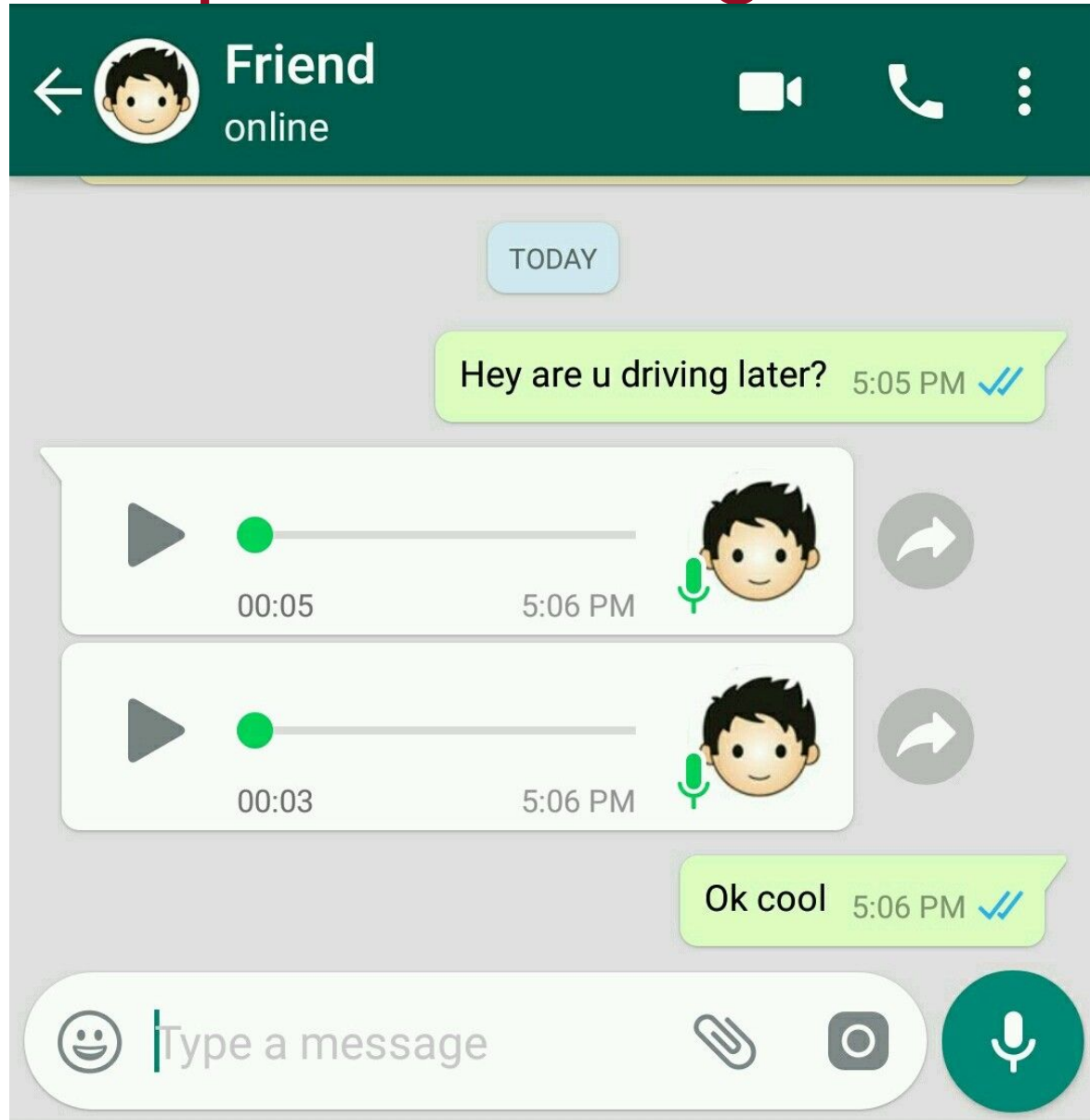


INF 112 - Programação II

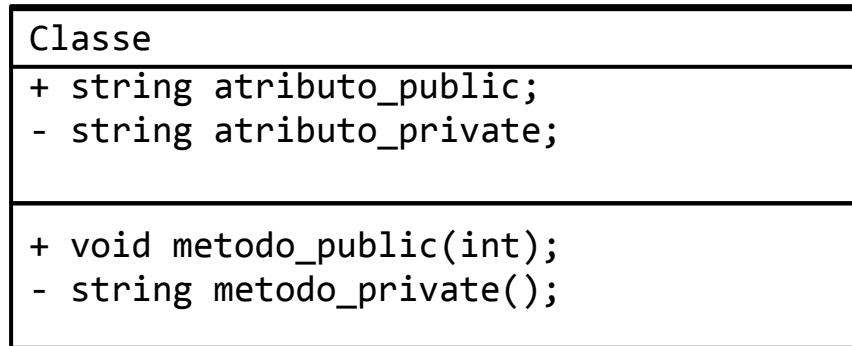
Interface e Polimorfismo

Interfaces

Diferentes tipos de Mensagens

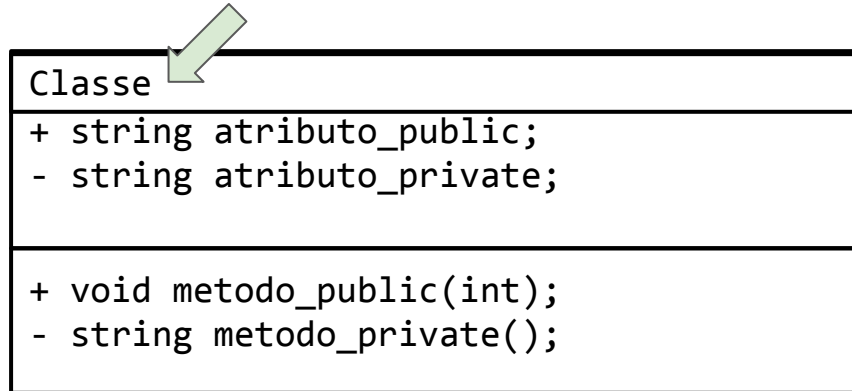


Unified Modelling Language



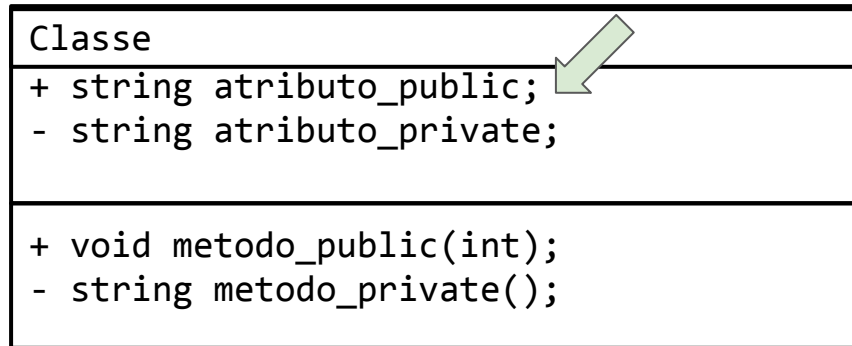
- UML define uma padrão de diagramas
- Úteis para o resto da disciplina

Unified Modelling Language



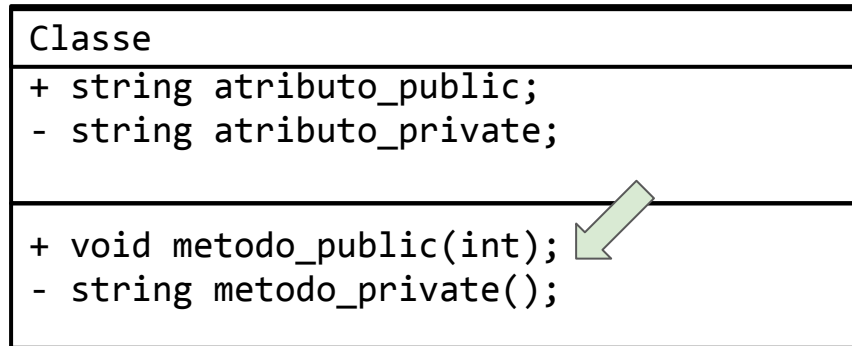
■ Nome da Classe

Unified Modelling Language



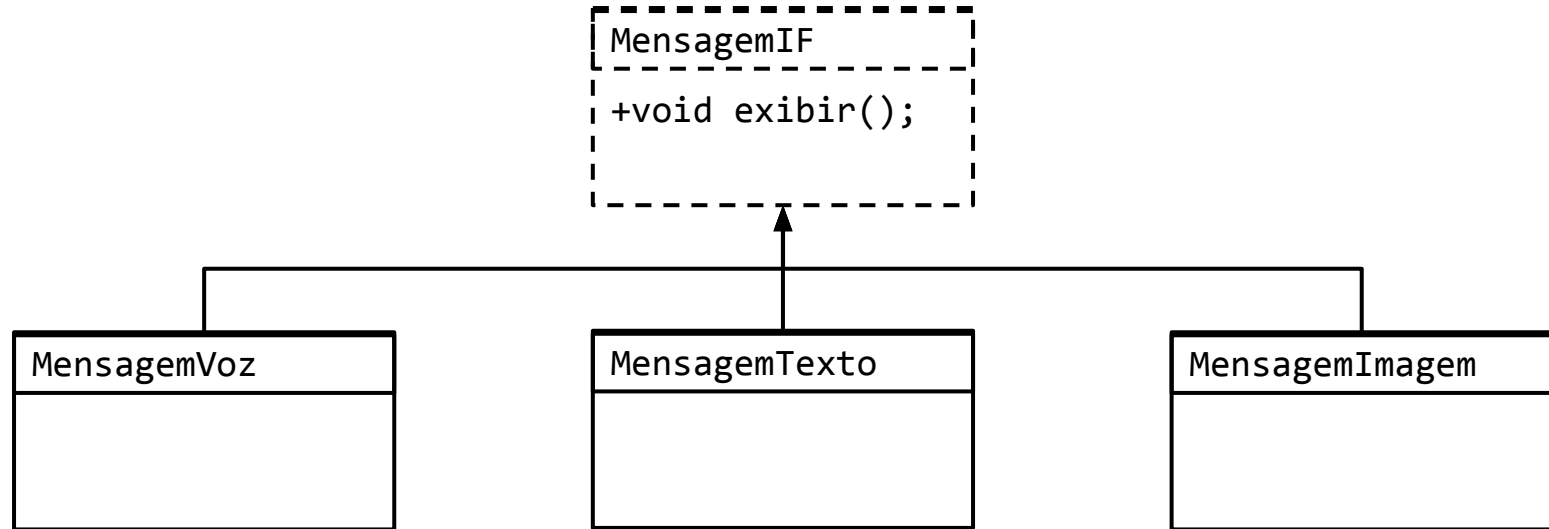
- Atributos
 - + → public
 - - → private

