
- Os objetos possuem atributos que armazenam estados (variáveis do objeto) e operações (métodos) que executam ações sobre esses dados.



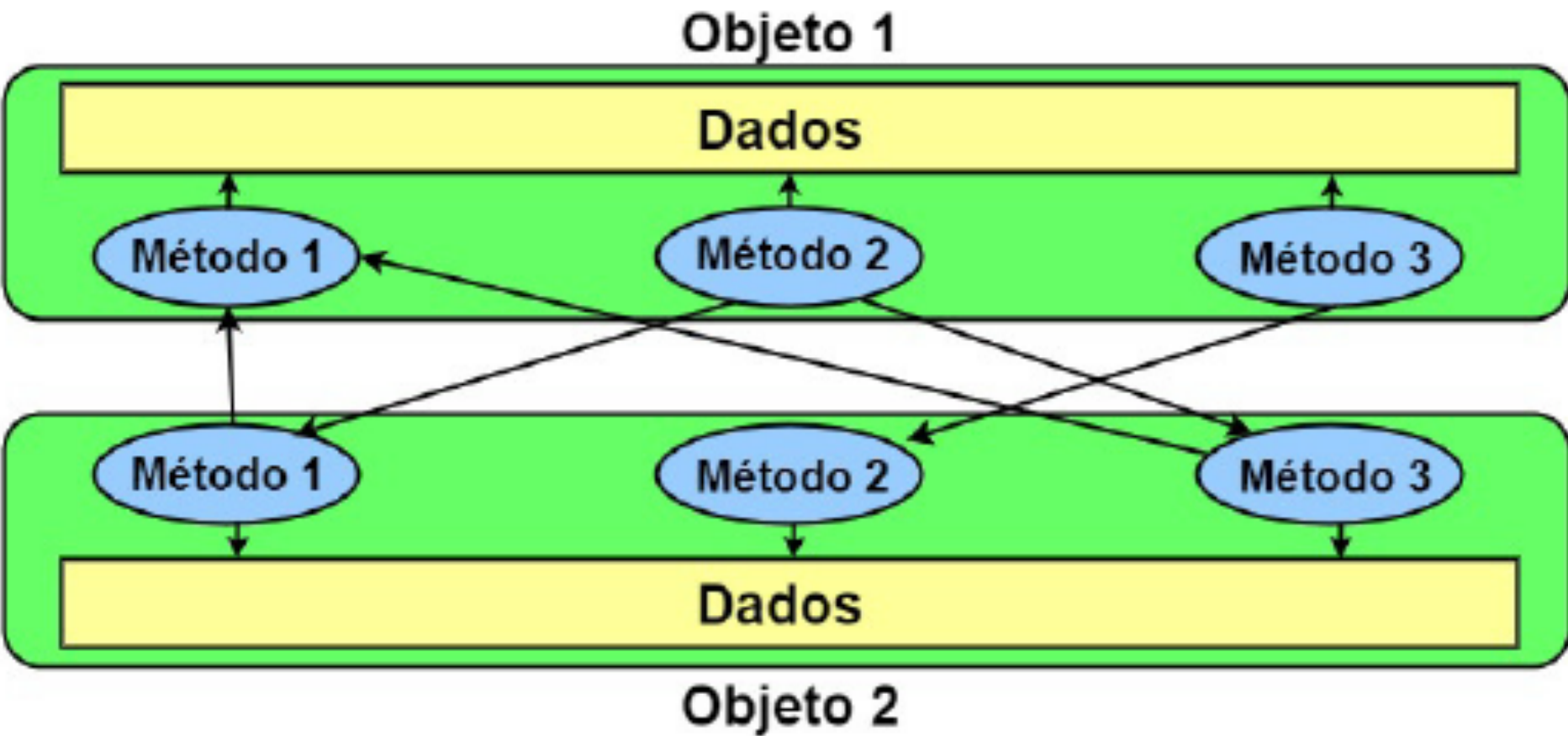
O que é Programação Orientada a Objetos?

