UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Programação Orientada a Objetos (Polimorfismo – 2/2)

Prof. Douglas G. Macharet
douglas.macharet@dcc.ufmg.br

DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Introdução

- Polimorfismo Estático
  - Tempo de compilação
  - Ligação Prematura (Early/Static binding)
  - Mesmo contexto → Sobrecarga
- Polimorfismo Dinâmico
  - Tempo de execução
  - Ligação Tardia (Late/Dynamic binding)
  - Hierarquia → Sobrescrita

Introdução

- Polimorfismo Estático
  - Geralmente considerado mais eficiente
- Polimorfismo Dinâmico
  - Apresenta uma flexibilidade maior
- Como combinar os dois tipos?
  - Será que pode haver algum problema?
  - Quais cuidados devem ser tomados?

Exemplo 1

```
class A {

  public:
    virtual void f() {
      cout << "A::f()" << endl;
    }

};

class B : public A {

  public:
    void f() override {
      cout << "B::f()" << endl;
    }

};
```

```
int main() {

  A *pa = new B();
  pa->f();

  return 0;
}
```

Saída:

```
B::f()
```


Exemplo 2

```
int main() {

  A a;
  B b;
  a = b;
  a.f();

  return 0;
}
```

Saída:

```
A::f()
```

Apenas a parte relativa a A de 'b' é copiada em 'a'.

Static vs. Dynamic Dispatch

- Method Dispatch
  - Como uma linguagem seleciona qual implementação de um método ou função usar
- Static Dispatch (compilação)
  - Há uma garantia de que há apenas uma única implementação do método em questão
- Dynamic Dispatch (execução)
  - Adiar a seleção da implementação apropriada até que o *runtime type* seja conhecido

https://en.wikipedia.org/wiki/Dynamic_dispatch

Static vs. Dynamic Dispatch

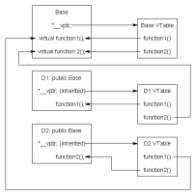
- Quando C++ usa Dynamic Dispatch?

Tipo	Valor ou Referência?	Chamada	Como f declarada em A?	Static ou Dynamic?
`A a;`	Valor	`a.f()`	Virtual	Static
`A a;`	Valor	`a.f()`	Não virtual	Static
`A* pa;`	Referência	`pa->f()`	Virtual	Dynamic
`A* pa;`	Referência	`pa->f()`	Não virtual	Static

Static vs. Dynamic Dispatch
Virtual Method Table (vtable)

- Sempre que uma classe define um método virtual, o compilador adiciona uma variável de membro oculto à classe que aponta para uma matriz de ponteiros para funções (virtuais) chamada de tabela de método virtual.

```
class Base {
  public:
    virtual void function1() {};
    virtual void function2() {};
};

class D1: public Base {
  public:
    void function1() override {};
};

class D2: public Base {
  public:
    void function2() override {};
};
```



https://en.wikipedia.org/wiki/Virtual_method_table
g++ -fdump-lang-class

Static vs. Dynamic Dispatch

```
class ClasseBase {

  public:
    ClasseBase() {
      cout << "BASE Constructor..." << endl;
    }
    ~ClasseBase() {
      cout << "BASE Destructor..." << endl;
    }

};

class ClasseDerivada : public ClasseBase {

  public:
    ClasseDerivada() {
      cout << "DERIVADA Constructor..." << endl;
    }
    ~ClasseDerivada() {
      cout << "DERIVADA Destructor..." << endl;
    }

};
```


Static vs. Dynamic Dispatch

```
int method() {
  ClasseDerivada d;
}

int main() {

  method();

  cout << "----------" << endl;

  ClasseBase *b = new ClasseDerivada();
  delete b;

  return 0;
}
```

https://wandbox.org/permlink/neHStO54v649efVs

Saída:

```
BASE Constructor...
DERIVADA Constructor...
DERIVADA Destructor...
BASE Destructor...
----------
BASE Constructor...
DERIVADA Constructor...
BASE Destructor...
```


Static vs. Dynamic Dispatch
Destrutores virtuais

- Destrutor da classe base deve ser virtual quando for usada de maneira polimórfica
  - Desalocar um objeto do tipo derivado por meio de um ponteiro para seu tipo base
  - Ocorre um *static dispatch* se não for virtual
- Tipo estático vs. Tipo dinâmico
  - Tipo da variável declarada (contrato/referência)
  - Tipo do objeto na memória (comportamento)

Effective C++: Item 7 – Pg. 40.

