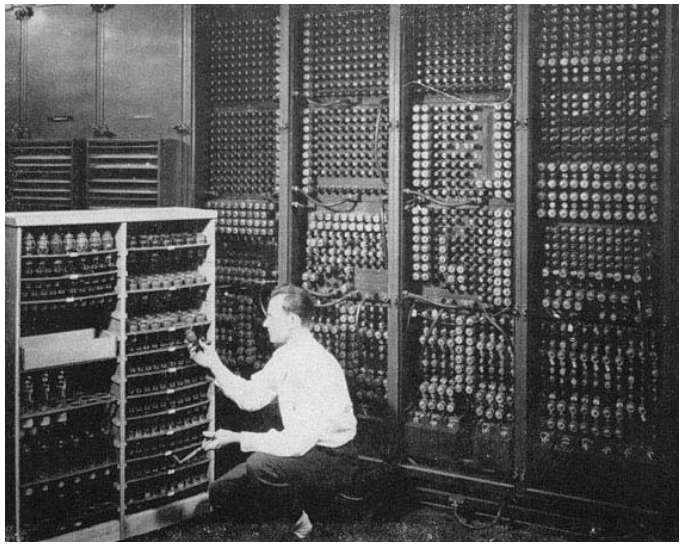
Desenvolvimento de Software

Cássio Seffrin

Gerações de Computadores

- 1a. Geração – **Válvulas**
- 2a. Geração – **Transistores**
- 3a. Geração – **Circuitos Integrados**
- 4a. Geração – **VLSI**
- 5a. Geração – **Conectividade**, **mobilidade**