Unified Modelling Language



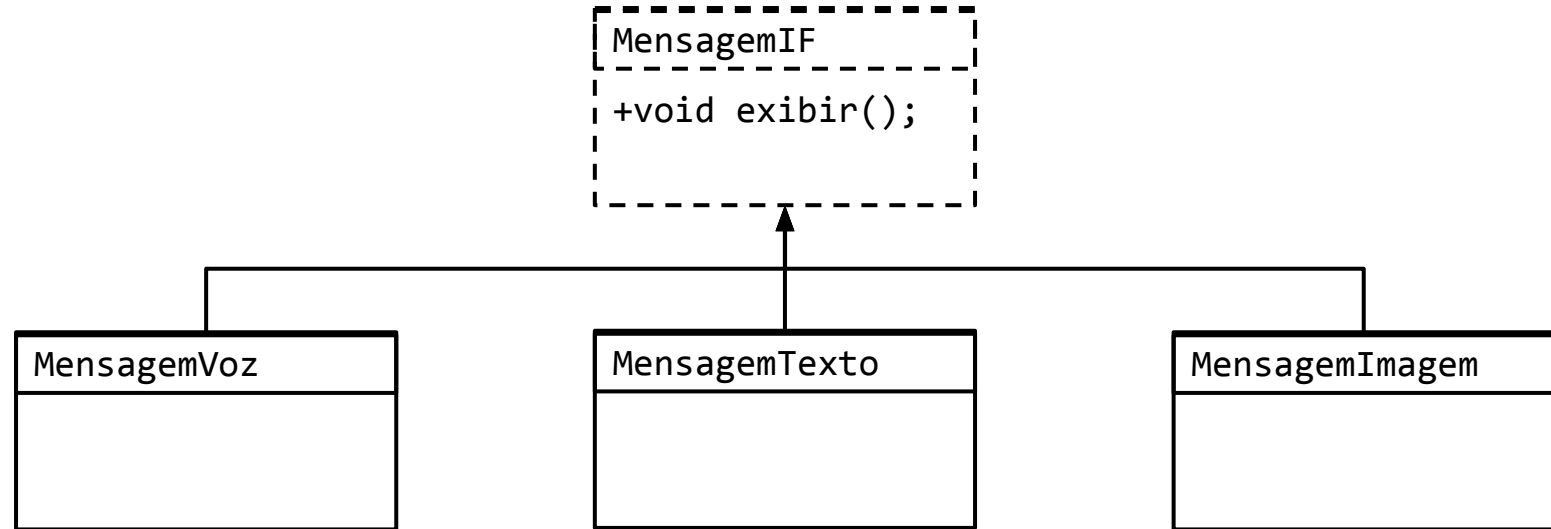
- Métodos
 - + → public
 - - → private

Unified Modelling Language



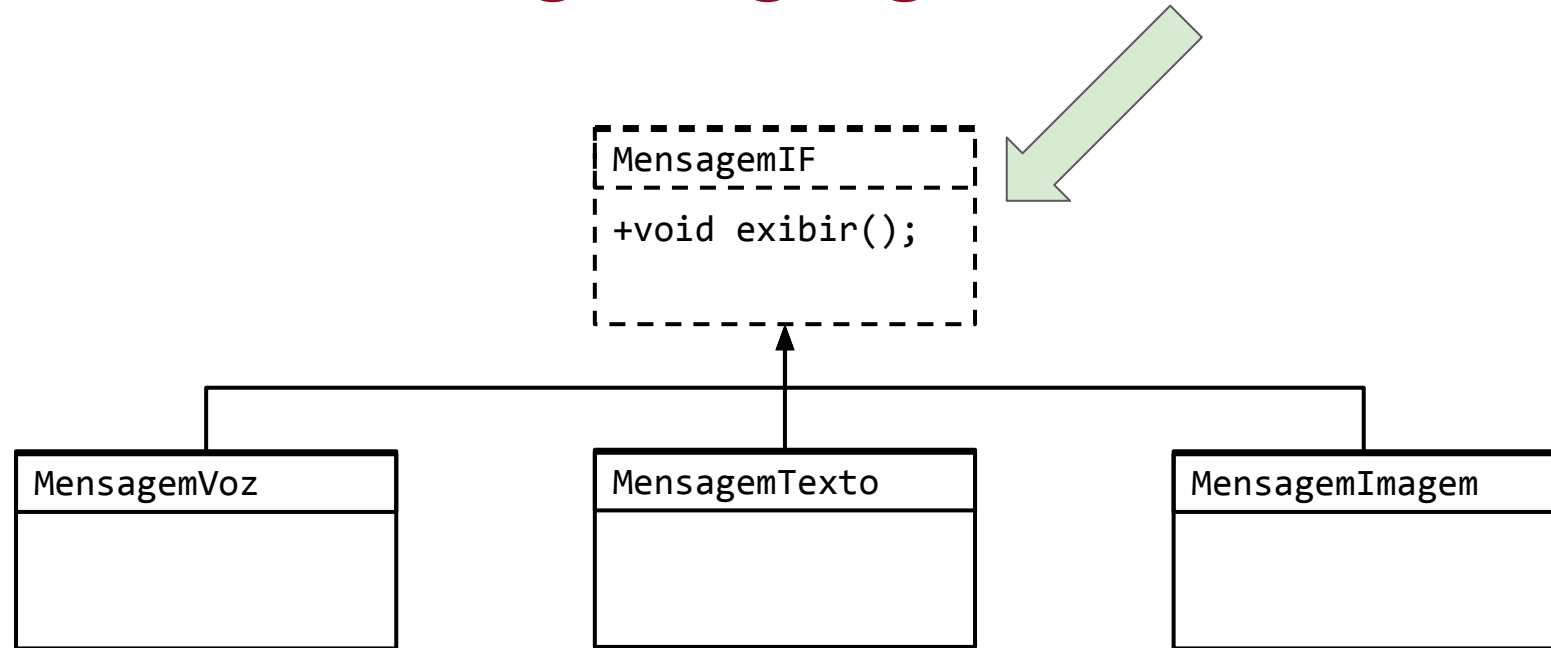
■ Entendendo o diagrama:

Unified Modelling Language



- As **classes** ajudam a definir um objeto e seu comportamento e as **interfaces** que auxiliam na definição dessas classes
- Declaração (assinatura) de métodos

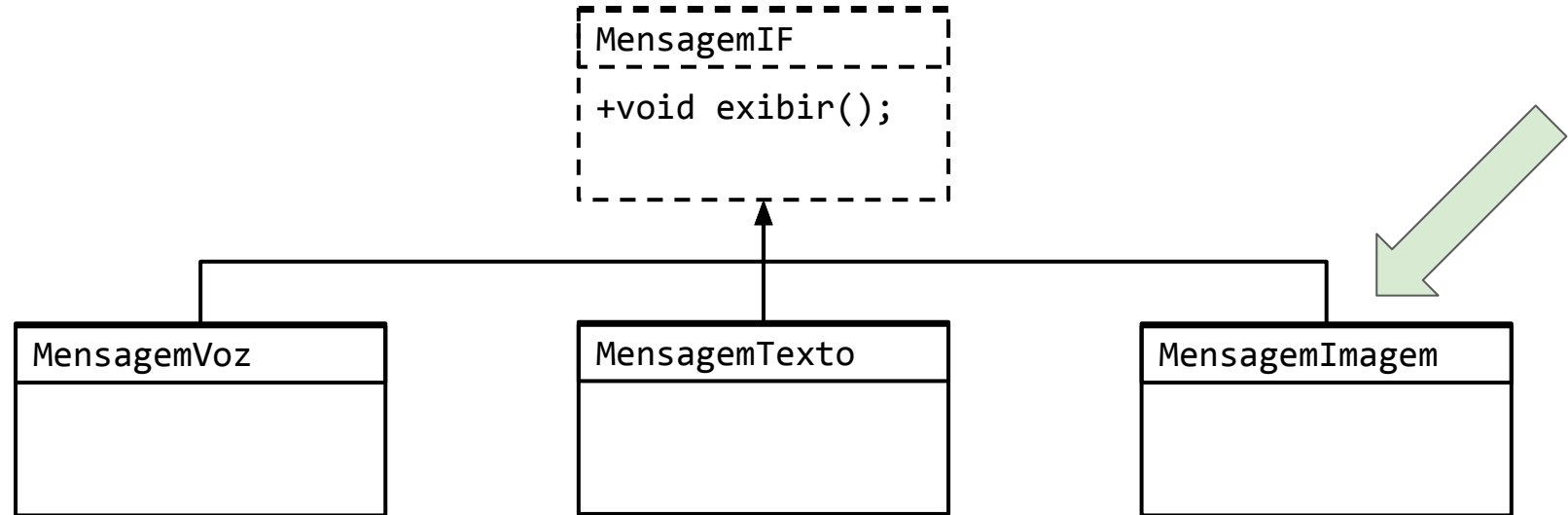
Unified Modelling Language



■ Interfaces

- Topo da hierarquia de mensagens
- Pontilhada pois nunca é implementada

Unified Modelling Language

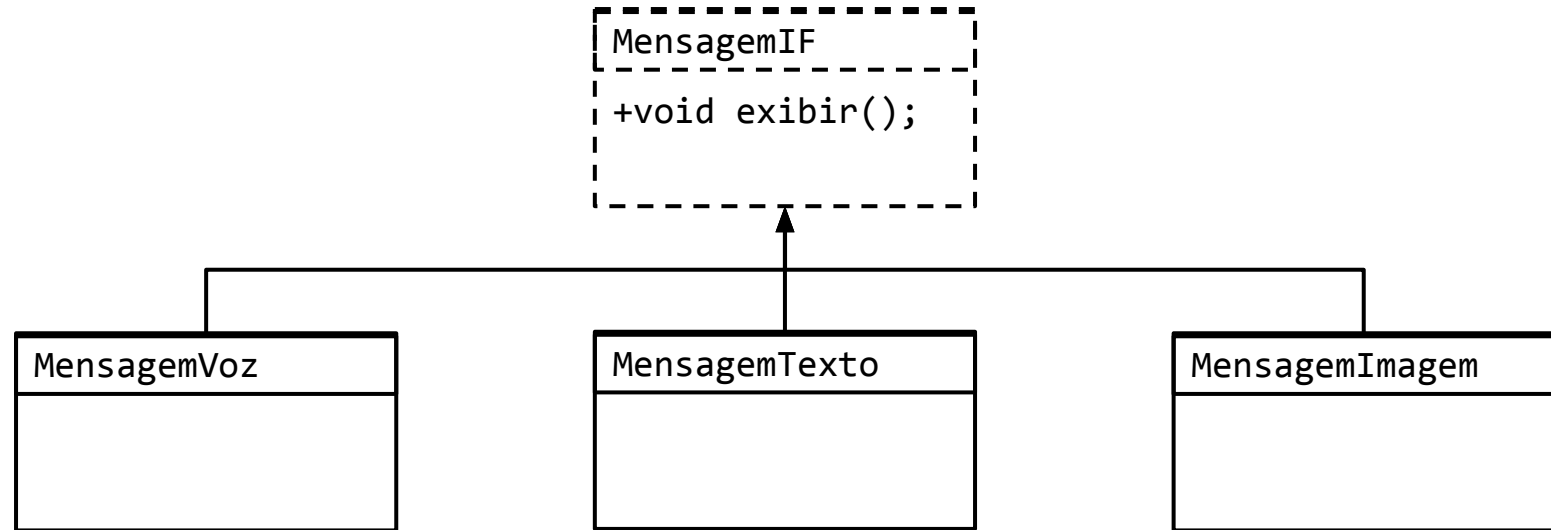


Classes:

- Já conhecemos elas

- Definem um comportamento comum

Comportamento Padrão



- Como esses objetos devem responder ao receberem o mesmo sinal: `exibir`?
- Todos respondem da mesma forma?
- Vai existir um comportamento padrão?