Static vs. Dynamic Dispatch
Destrutores virtuais

```
class ClasseBase {

  public:
    ClasseBase() {
      cout << "BASE Constructor..." << endl;
    }
    virtual ~ClasseBase() {
      cout << "BASE Destructor..." << endl;
    }

};

class ClasseDerivada : public ClasseBase {

  public:
    ClasseDerivada() {
      cout << "DERIVADA Constructor..." << endl;
    }
    ~ClasseDerivada() {
      cout << "DERIVADA Destructor..." << endl;
    }

};
```

Exemplo 3

```
class C {
  public:
    void m(A a) {
      cout << "C::m(A)" << endl;
    }

    void m(B b) {
      cout << "C::m(B)" << endl;
    }
};

int main() {

  A* ta = new A();
  A* tab = new B();


  C c;
  c.m(*ta);
  c.m(*tab);

  return 0;
}
```

https://wandbox.org/permlink/c7IBFZnfHcescxlB

Saída:

```
C::m(A)
C::m(A)
```

Por que será que isso acontece?!

Double Dispatch

- Problema
  - Sobrecarga → Static Dispatch
  - Sobrescrita (virtual) → Dynamic Dispatch
- Double Dispatch
  - Mecanismo que despacha uma chamada de função para diferentes funções concretas dependendo dos *runtime types* dos objetos
  - Receptor / Argumentos

https://en.wikipedia.org/wiki/Double_dispatch#Double_dispatch_in_C++

Double Dispatch
Exemplo

```
class SpaceShip {};
class ApolloSpacecraft : public SpaceShip {};

class Asteroid {
  public:
    virtual void CollideWith(SpaceShip&) {
      cout << "Asteroid hit a SpaceShip" << endl;
    }
    virtual void CollideWith(ApolloSpacecraft&) {
      cout << "Asteroid hit an ApolloSpacecraft" << endl;
    }
};

class ExplodingAsteroid : public Asteroid {
  public:
    virtual void CollideWith(SpaceShip&) {
      cout << "ExplodingAsteroid hit a SpaceShip" << endl;
    }
    virtual void CollideWith(ApolloSpacecraft&) {
      cout << "ExplodingAsteroid hit an ApolloSpacecraft" << endl;
    }
};
```


Double Dispatch
Exemplo

```
int main() {
  SpaceShip theSpaceShip;
  ApolloSpacecraft theApolloSpacecraft;

  Asteroid theAsteroid;
  theAsteroid.CollideWith(theSpaceShip);
  theAsteroid.CollideWith(theApolloSpacecraft);

  ExplodingAsteroid theExplodingAsteroid;
  theExplodingAsteroid.CollideWith(theSpaceShip);
  theExplodingAsteroid.CollideWith(theApolloSpacecraft);

  return 0;
}
```

```
Asteroid hit a SpaceShip
Asteroid hit an ApolloSpacecraft
ExplodingAsteroid hit a SpaceShip
ExplodingAsteroid hit an ApolloSpacecraft
```


Double Dispatch
Exemplo

```
int main() {
  SpaceShip theSpaceShip;
  ApolloSpacecraft theApolloSpacecraft;

  Asteroid *otherExplodingAsteroid = new ExplodingAsteroid();
  otherExplodingAsteroid->CollideWith(theSpaceShip);
  otherExplodingAsteroid->CollideWith(theApolloSpacecraft);

  return 0;
}
```

```
ExplodingAsteroid hit a SpaceShip
ExplodingAsteroid hit an ApolloSpacecraft
```