1a. Geração – 1942 à 1955



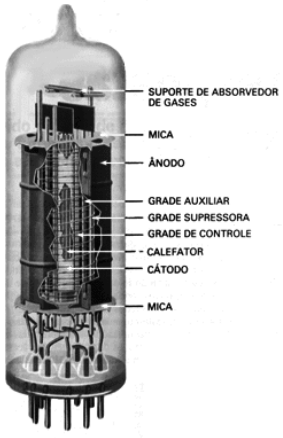
Estímulo: 2a. Guerra Mundial

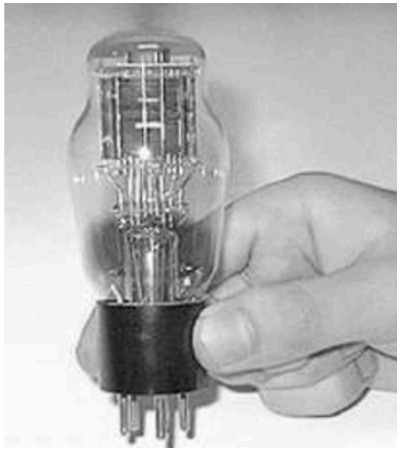
Usavam válvulas eletrônicas

Esquentavam MUITO

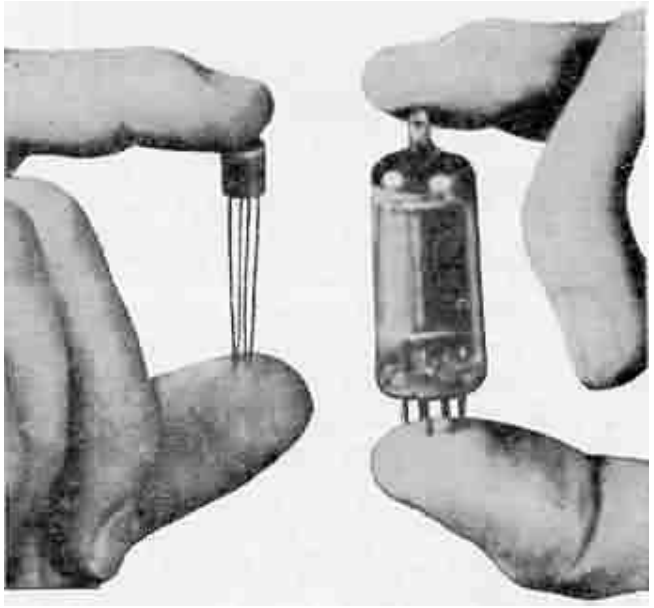
Vários quilometros de fios
Lentos

Ocupavam MUITO espaço





2a. Geração – 1955 à 1965



As válvulas foram substituídas

por transistores

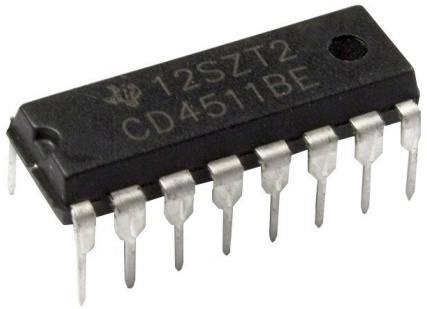
Os fios foram substituídos por ligação

por circuito impresso

Estas substituições permitiram

- Redução de custo
- Redução de tamanho
- Aumento da velocidade de processamento

3a. Geração – 1965 à 1980



Construída a partir de circuitos integrados

Os circuitos integrados permitiram

- Redução de custo
- Redução de tamanho
- Aumento da velocidade de processamento que alcançou a ordem de microsegundos (10^{-6})

Têm início o uso de Sistemas Operacionais + avançados

4a. Geração – 1980 à atual

Aperfeiçoamento da tecnologia atual

VLSI (Very Large Scale Integration)

VLSI permitiram

- Redução de custo
- Redução de tamanho
- Aumento da velocidade de processamento

5a. Geração

1991 à …

CONECTIVIDADE (802.11, GSM, GPRS, 3G, 4g LTE, 5g..)

MOBILIDADE (notebooks, smartphones, relogios)