Polimorfismo

- Termo originário do grego
 - Poli: muitas
 - Morphos: formas
- POO
 - Objetos de classes diferentes responderem a uma mesma mensagem de diferentes maneiras
- Várias formas de responder à mensagem

Polimorfismo

- Utilizar um mesmo nome para se referir a diferentes métodos sobre um certo tipo
 - Objeto decide qual método deve ser
- Exemplo
 - Hierarquia de mensagens
 - Classe mais genérica possui o método **exibir**

Polimorfismo

- Programação voltada a tipos abstratos
- Possibilidade de um tipo abstrato (classe abstrata ou interface) ser utilizado sem que se conheça a implementação concreta
 - Independência de implementação
 - Maior foco na interface (fronteira, contrato)

Interfaces

- Definidas com métodos virtuais
- Não podem ser instanciadas
- `virtual = 0` garantem isso

```
#ifndef INF112_MENSAGEM_H
#define INF112_MENSAGEM_H

class MensagemIF {
public:
    virtual void exhibir() = 0;
};

#endif
```


Interfaces

- Não podem ser instanciadas
 - 2 linhas com erro abaixo

```
int main(void) {  
    MensagemIF msg = MensagemIF();  
    MensagemIF *msg2 = new MensagemIF();  
    MensagemIF msg3;  
}
```



Interfaces

■ Como fazer uso?!

```
int main(void) {  
    MensagemIF msg = MensagemIF();  
    MensagemIF *msg2 = new MensagemIF();  
    MensagemIF msg3;  
}
```



Implementando Interfaces

Definimos o comportamento nas classes

```
#ifndef INF112_MENSAGEMTEXTTO_H
#define INF112_MENSAGEMTEXTTO_H

#include <string>
#include "mensagem.h"

class MensagemTexto : public MensagemIF {
private:
    std::string _msg;
public:
    MensagemTexto(std::string msg);
    virtual void exibir();
};

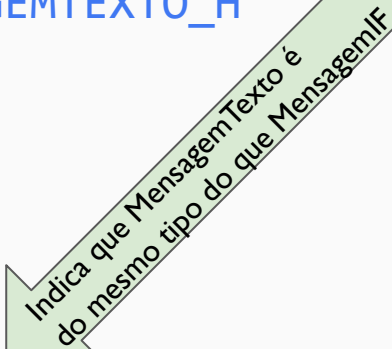
#endif
```

Implementando Interfaces

Definimos o comportamento nas classes

```
#ifndef INF112_MENSAGEMTEXTTO_H  
#define INF112_MENSAGEMTEXTTO_H
```

```
#include <string>  
#include "mensagem.h"
```



Indica que MensagemTexto é
do mesmo tipo do que MensagemIF

```
class MensagemTexto : public MensagemIF {  
private:  
    std::string _msg;  
public:  
    MensagemTexto(std::string msg);  
    virtual void exhibir();  
};
```



Método vem de uma interface

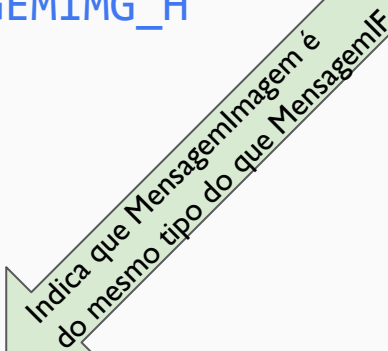
```
#endif
```

Implementando Interfaces

Podemos definir uma mensagem com imagens

```
#ifndef INF112_MENSAGEMIMG_H  
#define INF112_MENSAGEMIMG_H
```

```
#include <string>  
#include "mensagem.h"
```



Indica que MensagemImagem é do mesmo tipo do que MensagemIF

```
class MensagemImagem : public MensagemIF {  
private:  
    std::string _arquivo;  
public:  
    MensagemTexto(std::string arquivo);  
    virtual void exibir();  
};
```



Método vem de uma interface

```
#endif
```

Implementando

Mais de um tipo de mensagem

```
#include "mensagemtexto.h"

#include <iostream>

MensagemTexto::MensagemTexto(std::string msg) {
    this->_msg = msg;
}

void MensagemTexto::exibir() {
    std::cout << this->_msg;
    std::cout << std::endl;
}
```

```
#include "mensagemimg.h"
```

```
#include <fstream>
#include <iostream>
```

```
MensagemImagem::MensagemImagem(std::string arquivo) {
    this->_arquivo = arquivo;
}

void MensagemImagem::exibir() {
    std::ifstream arquivo(this->_arquivo);
    std::string line;
    while (std::getline(arquivo, line))
        std::cout << line << std::endl;
    arquivo.close();
}
```

Implementando

Mais de um tipo de mensagem

```
#include "mensagemtexto.h"
```

```
#include <iostream>
```

```
MensagemTexto::MensagemTexto(std::string msg) {  
    this->_msg = msg;  
}
```

```
void MensagemTexto::exibir() {  
    std::cout << this->_msg;  
    std::cout << std::endl;  
}
```

Exibe um texto

```
#include "mensagemimg.h"
```

```
#include <fstream>
```

```
#include <iostream>
```

```
MensagemImagem::MensagemImagem(std::string arquivo) {  
    this->_arquivo = arquivo;  
}
```

```
void MensagemImagem::exibir() {  
    std::ifstream arquivo(this->_arquivo);  
    std::string line;  
    while (std::getline(arquivo, line))  
        std::cout << line << std::endl;  
    arquivo.close();  
}
```

Exibe uma imagem ascii

Polimorfismo em ação

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```


Polimorfismo em ação

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
Oi, tem aula de INF112 hoje?
```

Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
Oi, tem aula de INF112 hoje?
```

Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
Oi, tem aula de INF112 hoje?
Tocando o arquivo... audio.wav
```

Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
Oi, tem aula de INF112 hoje?
Tocando o arquivo... audio.wav
```

Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
Oi, tem aula de INF112 hoje?
Tocando o arquivo... audio.wav
```

Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
Oi, tem aula de INF112 hoje?
Tocando o arquivo... audio.wav

      /      \
    :  o  o  ;
   (    ( _    )
  :                ;
 \      _      /
  \  . _ _ _ . /
   /`""""`\
  /      ,      \
 /|/\|/\| _ \|
(_|/\|/\|/\|_)
 |_____|
 _)_ _ | _ ( _
 (_____|_____)

```


Erros Comuns

- Tentar usar o tipo genérico na declaração
- Erro de compilação
 - Tipos com tamanhos diferentes (assinatura)



```
void exibir_na_tela(MensagemIF &msg) {  
    msg.exibir();  
}  
  
int main(void) {  
    MensagemIF texto = MensagemTexto("Oi, tem aula de PDS2 hoje?");  
    MensagemIF audio = MensagemVoz("audio.wav");  
  
    exibir_na_tela(texto);  
    exibir_na_tela(audio);  
}
```

Solução (se necessário)

- Ponteiros
- Sempre tem um tamanho fixo

```
#include "mensagem.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF *msg) {
    msg->exibir();
}

int main(void) {
    MensagemIF *texto = new MensagemTexto("Oi, tem aula de PDS2 hoje?");
    MensagemIF *audio = new MensagemVoz("audio.wav");
    exibir_na_tela(texto);
    exibir_na_tela(audio);
    delete texto;
    delete audio;
}
```

Exercício I



Exercício I

```
class Animal
{
    public:
        virtual void fale();
};
```

```
class Gato : public Animal
{
    public:
        void fale() override {
            cout << "Miau!" << endl;
        }
};
```

```
class Cachorro : public Animal
{
    public:
        void fale() override {
            cout << "Au!Au!" << endl;
        }
};
```

Exercício I

```
int main()
{
    Cachorro c;
    c.fale();

    Gato g;
    g.fale();

    return 0;
};
```

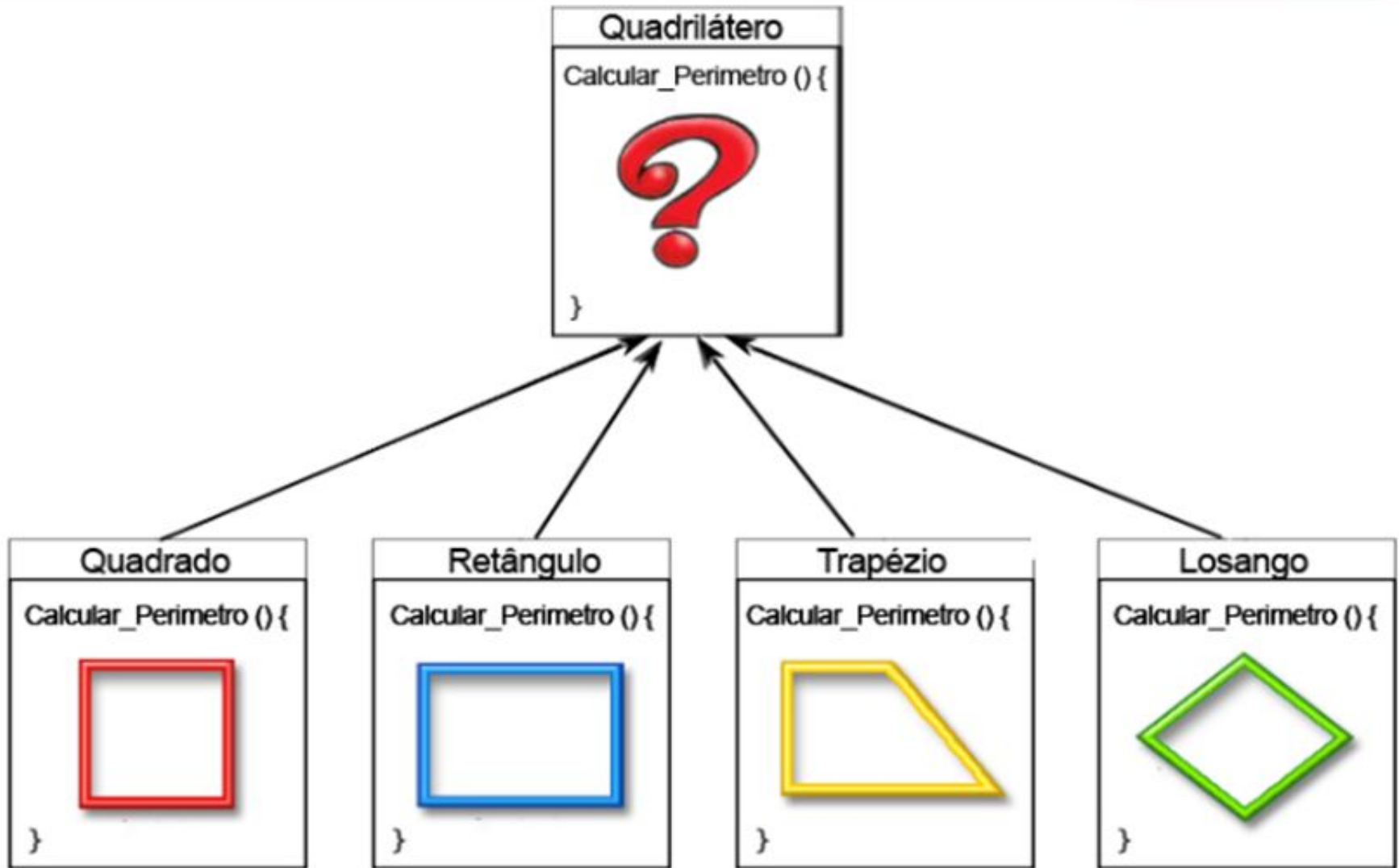
Exercício I

```
int main()
{
    Animal * c = new Cachorro();
    c -> fale();
    delete c;

    Animal * g = new Gato();
    g -> fale();
    delete g;

    return 0;
};
```