Double Dispatch
Exemplo

```
int main() {
  Asteroid theAsteroid;
  Asteroid *otherExplodingAsteroid = new ExplodingAsteroid();

  SpaceShip *otherApolloSpacecraft = new ApolloSpacecraft();
  theAsteroid.CollideWith(*otherApolloSpacecraft);
  otherExplodingAsteroid->CollideWith(*otherApolloSpacecraft);

  return 0;
}
```

```
Asteroid hit a SpaceShip
ExplodingAsteroid hit a SpaceShip
```

Double Dispatch
Exemplo

```
class SpaceShip {
  public:
    virtual void CollideWith(Asteroid& inAsteroid) {
      inAsteroid.CollideWith(*this);
    }
};

class ApolloSpacecraft : public SpaceShip {
  public:
    virtual void CollideWith(Asteroid& inAsteroid) {
      inAsteroid.CollideWith(*this);
    }
};
```


Double Dispatch
Exemplo

```
int main() {
  Asteroid theAsteroid;
  Asteroid *otherExplodingAsteroid = new ExplodingAsteroid();

  SpaceShip *otherApolloSpacecraft = new ApolloSpacecraft();
  otherApolloSpacecraft->CollideWith(theAsteroid);
  otherApolloSpacecraft->CollideWith(*otherExplodingAsteroid);

  return 0;
}
```

https://wandbox.org/permlink/dMYHrYDcNw6ZYM4V

```
Asteroid hit an ApolloSpacecraft
ExplodingAsteroid hit an ApolloSpacecraft
```

https://en.wikipedia.org/wiki/Visitor_pattern

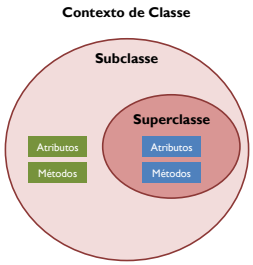
Conversão de tipo

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

Conversão de tipo
Upcasting

- Ocorre no sentido Subclasse → Superclasse
- Não há necessidade de indicação explícita
- A classe derivada sempre vai manter as características públicas da superclasse

Conversão de tipo
Upcasting



Conversão de tipo
Downcasting

- Ocorre no sentido Superclasse → Subclasse
- Não é feito de forma automática!
- Deve-se deixar explícito, informando o nome do subtipo antes do nome da variável
- Isso sempre será válido?
  - Não! Por que?
  - Subclasse → Características específicas

Conversão de tipo
Downcasting

- Nem sempre uma superclasse poderá assumir o tipo de uma subclasse
  - Exemplo: Todo Gato é Animal, mas nem todo Animal é Gato, pode ser Cachorro ou Pato
- Verificação: dynamic_cast
- Exceção: bad_cast

http://www.cplusplus.com/doc/tutorial/typecasting/
https://en.cppreference.com/w/cpp/language/dynamic_cast

Conversão de tipo
Downcasting – Exemplo 1

```
class Base { virtual dummy(){}; };
class Derived_A : public Base { };
class Derived_B : public Base { };

int main() {
  Base* b1 = new Derived_A();
  Base* b2 = new Base();

  if (Derived_A* d1 = dynamic_cast<Derived_A*>(b1))
    cout << "Essa chamada eh valida!" << endl;
  else
    cout << "Essa chamada NAO eh valida!" << endl;

  if (Derived_A* d2 = dynamic_cast<Derived_A*>(b2))
    cout << "Essa chamada eh valida!" << endl;
  else
    cout << "Essa chamada NAO eh valida!" << endl;

  if (Derived_B* d3 = dynamic_cast<Derived_B*>(b1))
    cout << "Essa chamada eh valida!" << endl;
  else
    cout << "Essa chamada NAO eh valida!" << endl;

  return 0;
}
```