notebooks: autonomia entre 30 minutos à ~8 horas

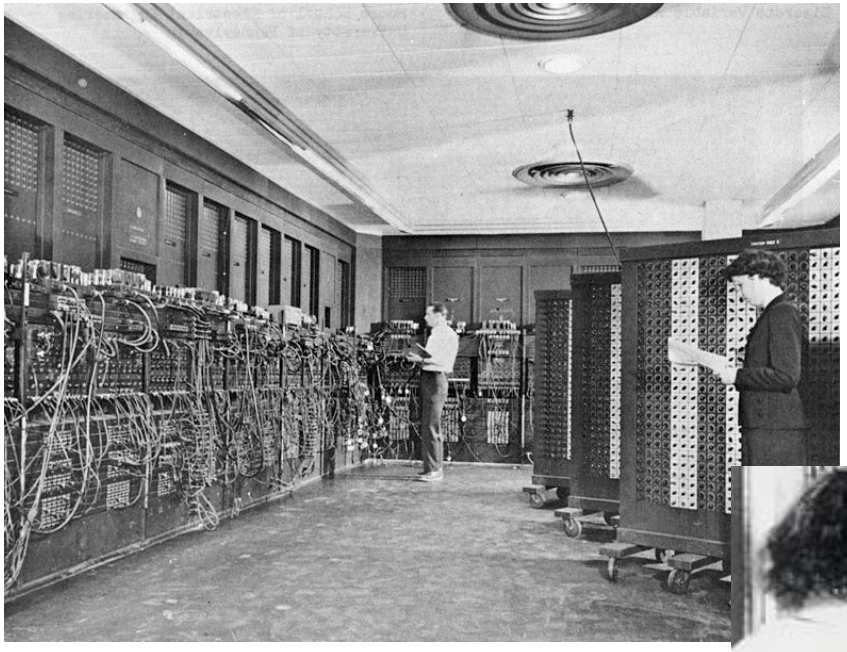
Data Science

Computer Vision

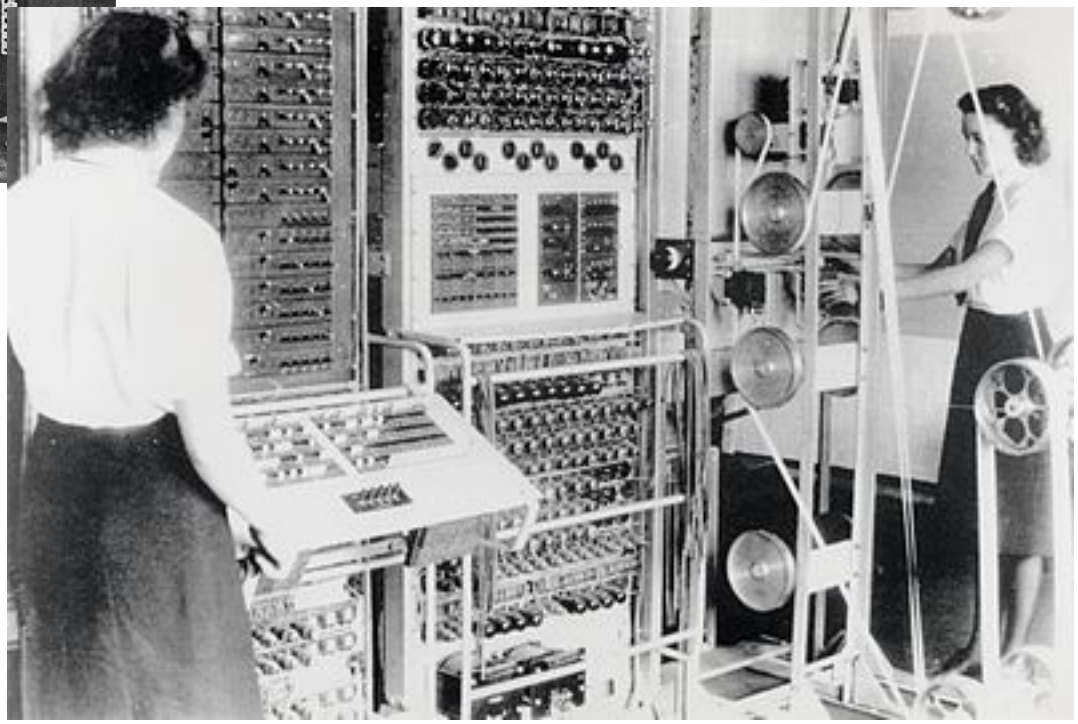
Desenvolvimento de Software

- Metodologia Procedural (PE)
- Metodologia Orientada o Objetos (POO)
- Objeto (estrutura de dados que representam estado / comportamento

- Fortran IBM (1954)
- Basic 65 Microsoft
- Simula 67 POO
- Pascal Foi criada em 1970 pelo suíço Niklaus Wirth
- C(73), C++ (80), C#, Objective C
- Smalltalk 72-80 POO
- Phyton 1991 POO
- Java (1992)
- PHP Rasmus Lerdorf (1995)
- Ruby (1995)
- Lua (nmap Scripting Engine) - The Lua language is designed, implemented, and maintained at PUC-Rio in Brazil since 1993 (Waldemar, Roberto, Luiz)



ENIAC 1945 - 1947 - US



Computador Colossus sendo operado UK
Mark I- Dezembro de 1943 e
Mark II – 1 de Junho de 1944

Simula 67, cuja primeira versão foi apresentada em 1966 foi a 1ª linguagem orientada a objetos e introduziu os conceitos de classes e herança.

Exemplo de Classe em Simula:

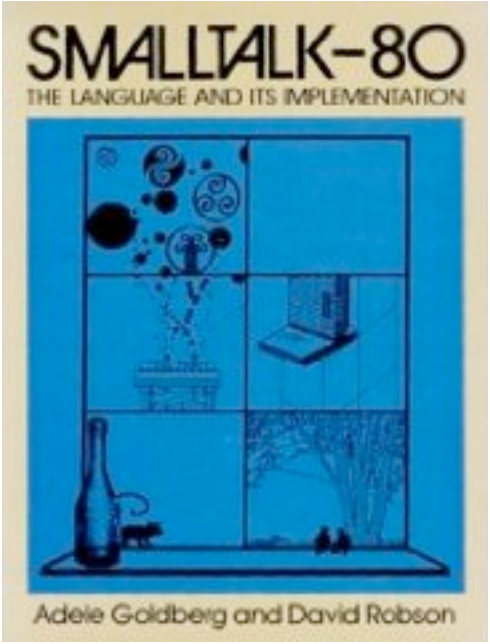
```
Glyph Class Line (elements);
    Ref (Glyph) Array elements;
  Begin
    Procedure print;
    Begin
      Integer i;
      For i:= 1 Step 1 Until UpperBound (elements, 1) Do
        elements (i).print;
      OutImage;
    End;
  End;
```

Ken Thompson e Dennis Ritchie (C 1972 - Unix 1983)

AT&T Bell Labs entre 1969 e 1973

Richard Matthew Stallman: FSF e GNU

Smalltalk-80, ou simplesmente Smalltalk, é uma linguagem de programação orientada a objeto fracamente tipada.