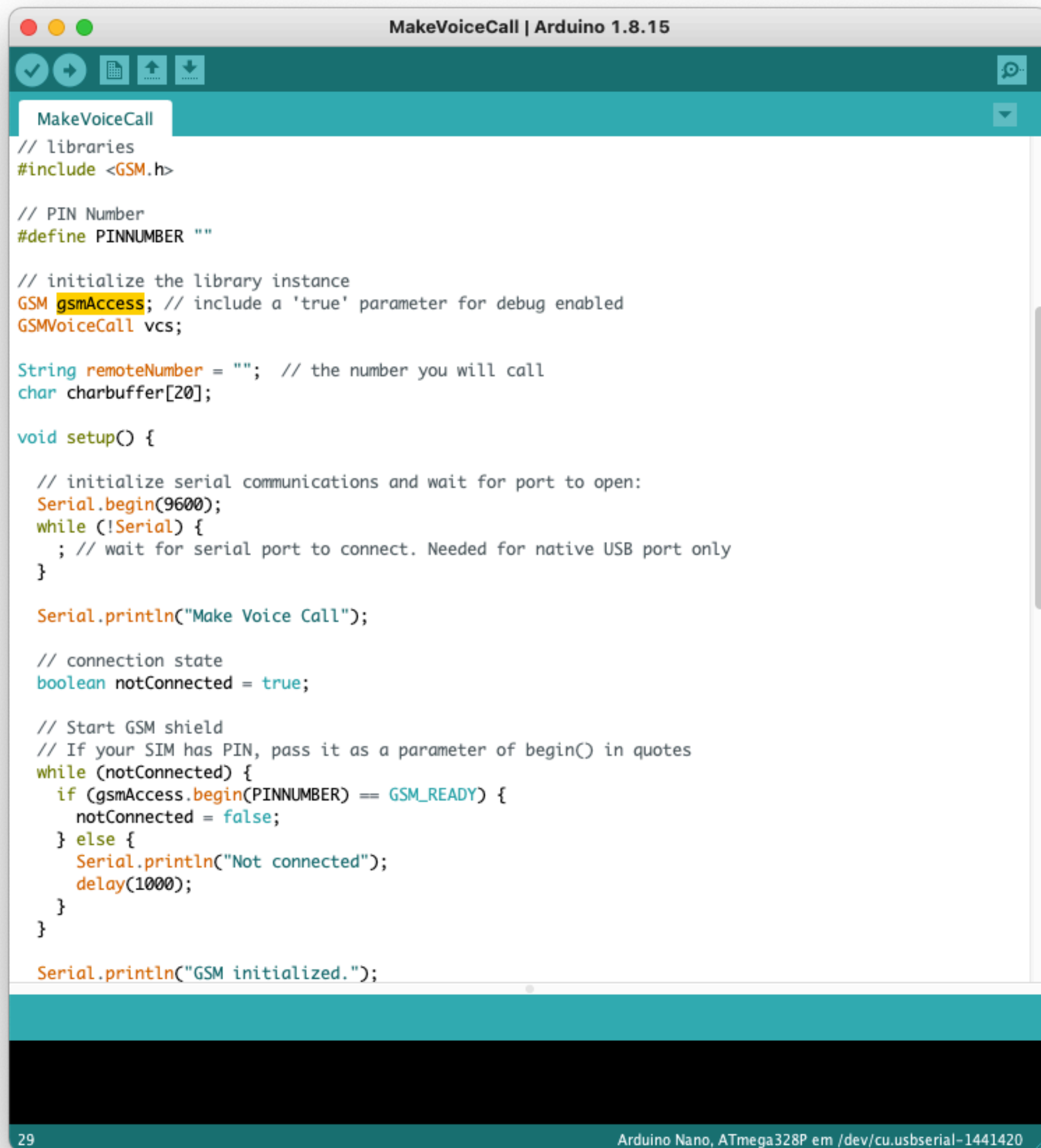


Programação estruturada



POO





```
MakeVoiceCall | Arduino 1.8.15

// libraries
#include <GSM.h>

// PIN Number
#define PINNUMBER ""

// initialize the library instance
GSM gsmAccess; // include a 'true' parameter for debug enabled
GSMVoiceCall vcs;

String remoteNumber = ""; // the number you will call
char charbuffer[20];

void setup() {

    // initialize serial communications and wait for port to open:
    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }

    Serial.println("Make Voice Call");