Exercício 2

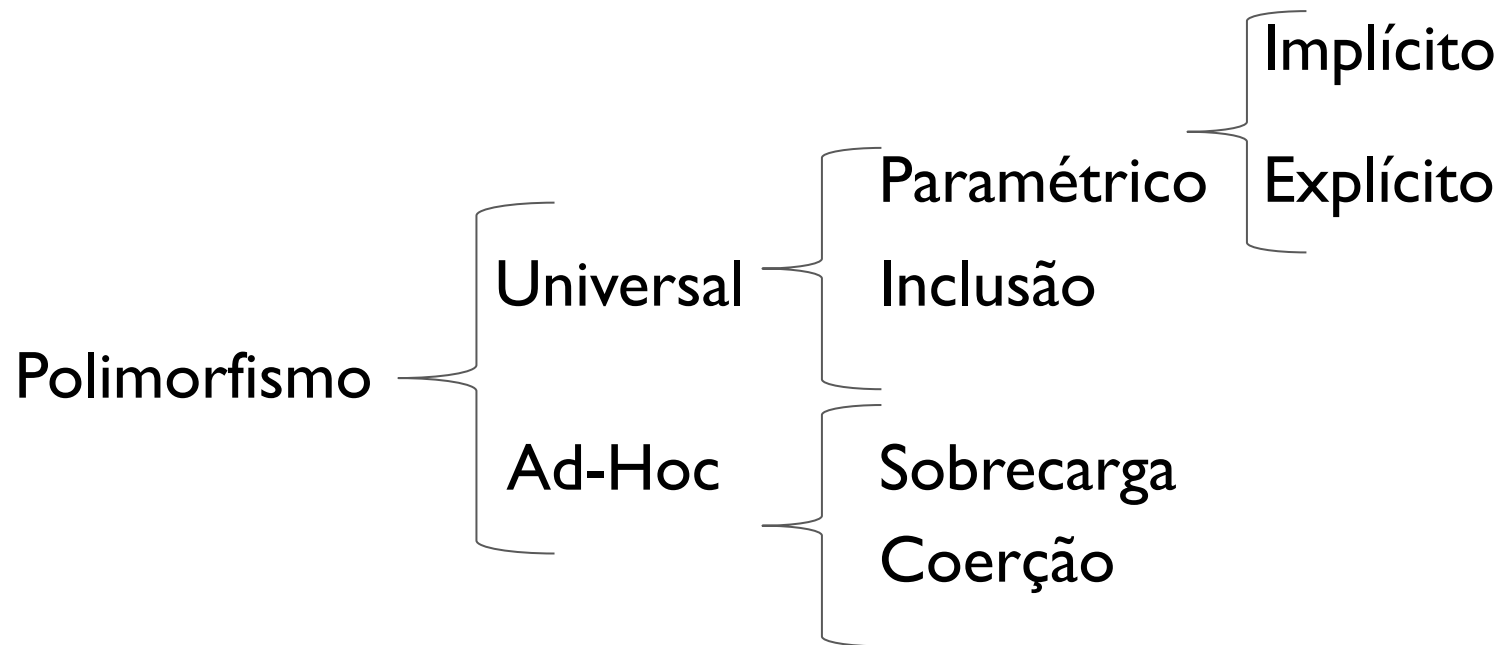


Tipos de Polimorfismo

Polimorfismo

- Seleção da instância (forma) do objeto
 - Ligação Prematura (Early binding)
 - As decisões são feitas durante a compilação
 - Ligação Tardia (Late binding)
 - As decisões são feitas durante a execução
 - É a chave para o funcionamento do polimorfismo
- C++ \Rightarrow Padrão é ligação prematura
 - Ligação tardia utiliza o comando “virtual”

Polimorfismo



Tipos de Polimorfismo

Universal

- Universal ou Verdadeiro
 - Quando uma função ou tipo trabalha de maneira uniforme para uma gama de tipos definidos na linguagem
- A mesma definição (código) de uma função pode ser utilizada por diferentes tipos
- Potencialmente número infinito de variações

Tipos de Polimorfismo

Universal Paramétrico

- Torna a linguagem mais expressiva
 - Templates em C++
- Universal paramétrico
 - Os tipos são identificados pelo compilador
 - São passados implicitamente à função

Tipos de Polimorfismo

Early Binding

```
#include <list>
```

```
int main() {
```

```
    std::list<Pessoa> lista;
```

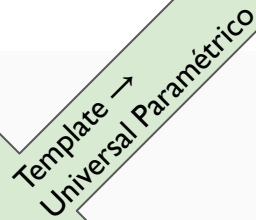
```
    Pessoa p;
```

```
    lista.push_back(p);
```

```
    std::cout << lista.size() << std::endl;
```

```
    return 0;
```

```
}
```



Template →
Universal Paramétrico

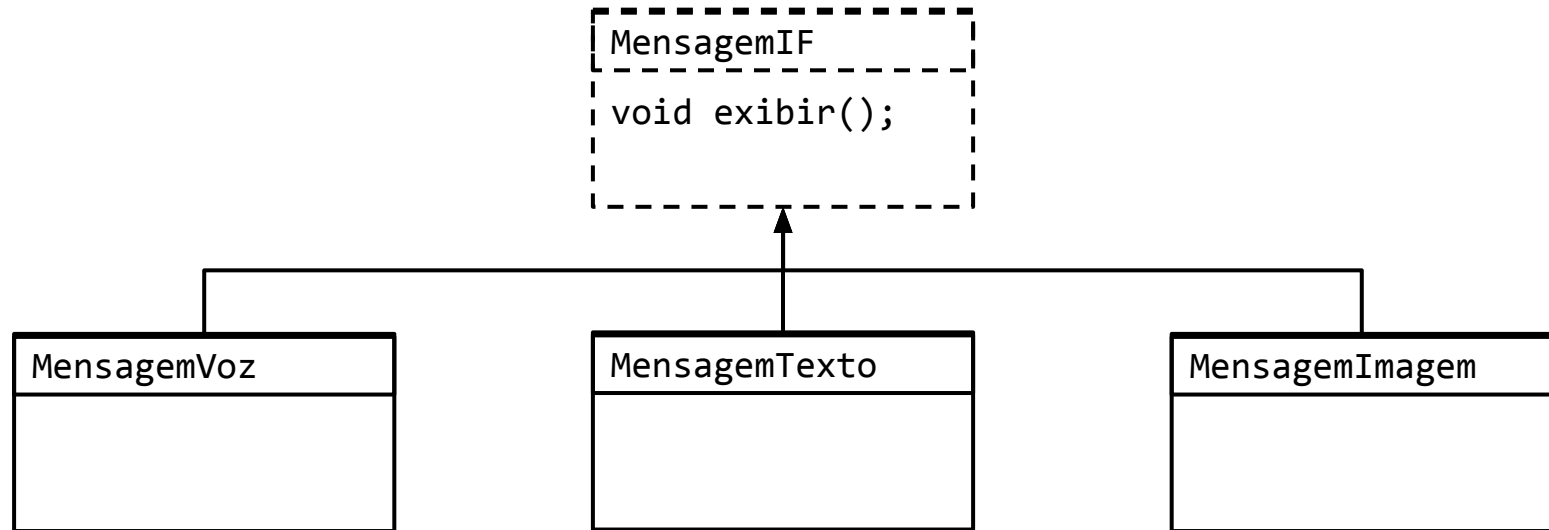
Tipos de Polimorfismo

Late Binding (Universal - Inclusão)

- **Modela subtipos**
 - Redefinição em classes descendentes
 - O subtipo está incluído no próprio tipo
- **Onde um objeto de um tipo for esperado, um objeto do subtipo deve ser aceito**
- **Princípio da substituição de Liskov**
 - se S é um subtipo de T, então os objetos do tipo T, em um programa, podem ser substituídos pelos objetos de tipo S sem que seja necessário alterar as propriedades deste programa.
- **O contrário nem sempre é válido!**

Tipos de Polimorfismo

Universal - Inclusão

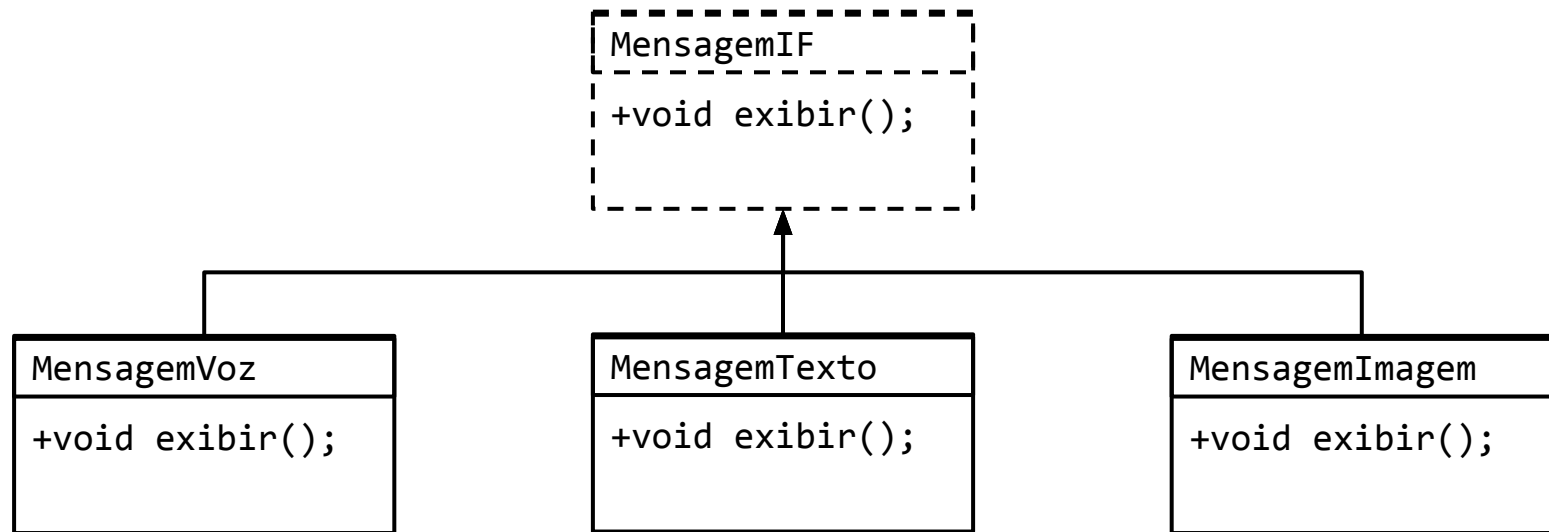


```
void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}
```