Em *Smalltalk* tudo é objeto: os números, as classes, os métodos, blocos de código, etc. Não há tipos primitivos, ao contrário de outras linguagens orientadas a objeto; strings, números e caracteres são implementados como classes em *Smalltalk*, por isso esta linguagem é considerada puramente orientada a objetos. Tecnicamente, todo elemento de *Smalltalk* é um objeto de primeira ordem.

James Gosling

Linguagem Java

- Oak (árvore de carvalho)
- Java
- POO
- Compilador Bytecode
- Tipagem forte
- Classes
- Atributos
- Construtores
- Métodos
- Objeto
- Modificadores de acesso (private, protected (subclasse), public)
- Outros Modificadores (final/static/abstract (implem. subclasses))

Tipos Primitivos

		Valores possíveis				
Tipos	***Primitivo***	***Menor***	***Maior***	***Valor Padrão***	***Tamanho***	***Exemplo***
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;

	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d;
						ou
						double ex6 = 10.20;

Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;

Linguagem Java

Exemplo de um método em java

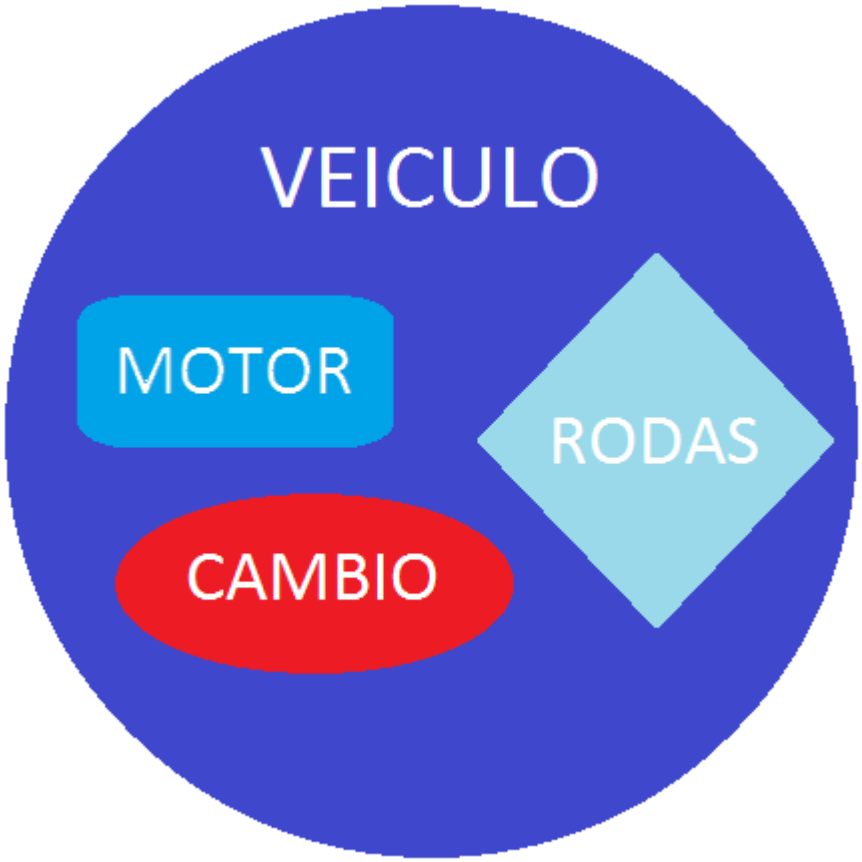
```
class Matematica{
	float raizQuadrada(float numero) {
		float resultado = (float) Math.sqrt(numero);
		return resultado;
	 }

	float multiplica(float numero1, float numero 2) {
		float resultado = numero1 * numero2;
		return resultado;
	 }
}
```

Linguagem Java

- Principais características
  - Polimorfismo (parâmetros)
  - Composição (todo/parte)
  - Interfaces
  - Herança
  - Herança múltipla