    // connection state
    boolean notConnected = true;

    // Start GSM shield
    // If your SIM has PIN, pass it as a parameter of begin() in quotes
    while (notConnected) {
        if (gsmAccess.begin(PINNUMBER) == GSM_READY) {
            notConnected = false;
        } else {
            Serial.println("Not connected");
            delay(1000);
        }
    }

    Serial.println("GSM initialized.");
}
```

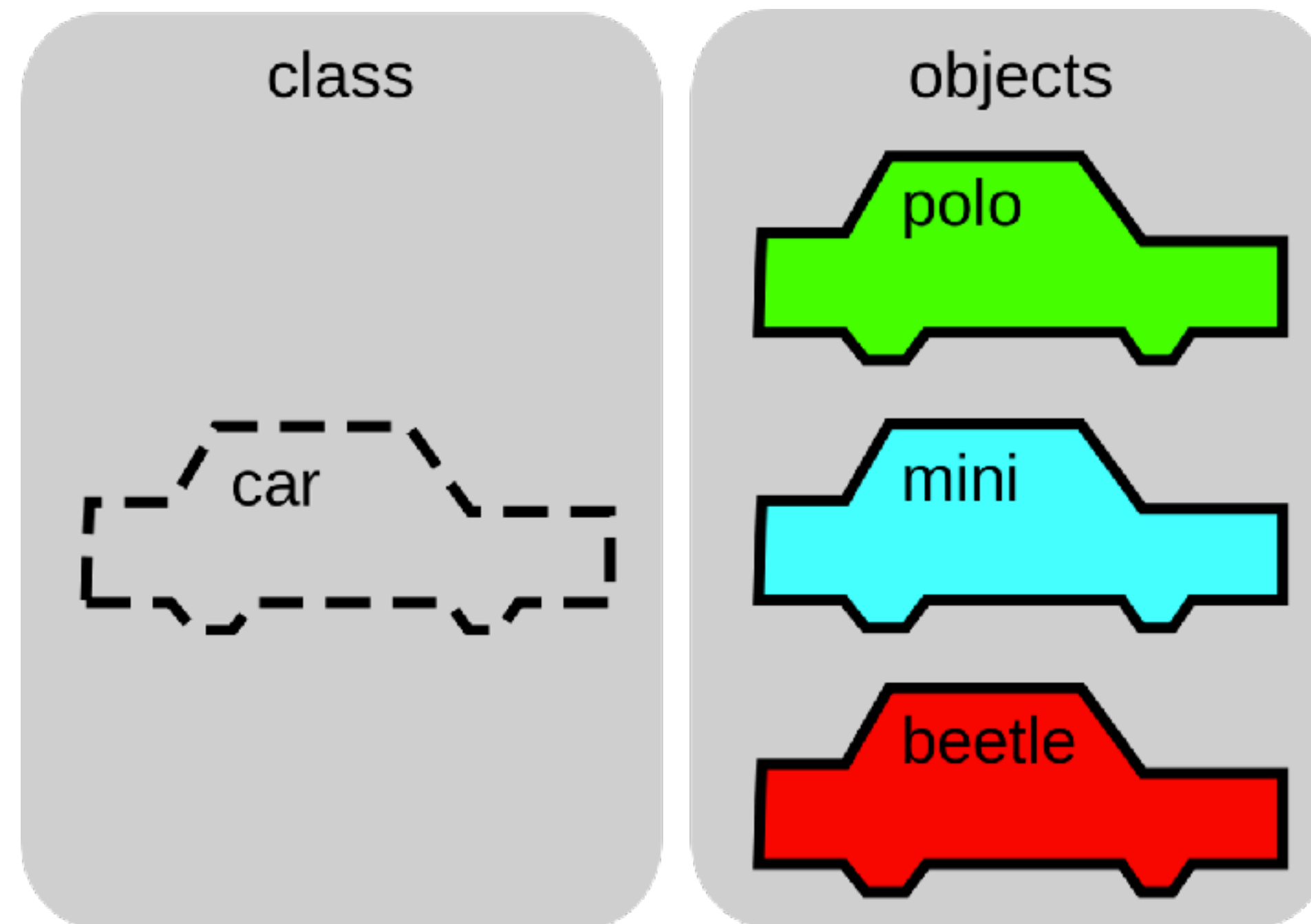
29 Arduino Nano, ATmega328P em /dev/cu.usbserial-1441420