■ Podemos passar qualquer subtipo de **MensagemIF** para o método acima

Tipos de Polimorfismo

Universal - Inclusão

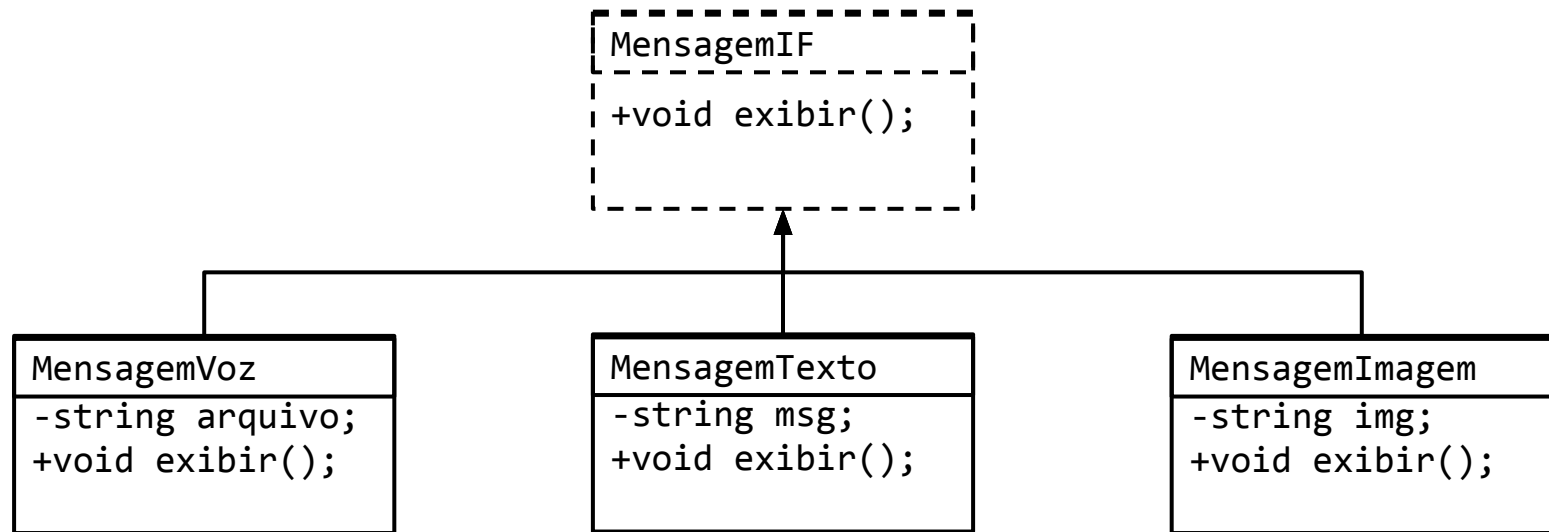


```
void exibir_na_tela(MensagemIF &msg) {  
    msg.exibir();  
}
```

- Todas suportam o `exibir`

Tipos de Polimorfismo

Universal - Inclusão



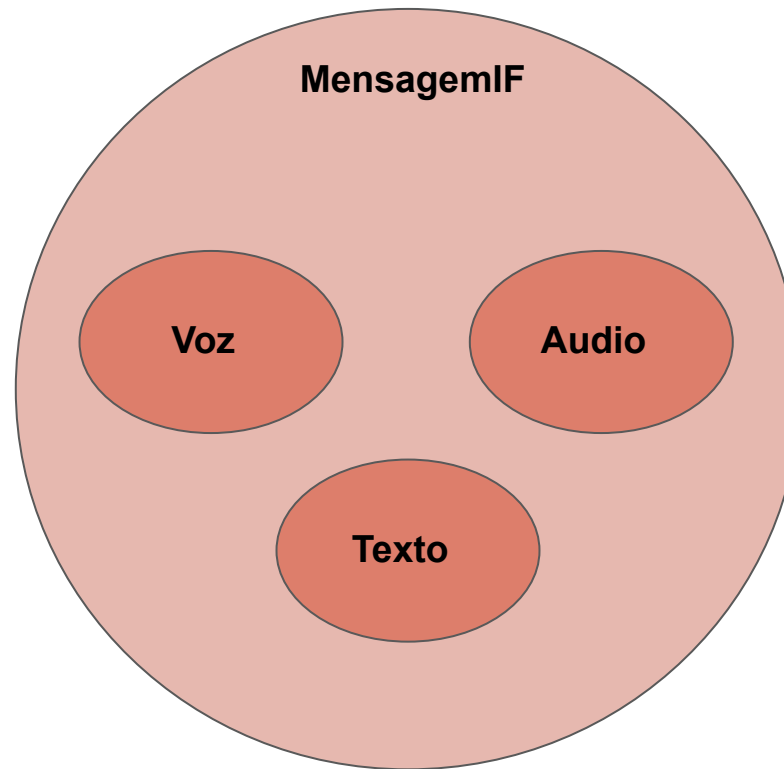
```
void exibir_na_tela(MensagemIF &msg) {  
    msg.exibir();  
}
```

- Porém cada uma tem comportamento interno (private) diferente

Tipos de Polimorfismo

Inclusão

Contexto de Tipos



Tipos de Polimorfismo

Ad-hoc - Sobre carga

- Número finito de entidades distintas, todas com mesmo nome, mas códigos distintos
- Função ou valor conforme o contexto

Tipos de Polimorfismo

Ad-hoc - Sobrecarga

- O mesmo identificador denota diferentes funções que operam sobre tipos distintos
- Resolvido estaticamente (compilação)
 - Considera os tipos para escolher a definição
 - Difere no número e no tipo dos parâmetros

Tipos de Polimorfismo

Ad-hoc - Sobrecarga

```
class Ponto {  
private:  
    double _x = 0;  
    double _y = 0;  
public:  
    void set_xy(double x, double y) {  
        this->_x = x;  
        this->_y = y;  
    }  
  
    void set_xy(double xy) {  
        this->_x = xy;  
        this->_y = xy;  
    }  
};
```

Tipos de Polimorfismo

Ad-hoc - Coerção

- Conversão automática de tipo
 - Utilizada para satisfazer o contexto atual
 - Considera a definição para escolher o tipo
- Linguagem possui um mapeamento interno

Conversão de Tipos (Casting)

Caso de Estudo 2

Coleções

```
#ifndef INF112_LISTADUPLA_H
#define INF112_LISTADUPLA_H

struct node_t {
    int elemento;
    node_t *anterior;
    node_t *proximo;
};

class ListaDuplamenteEncadeada {
private:
    node_t *_inicio;
    node_t *_fim;
    int _num_elementos_inseridos;
public:
    ListaDuplamenteEncadeada();
    ~ListaDuplamenteEncadeada();
    void inserir_elemento(int elemento);
    void imprimir();
    int tamanho();
    void remove_iesimo(int i);
};
#endif
```

```
#ifndef INF112_BST_H
#define INF112_BST_H

#include "node.h"

class BST {
private:
    Node *_raiz;
    int _num_elementos_inseridos;
public:
    BST();
    ~BST();
    void inserir_elemento(int elemento);
    void imprimir();
    int tamanho();
    bool tem_elemento(int elemento);
};

#endif
```


Caso de Estudo 2

Note três comportamentos repetidos: insere, imprime, tamanho

```
#ifndef INF112_LISTADUPLA_H
#define INF112_LISTADUPLA_H

struct node_t {
    int elemento;
    node_t *anterior;
    node_t *proximo;
};

class ListaDuplamenteEncadeada {
private:
    node_t *_inicio;
    node_t *_fim;
    int _num_elementos_inseridos;
public:
    ListaDuplamenteEncadeada();
    ~ListaDuplamenteEncadeada();
    void inserir_elemento(int elemento);
    void imprimir();
    int tamanho();
    void remove_iesimo(int i);
};
#endif
```

```
#ifndef INF112_BST_H
#define INF112_BST_H

#include "node.h"

class BST {
private:
    Node *_raiz;
    int _num_elementos_inseridos;
public:
    BST();
    ~BST();
    void inserir_elemento(int elemento);
    void imprimir();
    int tamanho();
    bool tem_elemento(int elemento);
};

#endif
```

Agregando o comportamento similar

Fazendo uso de interfaces

■ Note o destrutor virtual

```
#ifndef INF112_COLECAO_H
#define INF112_COLECAO_H

class ColecaoIF {
public:
    virtual ~ColecaoIF() {};
    virtual void inserir_elemento(int elemento) = 0;
    virtual void imprimir() = 0;
    virtual int tamanho() = 0;
};

#endif
```

Agregando o comportamento similar

Fazendo uso de interfaces

- Note o destrutor virtual
 - Às vezes precisamos chamar o destrutor quando temos um tipo da interface

```
#ifndef INF112_COLECAO_H
#define INF112_COLECAO_H
class ColecaoIF {
public:
    virtual ~ColecaoIF() {};
    virtual void inserir_elemento(int elemento) = 0;
    virtual void imprimir() = 0;
    virtual int tamanho() = 0;
};
#endif
```

Pulando para o main

Uso do destrutor virtual

```
#include "colecao.h"
#include "listadupla.h"
#include "bst.h"

int main(void) {
    ColecaoIF *lista = new ListaDuplamenteEncadeada();
    lista->inserir_elemento(2);
    lista->inserir_elemento(3);

    ColecaoIF *bst = new BST();
    bst->inserir_elemento(10);
    bst->inserir_elemento(-1);
    bst->inserir_elemento(6);

    lista->imprimir();
    bst->imprimir();

    delete lista;
    delete bst;
    return 0;
}
```



lista e bst são ColecaoIF

Agregando o comportamento similar

Fazendo uso de interfaces

- O destrutor virtual é um código sem nada
- Podemos implementar o mesmo caso tenha um comportamento comum (isso é raro)

```
#ifndef INF112_COLECAO_H
#define INF112_COLECAO_H
class ColecaoIF {
public:
    virtual ~ColecaoIF() {};
    virtual void inserir_elemento(int elemento) = 0;
    virtual void imprimir() = 0;
    virtual int tamanho() = 0;
};
#endif
```

Olhando para as implementações

Caso da Lista

```
#ifndef INF112_LISTADUPLA_H
#define INF112_LISTADUPLA_H

// . . .


class ListaDuplamenteEncadeada : public ColecaoIF {
private:
    node_t *_inicio;
    node_t *_fim;
    int _num_elementos_inseridos;
public:
    ListaDuplamenteEncadeada();

    // Comportamento comum
    virtual ~ListaDuplamenteEncadeada();
    virtual void inserir_elemento(int elemento);
    virtual void imprimir();
    virtual int tamanho();

    // Específico
    void remove_iesimo(int i);
};
#endif
```



Podemos usar via ColecaoIF/Lista



Podemos usar apenas via Lista

Conversão de Tipos

- Uma classe, ao herdar de outra, assume o tipo desta onde quer que seja necessário
- Upcasting
 - Conversão para uma classe mais genérica
- Downcasting
 - Conversão para uma classe mais específica

Conversão de Tipos

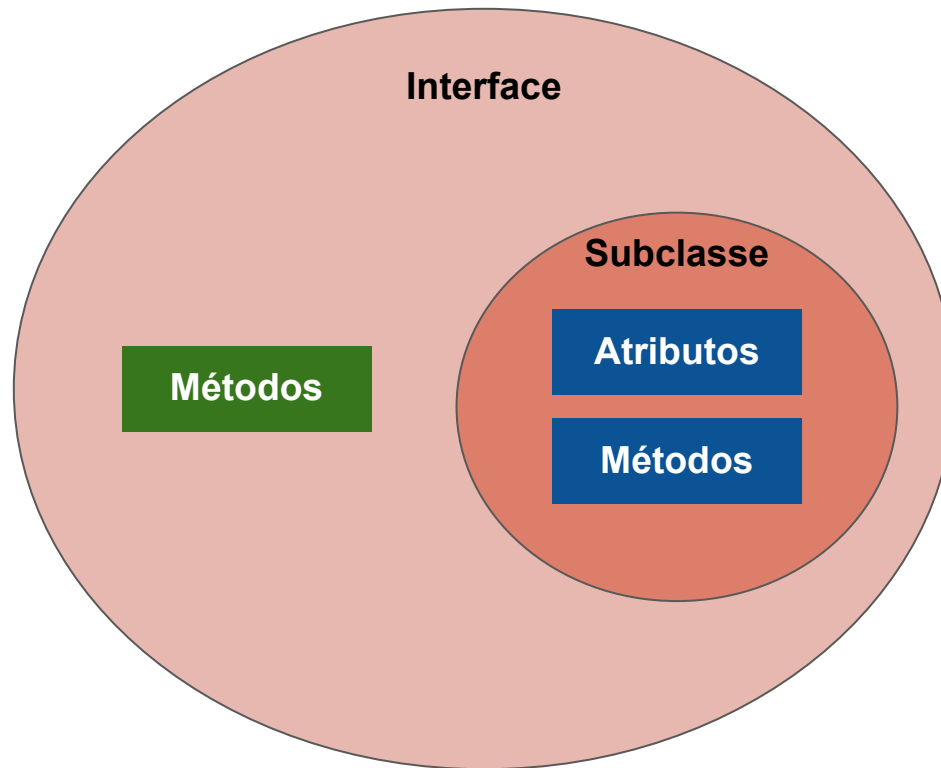
Upcasting

- Ocorre no sentido Classe \Rightarrow Interface
- Não há necessidade de indicação explícita
- A classe derivada sempre vai manter as características públicas da superclasse