Linguagem Java

Exemplo de Composição



Linguagem Java

Exemplo de uma interface

```
interface BicicletaInterface {
      void mudarMarcha(int newValue);
      void aumentarVelocidade(int increment);
      void freiar ();
}


class Bicicleta implements BicicletaInterface {
            /* comentário Java:
               implementar os 3 métodos da interface
               BicicletaInterface
            */
}
```

Linguagem Java

Exemplo de uma herança

```
class Pessoa {
    private String nome;
    private Integer idade;
    private String sexo;
    private Date dataNascimento;
              public Pessoa(){}
    /* .... Getters and setters .... */
}
class Funcionario extends Pessoa{
    private int salario ;
    public Funcionario(){}
  /* .... Getters and setters .... */

  public static void main (String a[]){
        Funcionario f = new Funcionario();
        f.setNome("Cassio");
        f.setSalario(2000);
        System.out.println("Nome: "+ f.getNome() + " Salario: "
+f.getSalario());
    }
}
```

Linguagem Java

Classes Abstratas

- Servem como modelo para uma classe concreta
- Não podem ser instanciadas diretamente
- Podem conter ou não métodos abstratos
- Pode implementar ou não um método

Exemplo no netbeans

Modelagem de sistema de software

- Complexidade cresce a medida que o software aumenta
- Casa para o cachorro
- Casa para família
- Edifício
- Software

- Características tratadas com diversos modelos, analogamente ao um avião que possui um diagrama elétrico, outro para aerodinâmica.

- Gerenciamento da complexidade
- Comunicação entre as pessoas envolvidas
- Redução dos custos no desenvolvimento
- Previsão do comportamento futuro do sistema

UML

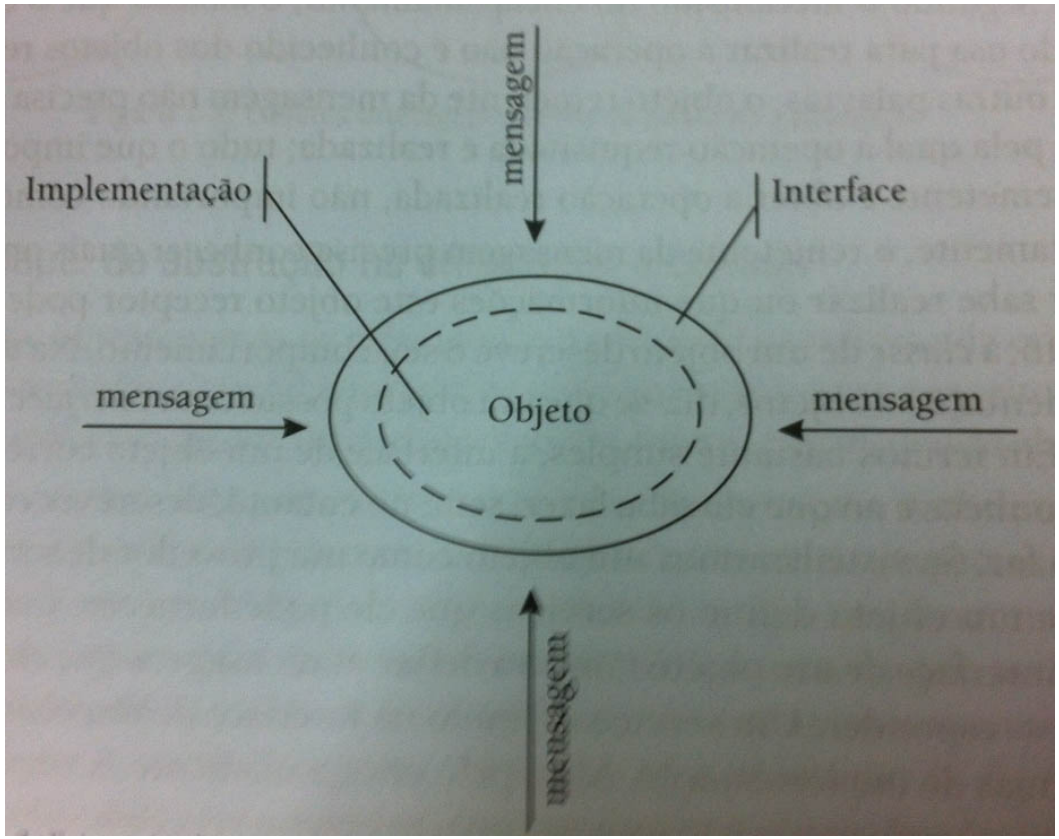
- Grandy Booch, James Rumbaugh e Ivar Jacobson. Os 3 amigos.
- Em 1997 a UML foi aprovada como padrão pelo OMG (Object Management Group. Consórcio Internacional que define e ratifica padrões na área de orientação a objetos.