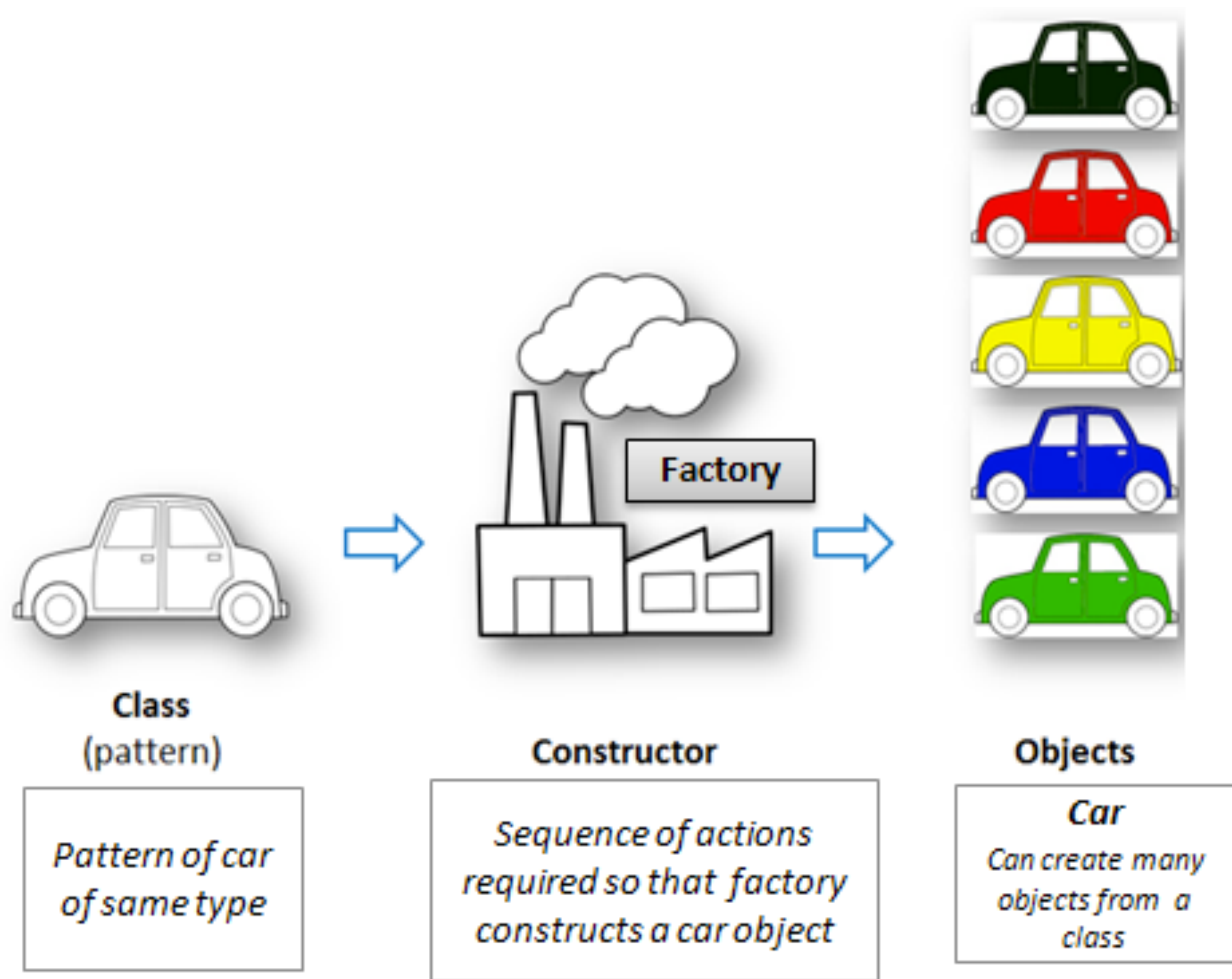
Exemplo de código estruturado

O problema do paradigma estruturado é que não existe uma forma simples de criar **conexão forte entre dados e funcionalidades**. No paradigma orientado a objetos, é muito fácil ter essa conexão por meio dos recursos da própria linguagem.

O que são as classes?