Conversão de Tipos

Upcasting

Contexto de Classe



Upcasting

Note que os objetos viram MensagemIF auto-magicamente

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de INF112 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :(");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```



Conversão de Tipos

Downcasting

- Ocorre no sentido Interface \Rightarrow Classe
- Não é feito de forma automática!
- Deve-se deixar explícito, informando o nome do subtipo antes do nome da variável

Conversão de Tipos

Downcasting

- Nem sempre uma superclasse poderá assumir o tipo de uma subclasse
- Toda MensagemTexto é uma MensagemIF
- Nem toda MensagemIF é MensagemTexto
- Caso não seja possível
 - Segmentation fault

Mensagens++

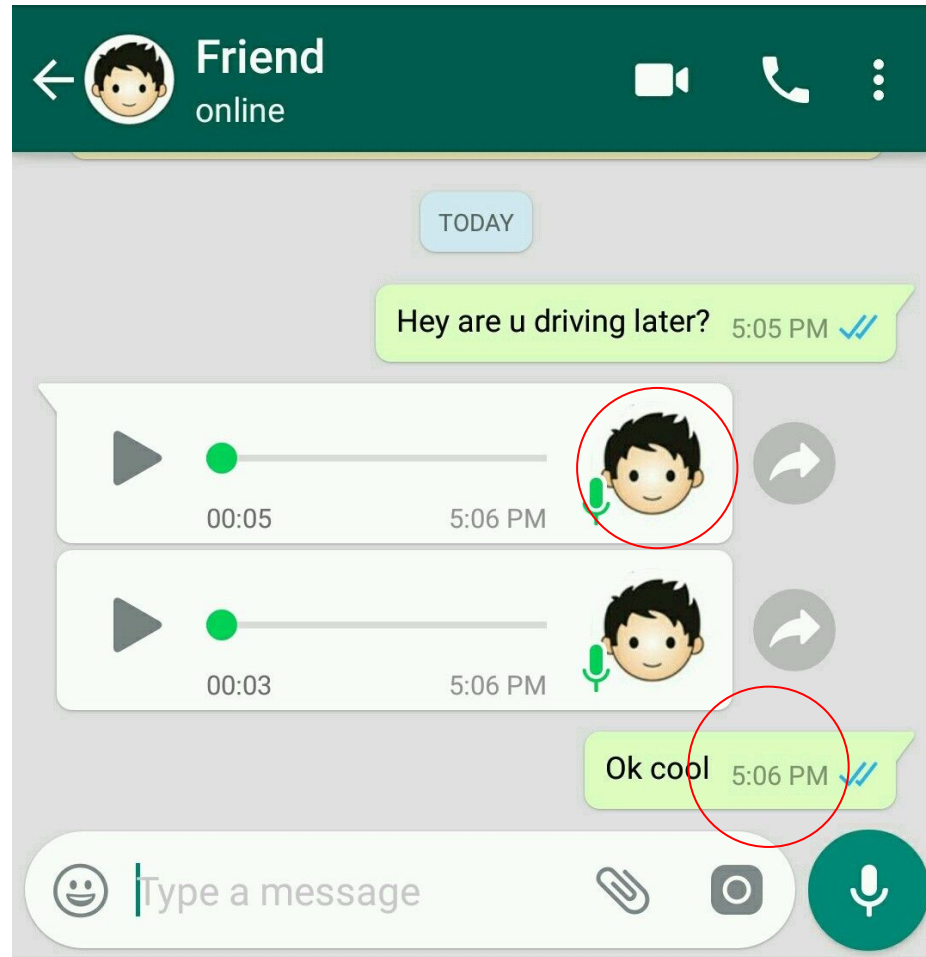
Voltando para o caso das mensagens

Qual o comportamento comum?



Voltando para o caso das mensagens

Avatar, Datas



Redefinindo a Mensagem


3 tipos de métodos

```
#ifndef INF112_MENSAGEM_H
#define INF112_MENSAGEM_H
#include <string>
#include <ctime>
class MensagemBase {
private:
    std::time_t _data;
    std::string _avatar;
public:
    MensagemBase(std::string avatar);
    std::time_t get_data() const;
    virtual std::string get_avatar() const;
    virtual void exhibir() const = 0;
};
#endif
```


Redefinindo a Mensagem

time_t representa datas em C++ (segundos desde 01/01/1970)


```
#ifndef INF112_MENSAGEM_H
#define INF112_MENSAGEM_H
#include <string>
#include <ctime>
class MensagemBase {
private:
    std::time_t _data;
    std::string _avatar;
public:
    MensagemBase(std::string avatar);
    std::time_t get_data() const;
    virtual std::string get_avatar() const;
    virtual void exhibir() const = 0;
};
#endif
```



Redefinindo a Mensagem

Método que não é virtual, sem late binding


```
#ifndef INF112_MENSAGEM_H
#define INF112_MENSAGEM_H
#include <string>
#include <ctime>
class MensagemBase {
private:
    std::time_t _data;
    std::string _avatar;
public:
    MensagemBase(std::string avatar);
    std::time_t get_data() const;
    virtual std::string get_avatar() const;
    virtual void exhibir() const = 0;
};
#endif
```



Redefinindo a Mensagem

Nunca vamos redefinir, não precisamos de **virtual**

```
#ifndef INF112_MENSAGEM_H
#define INF112_MENSAGEM_H
#include <string>
#include <ctime>
class MensagemBase {
private:
    std::time_t _data;
    std::string _avatar;
public:
    MensagemBase(std::string avatar);
    std::time_t get_data() const;
    virtual std::string get_avatar() const;
    virtual void exibir() const = 0;
};
#endif
```



Redefinindo a Mensagem

virtual, será re-definido

```
#ifndef INF112_MENSAGEM_H
#define INF112_MENSAGEM_H
#include <string>
#include <ctime>
class MensagemBase {
private:
    std::time_t _data;
    std::string _avatar;
public:
    MensagemBase(std::string avatar);
    std::time_t get_data() const;
    virtual std::string get_avatar() const;
    virtual void exhibir() const = 0;
};
#endif
```



Sobre o get_avatar

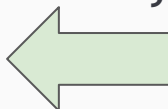
Note que as mensagens de texto não tem avatar. Sobreescrever



Redefinindo a Mensagem

= 0; forçadamente tem que aparecer na sub-classe.

```
#ifndef INF112_MENSAGEM_H
#define INF112_MENSAGEM_H
#include <string>
#include <ctime>
class MensagemBase {
private:
    std::time_t _data;
    std::string _avatar;
public:
    MensagemBase(std::string avatar);
    std::time_t get_data() const;
    virtual std::string get_avatar() const;
    virtual void exhibir() const = 0;
};
#endif
```



Sobreescrevendo

A mensagem de voz "remove" o avatar

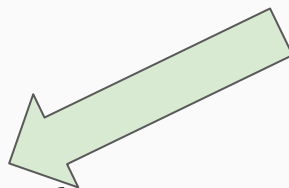
```
#include "mensagemtexto.h"
```

```
#include <iostream>
```

```
MensagemTexto::MensagemTexto(std::string avatar, std::string msg):  
    MensagemBase(avatar), _msg(msg) {}
```

```
void MensagemTexto::exibir() const {  
    std::cout << this->_msg;  
    std::cout << std::endl;  
}
```

```
std::string MensagemTexto::get_avatar() const {  
    return "";  
}
```



Sobreescrevendo


Porém é forçada a implementar o `exibir (=0)`

```
#include "mensagemtexto.h"
```

```
#include <iostream>
```

```
MensagemTexto::MensagemTexto(std::string avatar, std::string msg):  
    MensagemBase(avatar), _msg(msg) {}
```

```
void MensagemTexto::exibir() const {  
    std::cout << this->_msg;  
    std::cout << std::endl;  
}
```



```
std::string MensagemTexto::get_avatar() const {  
    return "";  
}
```


Mensagens de Audio

Usam o avatar default. Sobreescrevem o `exibir` apenas

```
#include "mensagemvoz.h"
```

```
#include <iostream>
```

```
MensagemVoz::MensagemVoz(std::string avatar, std::string arquivo):  
    MensagemBase(avatar), _arquivo(arquivo) {}
```



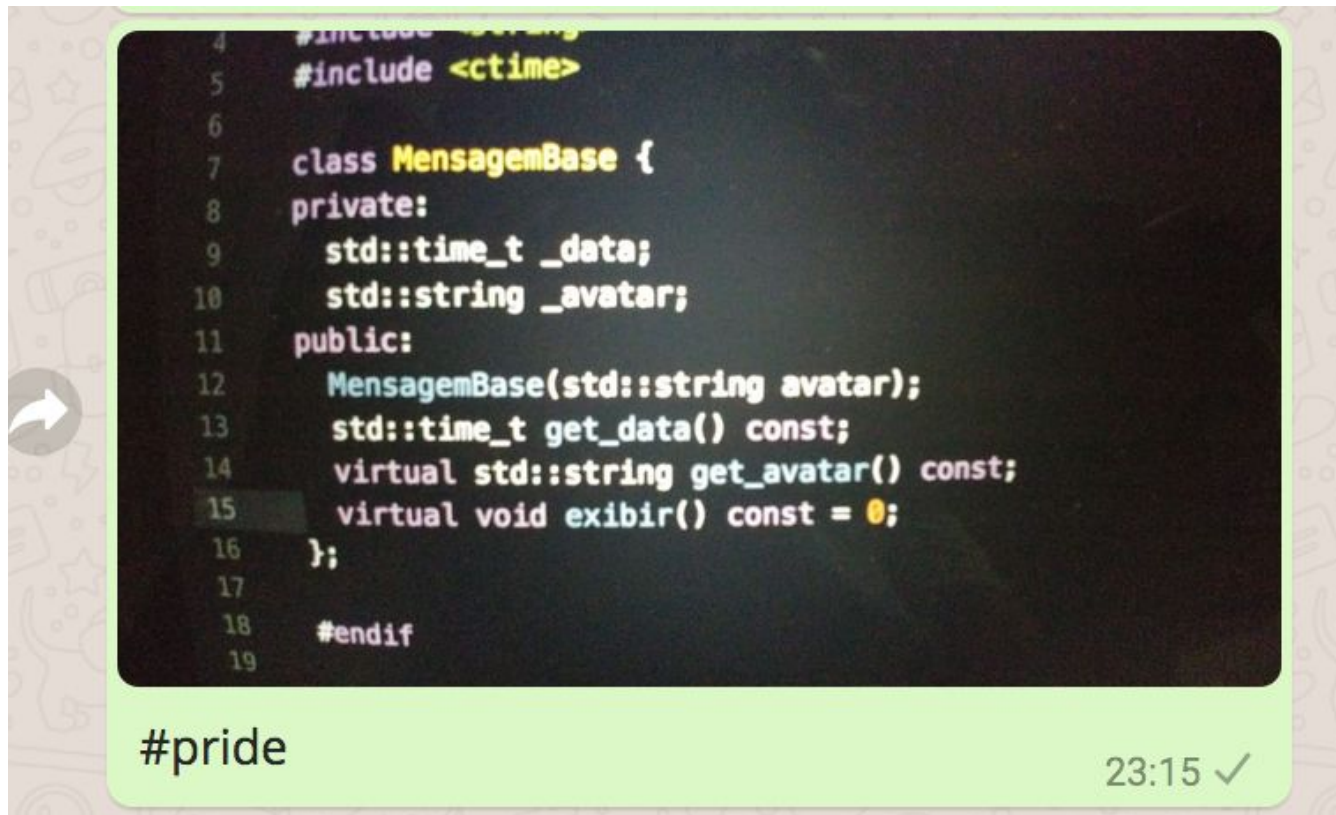
```
void MensagemVoz::exibir() const {  
    std::cout << "Tocando o arquivo... ";  
    std::cout << this->_arquivo;  
    std::cout << std::endl;  
}
```

Herança múltipla (Relembrando...)