Modelagem de sistema de software

O paradigma da orientação a objetos (uma forma de abordar um problema)

1. Qualquer coisa é um objeto
2. Objetos realizam tarefas por meio da requisição de serviços a outros objetos.
3. Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares.
4. A classe é um repositório para comportamento associados ao objeto.
5. Classes são organizadas em hierarquias.

Modelagem de sistema de software



Fonte: Lorenzo Ridolfi, Sérgio Colcher (2007)

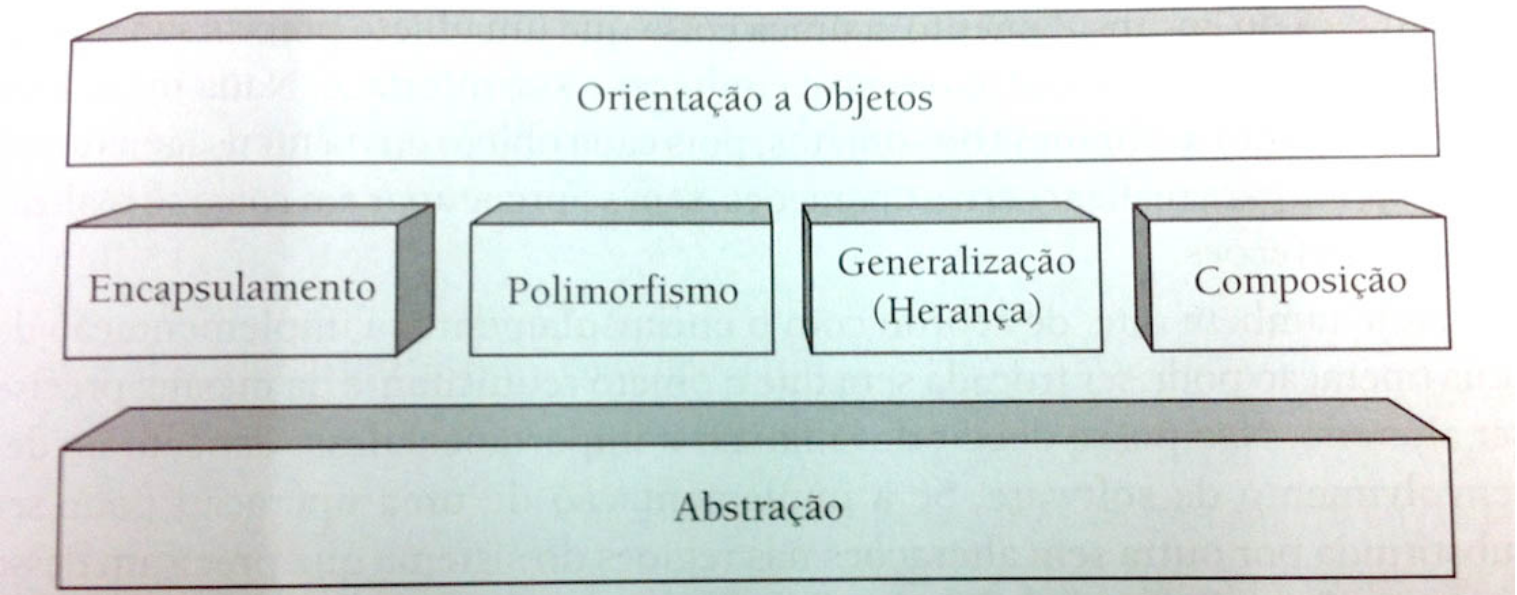


Figura 1-2: Princípios da orientação a objetos podem ser vistos como aplicações de um princípio mais básico, o da abstração.