- A criação e definição dos objetos ocorre através de classes. As classes são os modelos que definem o comportamento dos objetos.





Breve histórico

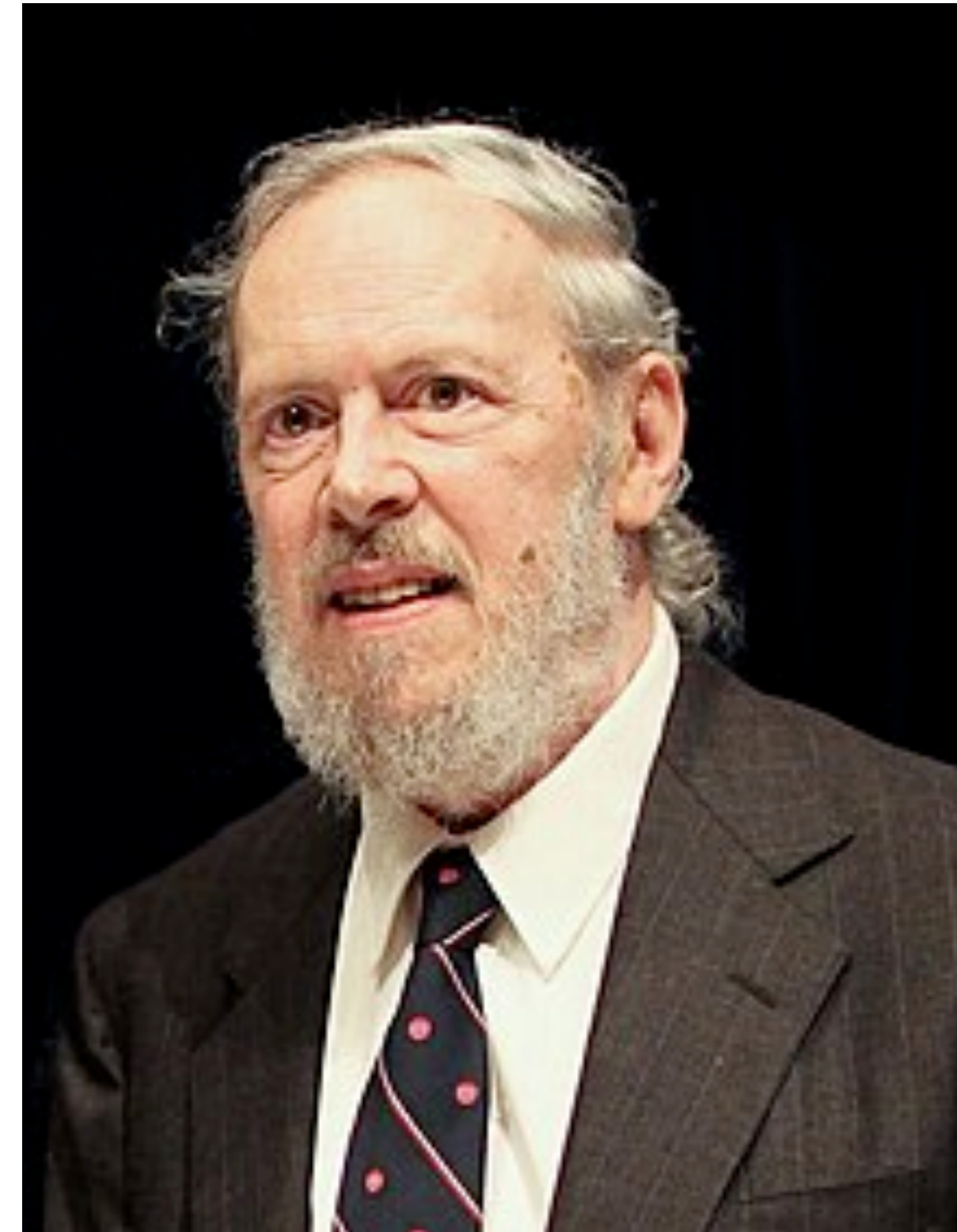
- 1962 - Dois pesquisadores Noruegueses criaram uma linguagem para simulação de eventos discretos: SIMULA I que depois ficou conhecida como SIMULA 67.