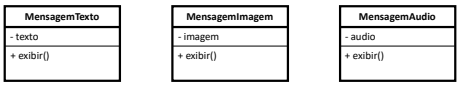
Conversão de tipo
Downcasting – Exemplo 2

```
#include <typeinfo>

class Base { virtual dummy(){}; };
class Derived_A : public Base { };
class Derived_B : public Base { };

int main() {
  Base* b1 = new Derived_A();
  Base* b2 = new Base();

  try {
    Derived_A& d1 = dynamic_cast<Derived_A&>(*b1);
    cout << "Conversao OK" << endl;

    Derived_A& d2 = dynamic_cast<Derived_A&>(*b2);
  } catch (bad_cast& e) {
    cout << e.what() << endl;
  }

  return 0;
}
```

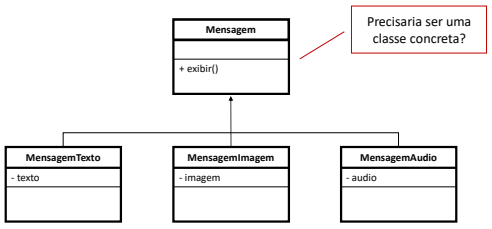
Exercício

- Qual o domínio do problema? Elementos?
  - Quais atributos devem existir?
  - Quais métodos devem existir?

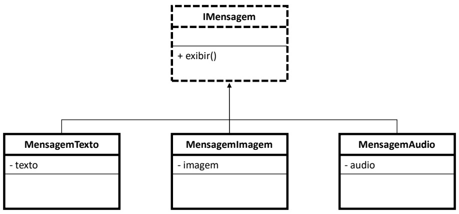
Exercício



Exercício



Exercício



Exercício

IMensagem.hpp

```cpp
#ifndef IMENSAGEM_H
#define IMENSAGEM_H

class IMensagem {

  public:
    virtual void exibir() = 0;

};

#endif
```


Exercício

MensagemTexto.hpp

```cpp
#ifndef MENSAGEMTEXTO_H
#define MENSAGEMTEXTO_H

#include <string>
#include "IMensagem.hpp"

class MensagemTexto : public IMensagem {

  private:
    std::string _msg;

  public:
    MensagemTexto(std::string msg);
    void exibir();

};

#endif
```

MensagemImagem.hpp

```cpp
#ifndef MENSAGEMIMAGEM_H
#define MENSAGEMIMAGEM_H

#include <string>
#include "IMensagem.hpp"

class MensagemImagem : public IMensagem {

  private:
    std::string _imagem;

  public:
    MensagemImagem(std::string imagem);
    void exibir();

};

#endif
```

MensagemAudio.hpp

```cpp
#ifndef MENSAGEMAUDIO_H
#define MENSAGEMAUDIO_H

#include <string>
#include "IMensagem.hpp"

class MensagemAudio : public IMensagem {

  private:
    std::string _audio;

  public:
    MensagemAudio(std::string audio);
    void exibir();

};

#endif
```


Exercício

MensagemTexto.cpp

```cpp
#include <iostream>
#include "MensagemTexto.hpp"

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

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

MensagemAudio.cpp

```cpp
#include <iostream>
#include "MensagemAudio.hpp"

MensagemAudio::MensagemAudio(std::string audio) {
  this->_audio = audio;
}

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


Exercício

MensagemImagem.cpp

```cpp
#include <iostream>
#include <fstream>
#include "MensagemImagem.hpp"

MensagemImagem::MensagemImagem(std::string imagem) {
  this->_imagem = imagem;
}

void MensagemImagem::exibir() {
  std::ifstream arquivo("./imgfiles/" + this->_imagem, std::fstream::in);
  if (!arquivo.is_open()) {
    exit(1);
  }

  std::string line;
  while (std::getline(arquivo, line))
    std::cout << line << std::endl;

  arquivo.close();
}
```