Fonte: Lorenzo Ridolfi, Sérgio Colcher (2007)

Encapsulamento

- Os objetos possuem comportamento (métodos)
- O encapsulamento é uma forma de restringir o acesso ao comportamento interno dos objetos.
- A relação entre objetos é feita através da troca de mensagens

Herança (Generalização)

- Na generalização, classes semelhantes são agrupadas em uma hierarquia.
- Exemplo: um funcionário estende de uma classe de pessoa.

Composição

- Objetos que compõe outros objetos
- Um livro é composto de paginas, capa, títulos, parágrafos.

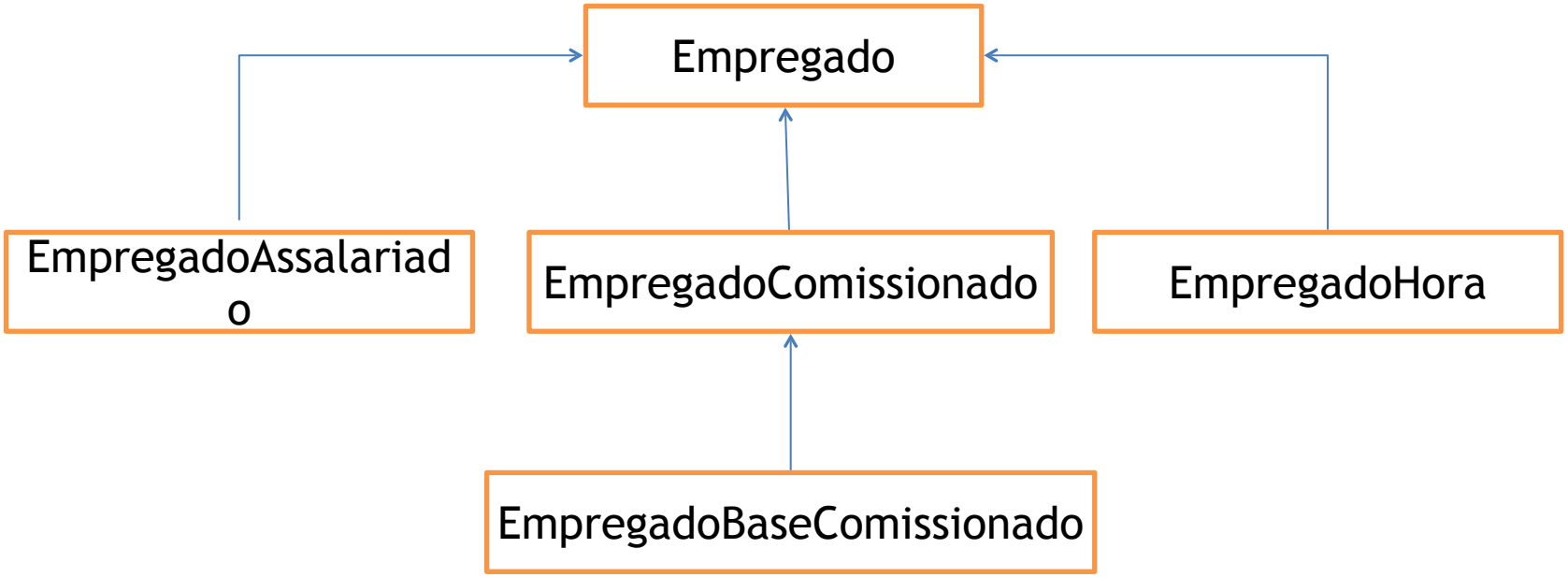
Polimorfismo

•O polimorfismo indica a capacidade de abstrair varias implementações diferentes em uma única interface. (Lorenzo Ridolfi, Sérgio Colcher)
•O polimorfismo permite programar no geral ao invés de programar no específico (deitel & deitel).
Exemplos de polimorfismo.

Sobrescrita e Sobrecarga de Métodos;
Implementações de métodos definidos como abstratos na superclasse;

Abstração

•O processo de abstração é usado para esconder certos detalhes e somente mostrar as características essenciais de um objeto, ou seja, apresentar uma visão externa de um objeto (interface)


Empregado
EmpregadoAssalariado
EmpregadoComissionado
EmpregadoHora
EmpregadoBaseComissionado

Generics

Apartir Java 5
O que é um tipo Genérico
Para que serve?
Exemplo de Array[] simples

Generics

```
Integer[] intArr = {1, 2, 3, 4, 5};
Double[] douArr = {1.2, 2.1, 3.4, 4.5, 5.6};
String[] strArr = {"Cassio", "Fred", "Juliana"};
```

Como imprimir sem Generics?
Como imprimir com Generics?