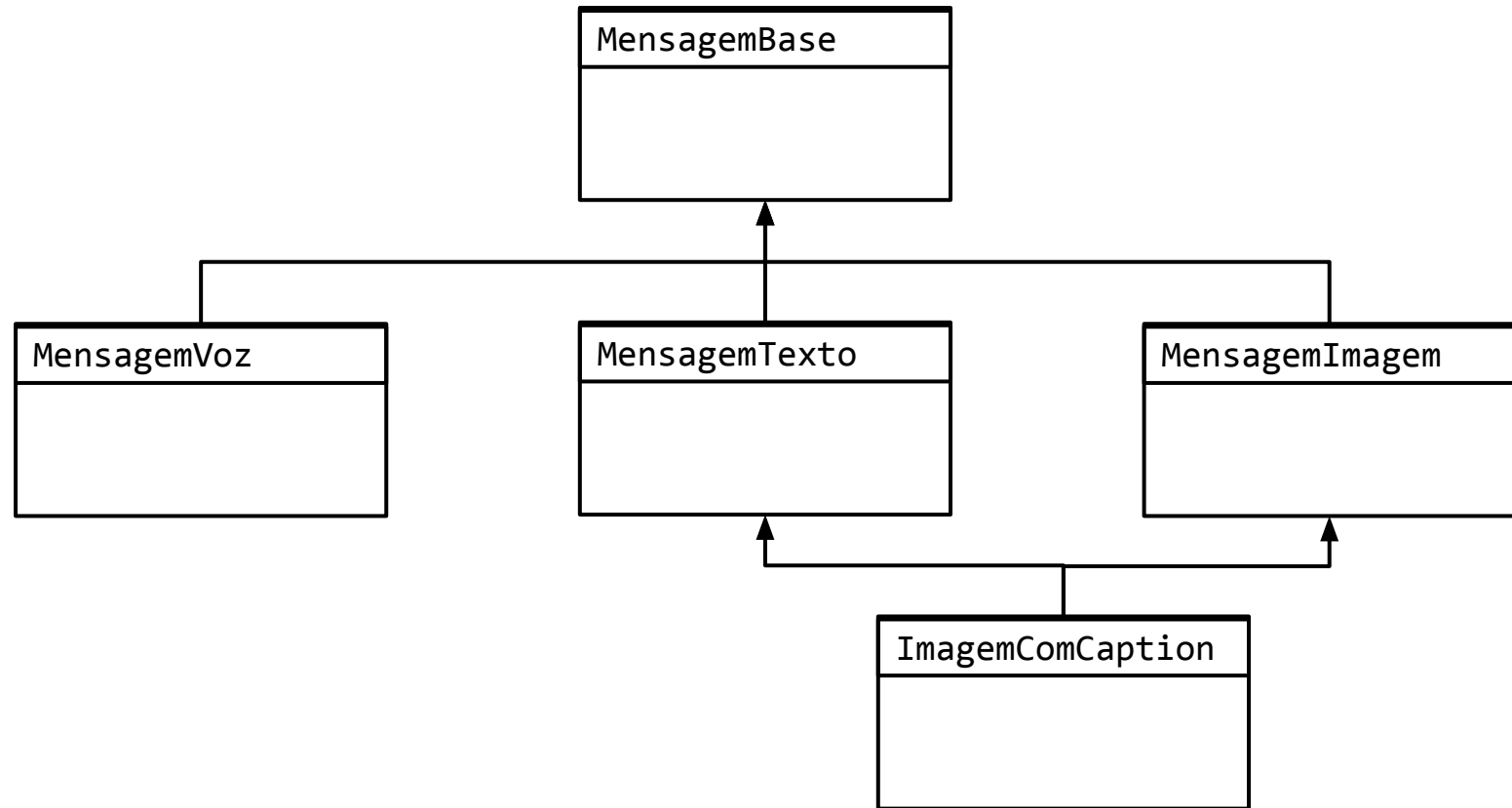
- Subclasse herda de mais de uma superclasse
 - Nem todas as linguagens permitem isso
- Problemas
 - Dificulta a manutenção do sistema
 - Também dificulta o entendimento
 - Reduz a modularização (super objetos)
 - Classes que herdam de todo mundo
 - Saída do preguiçoso

Imagens com Captions

- Conceito que existe no WhatsApp



Herança múltipla



Herança múltipla

- Possível em C++
- Nunca use.

```
class ImagemComCaption : public MensagemImagem, public MensagemTexto {  
  
};
```

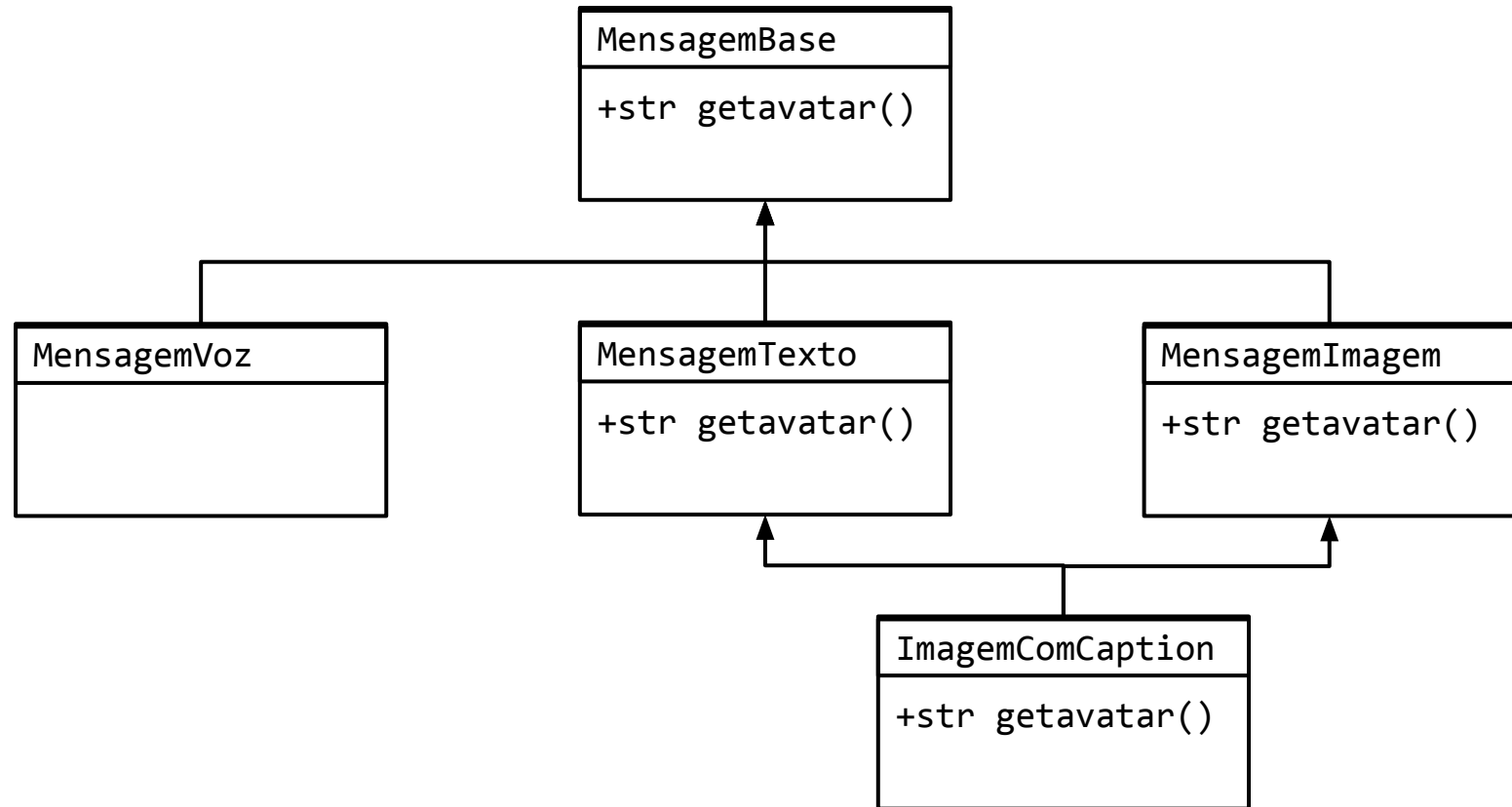
Herança múltipla

- Possível em C++
- Nunca use.
- Sério.

```
class ImagemComCaption : public MensagemImagem, public MensagemTexto {  
  
};
```