Exercício

main.cpp

```cpp
#include "IMensagem.hpp"

#include "MensagemTexto.hpp"
#include "MensagemImagem.hpp"
#include "MensagemAudio.hpp"



int main() {

  MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
  MensagemAudio audio("audio.wav");
  MensagemImagem imagem("imagem03.ascii");
  MensagemTexto texto2("Mas que puxa :(");



  return 0;
}
```

```
Oi, tem aula de PDS2 hoje?
Tocando o arquivo... audio.wav
       .-'&     '-.
      /             \
     :   o     o     ;
    (       (_        )
     :               ;
      \      __     /
       `-._______.-'
          /`"""`\
         /    ,   \
        /|/\/\/\ _\
       (_|/\/\/\\__)
         |_________|
         __)_ |_ (__
        (_____|_____)
Mas que puxa :(
```

Exercício

```
. project
├── Makefile
├── build
├── include
│   └── mensagem
│       └── IMensagem.hpp
│       └── MensagemAudio.hpp
│       └── MensagemImagem.hpp
│       └── MensagemTexto.hpp
└── imgfiles
└── src
│   └── main.cpp
│   └── mensagem
│       └── MensagemAudio.cpp
│       └── MensagemImagem.cpp
│       └── MensagemTexto.cpp
```


Exercício

```
CC=g++
CFLAGS=-std=c++11 -Wall
TARGET=program

BUILD_DIR   = ./build
SRC_DIR     = ./src
INCLUDE_DIR = ./include

$(BUILD_DIR)/$(TARGET): $(BUILD_DIR)/MensagemTexto.o $(BUILD_DIR)/MensagemImagem.o $(BUILD_DIR)/MensagemAudio.o $(BUILD_DIR)/main.o
        $(CC) $(CFLAGS) -o $(BUILD_DIR)/$(TARGET) $(BUILD_DIR)/*.o

$(BUILD_DIR)/MensagemTexto.o: $(INCLUDE_DIR)/mensagem/IMensagem.hpp $(INCLUDE_DIR)/mensagem/MensagemTexto.hpp $(SRC_DIR)/mensagem/MensagemTexto.cpp
        $(CC) $(CFLAGS) -I $(INCLUDE_DIR)/mensagem/ -c $(SRC_DIR)/mensagem/MensagemTexto.cpp -o $(BUILD_DIR)/MensagemTexto.o

$(BUILD_DIR)/MensagemImagem.o: $(INCLUDE_DIR)/mensagem/IMensagem.hpp $(INCLUDE_DIR)/mensagem/MensagemImagem.hpp $(SRC_DIR)/mensagem/MensagemImagem.cpp
        $(CC) $(CFLAGS) -I $(INCLUDE_DIR)/mensagem/ -c $(SRC_DIR)/mensagem/MensagemImagem.cpp -o $(BUILD_DIR)/MensagemImagem.o

$(BUILD_DIR)/MensagemAudio.o: $(INCLUDE_DIR)/mensagem/IMensagem.hpp $(INCLUDE_DIR)/mensagem/MensagemAudio.hpp $(SRC_DIR)/mensagem/MensagemAudio.cpp
        $(CC) $(CFLAGS) -I $(INCLUDE_DIR)/mensagem/ -c $(SRC_DIR)/mensagem/MensagemAudio.cpp -o $(BUILD_DIR)/MensagemAudio.o

$(BUILD_DIR)/main.o: $(INCLUDE_DIR)/mensagem/IMensagem.hpp $(INCLUDE_DIR)/mensagem/MensagemTexto.hpp $(INCLUDE_DIR)/mensagem/MensagemImagem.hpp
$(INCLUDE_DIR)/mensagem/MensagemAudio.hpp $(SRC_DIR)/main.cpp
        $(CC) $(CFLAGS) -I $(INCLUDE_DIR)/mensagem/ -c $(SRC_DIR)/main.cpp -o $(BUILD_DIR)/main.o


# Rule for cleaning files generated during compilation.
# Call "make clean" to use it
clean:
        rm -f $(BUILD_DIR)/*
```


Exercício

- Tarefas
  - Criar uma classe Chat com o método exibir()
    - Deve estar em um pacote diferente
    - Modificar o Makefile de acordo
  - E se eu quisesse agora exibir o horário?
  - E se eu quisesse guardar um histórico?