Generics

- Tipos genéricos em Arrays funcionam somente substituindo tipos por referencias como Integer, Float, String. Tipos primitivos (int, char, float) não funcionam
- Os nomes dos parametros de tipo por toda declaracao do metodo devem corresponder aqueles declarados na assinatura do metodo
- O tipo Object pode ser usado invés do conceito genérics quando o retorno do método é void.

Generics

Convenção para determinar nomes para Genéricos

- E – Element (usado para todos elementos do Java Collections Framework, exemplo ArrayList, Set etc.)
- K – Key (usado na coleção Map)
- N – Number
- T – Type
- V – Value (usado na coleção Map)
- S,U,V etc. – T, U, V etc.. quando não sabemos o tipo

GIT servidor/cliente

servidor git
#mkdir repo
#cd repo
#git init —bare

cliente com projeto existente
entrar no diretório
git init

```
git remote add origin /Users/cassioseffrin/Desktop/repo2
git add .
git commit –m”primeiro commit”
git push origin master
```

cliente2 com git clone em outro host
#mkdir cliente1
#cd cliente1
#git clone cassio@192.168.0.102:/Users/cassioseffrin/Desktop/repo
#echo "teste" > teste.txt
#git add teste.txt
#git commit -m”primeiro commit”
#git push origin master

GIT servidor/cliente

Passo 1: dentro do diretório workspace fazer o clone

Passo 2: git clone https://github.com/cassioseffrin DevSoftware2021.git

Pull Request (PR) no GitHub

Clicar o botão Fork no canto superior direito. Isso cria uma nova cópia do meu repositório de demonstração em sua conta de usuário do GitHub com um URL como:

https://github.com/<seunome>/farmaciaMvnFx
A cópia inclui todo o código, branches e commits do repositório original.

Em seguida, clone o repo abrindo o terminal em seu computador e executando o comando:

git clone https://github.com/<seunome>/farmaciaMvnFx
Depois que o repo é clonado, você precisa fazer duas coisas:

Crie uma nova branch emitindo o comando:

git checkout -b nova_branch
Crie um novo controle remoto para o repositório upstream com o comando:

git remote add upstream https://github.com/cassioseffrin/farmaciaMvnFx
Nesse caso, "repo upstream" se refere ao repositório original a partir do qual você criou seu fork.

Agora você pode fazer alterações no código. O código a seguir cria uma nova branch, faz uma alteração arbitrária e o envia para nova_branch:

Pull Request (PR) no GitHub

```
$ git checkout -b nova_branch
Switched to a new branch ‘nova_branch’
$ echo “texto qualquer” > teste.txt
$ git status
On branch nova_branch
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    teste.txt
nothing added to commit but untracked files present (use "git add" to track)
$ git add teste.txt
$ git commit -m “commit na nova_branch"
[nova_branch (root-commit) 4265ec8] Adding a test file to new_branch
 1 file changed, 1 insertion(+)
 create mode 100644 teste.txt
$ git push -u origin nova_branch
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 918 bytes | 918.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
Remote: Create a pull request for ‘nova_branch’ on GitHub by visiting:
Remote:   https://github.com/cassioseffrin/farmaciaMvnFx/pull/new/new_branch
Remote:
 * [new branch]         nova_branch -> nova_branch
```

Depois de fazer o push, voltar ao GitHub e verificar o botão verde Pull Request que irá aparecer.

Bibliografia

Lorenzo Ridolfi, Sérgio Colcher. Princípios de Análise e Projeto de Sistemas com UML,

FREEMAN & FREEMAN, Eric & E. Padrões de Projetos: Seu cérebro em padrões de projetos. Rio de Janeiro: ALTABOOKS, 2007.

METSKER, S. J. Padrões de projeto em Java.Porto Alegre: Bookman, 2004.

ANSELMO, F. Aplicando Lógica OO [Orientada a Objetos] em JAVA. 2. ed. Florianópolis: Visual Books Ltda., 2005. 178 p. ISBN 85-7502-162-1.