"As contribuições técnicas de SIMULA são impressionantes, quase incríveis. Dahl e Nygaard, em sua tentativa de criar uma linguagem onde os objetos do mundo real seriam de forma precisa e naturalmente descritos, apresentou avanços conceituais que se tornariam realidade somente quase duas décadas mais tarde: tipo abstrato de dados, o conceito de classe, herança, o conceito de co-rotina (método), [...] a criação, exclusão e operações de manipulação em objetos são apenas um exemplo".



Breve histórico

- 1970 - SIMULA Rodava em UNIVAC 1107. O pesquisador Alan Kay (Xerox) criou nova linguagem que pudesse ser usada nos emergentes PCS: Smalltalk. A Smalltalk é considerada a linguagem que tornou OO conhecida até os dias de hoje.
- 1983 - A linguagem C criada por Dennis Ritchie em 1972, evoluiu para C++ com contribuição de Bjarne Stroustrup e incorporou características de orientação a objetos.



Dennis Ritchie



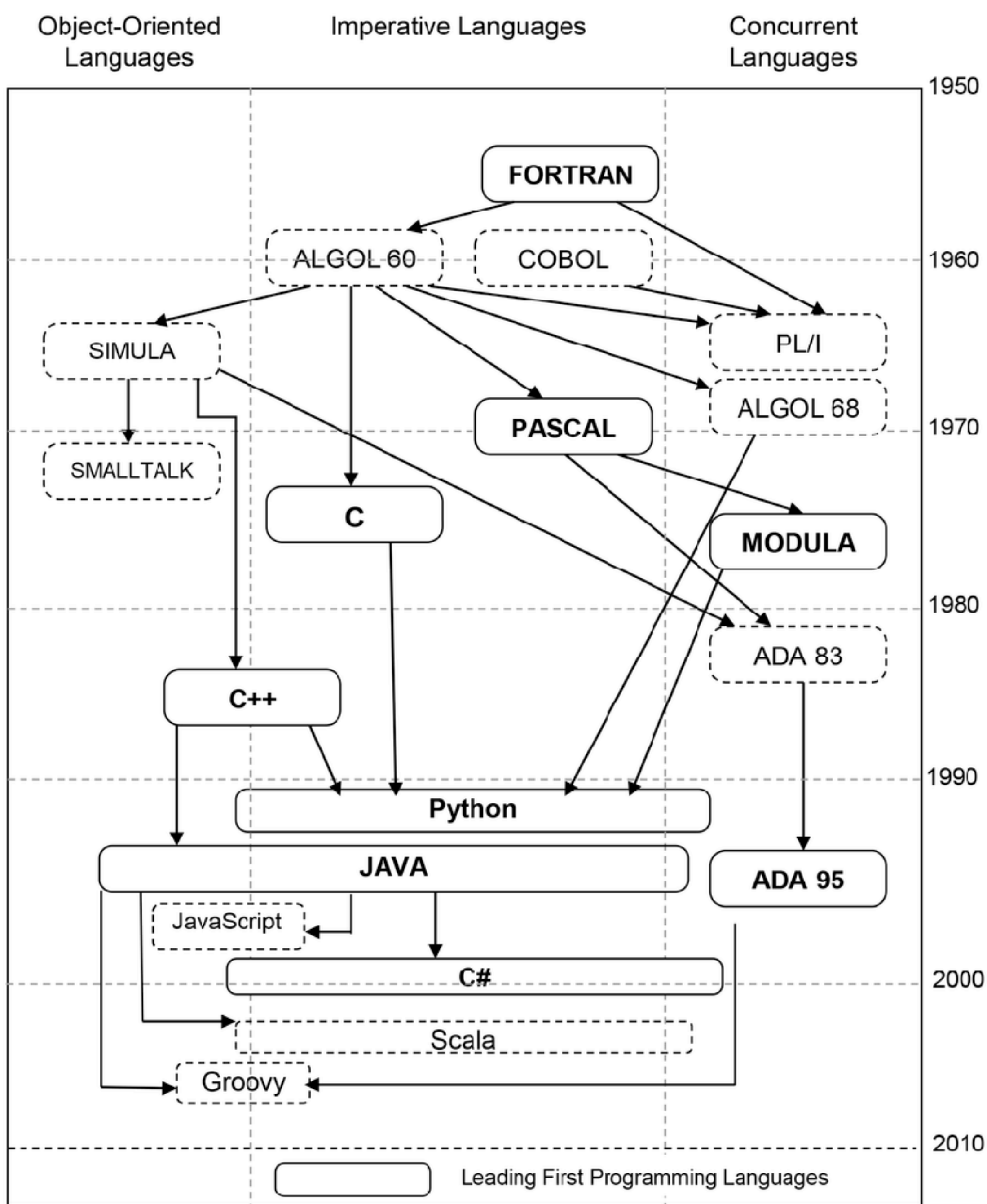
Bjarne Stroustrup

Breve histórico

- 1995 - Linguagem Java é oficialmente liberada tendo OO como paradigma. Java buscava trazer portabilidade.



James Gosling




```

C 1 2 3 4 5 6
C234567890123456789012345678901234567890123456789012345
PROGRAM BASKHARA
C