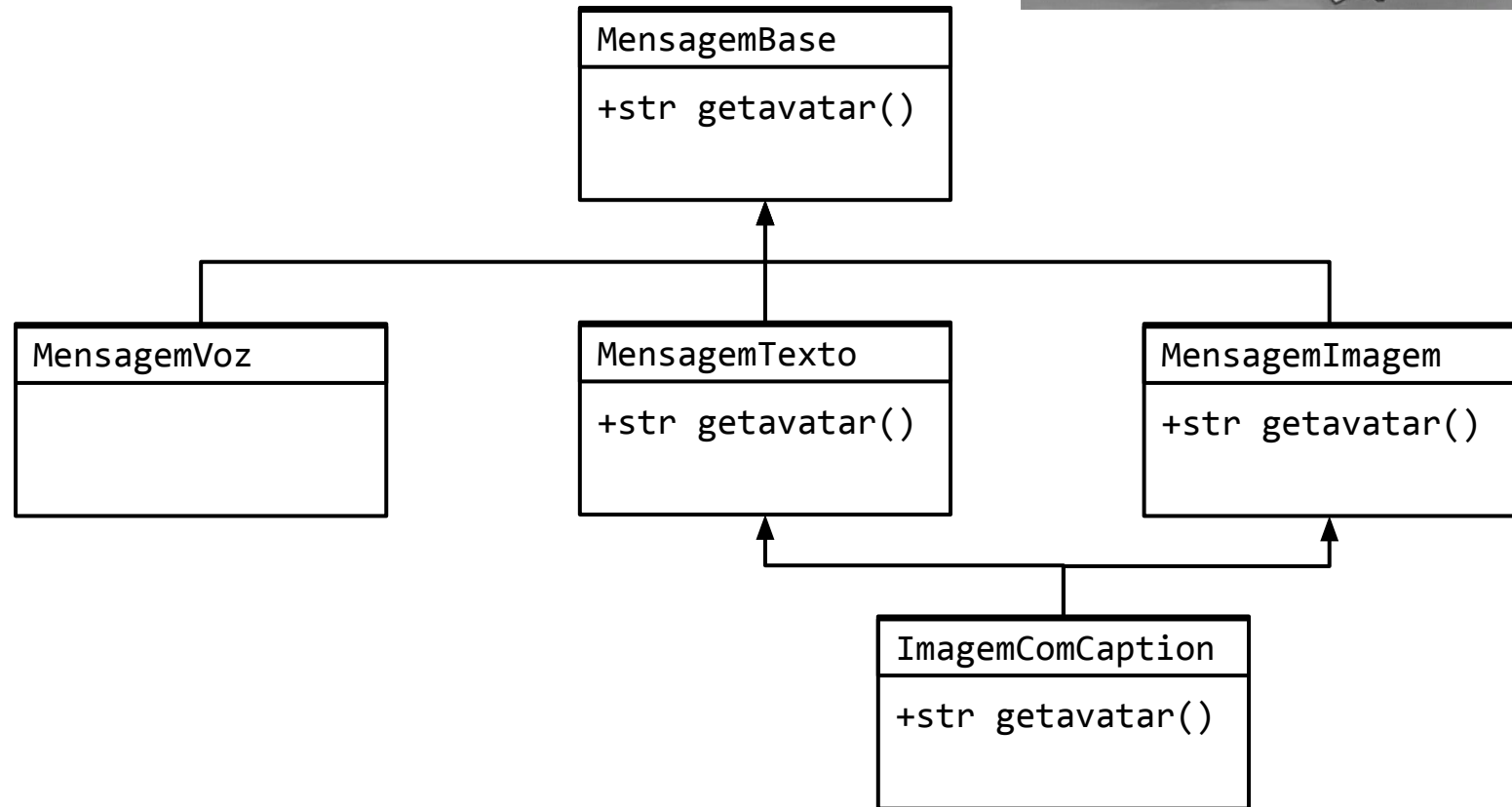
Herança múltipla

Existem 4 definições de `get_avatar` para `ImagemComCaption`



Herança múltipla

Qual vai ser utilizado!?



Herança

Críticas

- “Fere” o princípio do encapsulamento
 - Membros fazem parte de várias classes
- Cria interdependência entre classes
 - Mudanças em superclasses podem ser difíceis
- Como resolver isso?

Herança

Críticas

- “Fere” o princípio do encapsulamento
 - Membros fazem parte de várias classes
- Cria interdependência entre classes
 - Mudanças em superclasses podem ser difíceis
- Como resolver isso?

Composition is often more appropriate than inheritance.
When using inheritance, make it public.

– Google C++ Style Guide

Herança vs. Composição

- Herança

- Relação do tipo “é um” (is-a)
- Subclasse tratada como a superclasse
- Estudante é uma Pessoa

- Composição

- Relação do tipo “tem um” (has-a)
- Objeto possui objetos (≥ 1) de outras classes
- Estudante tem um Curso

Composição

- Técnica para criar um novo tipo não pela derivação, mas pela junção de outras classes de menor complexidade
- Não existe palavra-chave ou recurso
- Conceito lógico de agrupamento
 - Modo particular de implementação

Composição

- Uma ImagemComCaption
 - Tem uma Imagem
e
 - Tem um Texto
- **Não**
 - É uma imagem e texto ao mesmo tempo

Composição

```
#ifndef INF112_MENSAGEMCAPTION_H
#define INF112_MENSAGEMCAPTION_H

#include "mensagemimg.h"
#include "mensagemtexto.h"

class MensagemCaption : public MensagemBase {
private:
    MensagemImagem _img;
    MensagemTexto _texto;
public:
    MensagemCaption(std::string avatar, MensagemImagem img,
                    MensagemTexto texto);
    virtual void exhibir() const override;
};

#endif
```

Composição

```
#ifndef INF112_MENSAGEMCAPTION_H
#define INF112_MENSAGEMCAPTION_H
```

```
#include "mensagemimg.h"
#include "mensagemtexto.h"
```

```
class MensagemCaption : public MensagemBase {
private:
    MensagemImagem _img;
    MensagemTexto _texto;
public:
    MensagemCaption(std::string avatar, MensagemImagem img,
                    MensagemTexto texto);
    virtual void exhibir() const override;
};
```

```
#endif
```

Ainda é uma mensagem

Composta de duas outras

Composição

Note que repassamos a responsabilidade

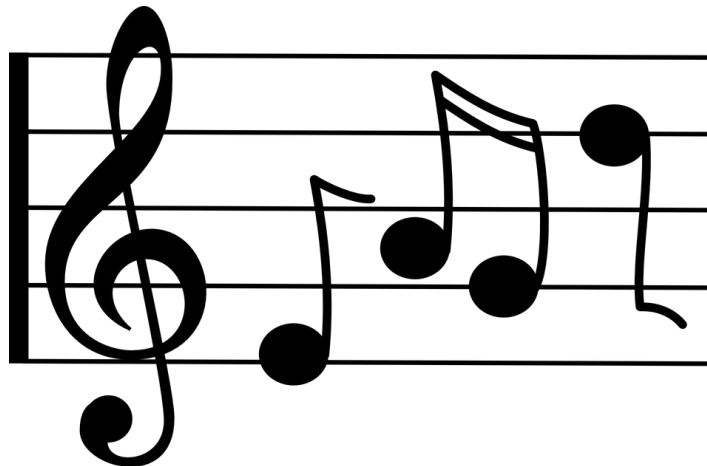
```
#include "mensagemcaption.h"
```

```
MensagemCaption::MensagemCaption(std::string avatar,  
                                  MensagemImagem img,  
                                  MensagemTexto texto):  
    MensagemBase(avatar), _img(img), _texto(texto) {}
```

```
void MensagemCaption::exibir() const {  
    this->_img.exibir();  
    this->_texto.exibir();  
}
```

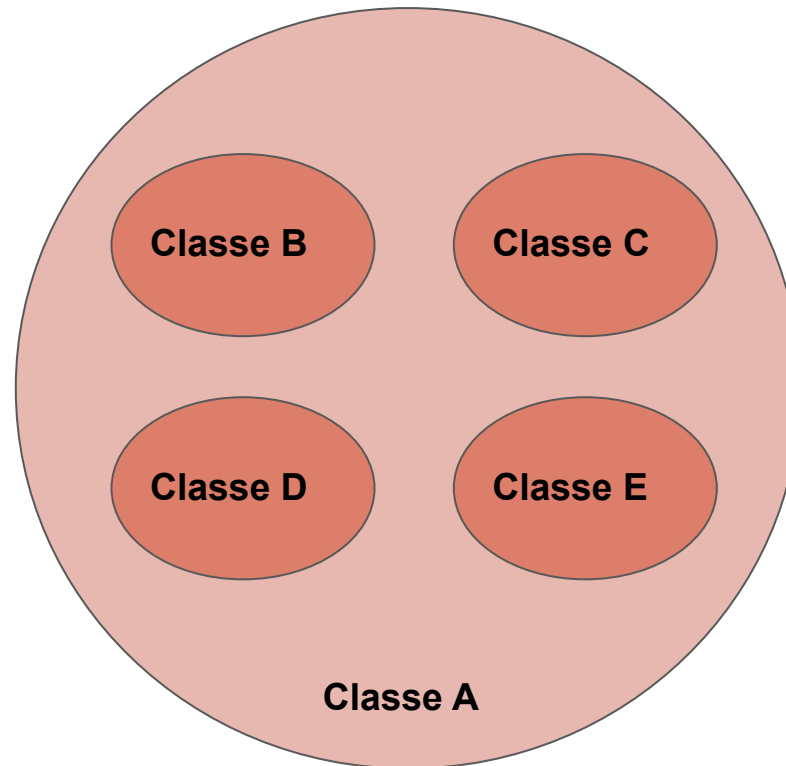

Composição

- Ao invés de copiar o comportamento
- Repassamos a responsabilidade
 - Boa prática!
- Cada mensagem faz uma única coisa
 - Compomos elas



Composição

Contexto de Objeto



Composição

- Ao invés de copiar o comportamento
- Repassamos a responsabilidade
 - Boa prática!
- Antes de usar herança pense:
 - (1) faz sentido a relação de **é** (is-a)?
 - (2) a composição fica mais complicado?
- Se qualquer um dos dois for **não**
 - Não use herança.

Exemplo: Uno

Jogo de Uno / 8 maluco / Mau Mau

- Cada Jogador tem recebe 7 cartas
- O resto do Baralho é oculto

Jogo de Uno / 8 maluco / Mau Mau

- Cada Jogador tem recebe 7 cartas
- O resto do Baralho é oculto
- Quais as classes até agora?

Jogo de Uno / 8 maluco / Mau Mau

- Cada **Jogador** tem recebe 7 **Cartas**
- O resto do **Baralho** é oculto

Jogo de Uno / 8 maluco / Mau Mau

- Cada **Jogador** tem recebe 7 **Cartas**
- O resto do **Baralho** é oculto
- Um baralho é composto de?

Jogo de Uno / 8 maluco / Mau Mau

- Cada **Jogador** tem recebe 7 **Cartas**
- O resto do **Baralho** é oculto
- Um baralho é composto de?
 - Coleção de **Cartas**

Jogo de Uno / 8 maluco / Mau Mau

- Cada **Jogador** tem recebe 7 **Cartas**
- O resto do **Baralho** é oculto
- Um baralho é composto de?
 - Coleção de **Cartas**
- Uno é um Jogo interessante.
 - Inicia no sentido horário, pode mudar

Jogo de Uno / 8 maluco / Mau Mau

- Cada **Jogador** tem recebe 7 **Cartas**
- O resto do **Baralho** é oculto
- Um baralho é composto de?
 - Coleção de **Cartas**
- Uno é um Jogo interessante.
 - Inicia no sentido horário, pode mudar
 - Isto é? Mantém um _____

Jogo de Uno / 8 maluco / Mau Mau

- Cada **Jogador** tem recebe 7 **Cartas**
- O resto do **Baralho** é oculto
- Um baralho é composto de?
 - Coleção de **Cartas**
- Uno é um Jogo interessante.
 - Inicia no sentido horário, pode mudar
 - Isto é? Mantém um **estado**

Jogo de Uno / 8 maluco / Mau Mau

- Cada **Jogador** tem recebe 7 **Cartas**
- O resto do **Baralho** é oculto
- Um baralho é composto de?
 - Coleção de **Cartas**
- Uno é um **Jogo** interessante.
 - Inicia no sentido horário, pode mudar
 - Isto é? Mantém um **estado**
 - Nova classe, atributo **sentido**

Jogo de Uno / 8 maluco / Mau Mau

- Ao modelar o mundo real:
 - **Definir objetos**
 - **Definir responsabilidades**
 - **Definir iterações**

Cartas

- Cada carta tem uma cor e um número
- Existem cartas especiais

Cartas

- Cada carta tem uma cor e um número
- Existem cartas especiais
 - Bom local para fazer uso de?

Cartas

- Cada carta tem uma cor e um número
- Existem cartas especiais
 - Bom local para fazer uso de?
 - **Polimorfismo**
- Cartas especiais podem:
 - Alterar o sentido do jogo
 - Pular jogadores
 - Ser jogada em qualquer momento
 - Aumentar número de cartas do adversário

Jogadores

- Tem uma pontuação
- 7 cartas iniciais.
- **Porém**
 - Pode aumentar, com uma carta especial de um adversário
- **Vector/Set**

User Stories

- Iniciar Jogo
- Realizar Jogada
- Fechar programa
 - Desistir
- Salvar jogo
 - Continuar no futuro