REAL :: A,B,C, DELTA, X1,X2, RE, IM
C
PRINT *, "Este programa resolve uma equação de 2o.grau"
PRINT *, "no formato: a*x**2 + b*x + c = 0"
C
PRINT 10, "Digite a, b, c: "
10 FORMAT( A, 1X, $)
20 READ(*, *, ERR=20)A, B, C
C
DELTA= B**2-4.0*A*C
C
IF( DELTA.GT.0 )THEN      ! (DUAS RAÍZES REAIS)
  X1= ( -B-SQRT(DELTA) ) / ( 2.0*A )
  X2= ( -B+SQRT(DELTA) ) / ( 2.0*A )
  PRINT *, "RAIZES: X1= ", X1
  PRINT *, "X2= ", X2
ELSE
  IF( DELTA.EQ.0 ) THEN ! (DUAS RAÍZES REAIS IGUAIS)
    X1= -B / ( 2.0*A )
    X2= X1
    PRINT *, "RAIZES: X1=X2= ", X1
  ELSE      ! (DUAS RAÍZES COMPLEXAS)
    RE= -B / ( 2.0*A )
    IM= SQRT( -DELTA ) / ( 2.0*A )
    PRINT *, "RAIZES COMPLEXAS: X1= ", RE,"- ", IM, "i"
    PRINT *, "X2= ", RE, "+ ",IM, "i"
  END IF
END IF
C
END PROGRAM BASKHARA

```

Fortran

Por que usar POO?

- **Encapsulamento** - Esconde do usuário (neste caso usuário da classe) os detalhes da implementação.
- **Reuso** - POO Facilita o reuso de mesma porção de código
- **Coesão** - Cada unidade de código executa raras que dizem respeito somente ao conceito que ela pretende representar
- **Acoplamento** - Permite gerar acoplamento com outras com unidades abstratas de código que podem ser modificadas independentemente.