

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

DCC UFMG

PDS 2 - Programação Orientada a Objetos (Polimorfismo – 2/2) 2

## 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?

DCC UFMG

PDS 2 - Programação Orientada a Objetos (Polimorfismo – 2/2) 3

## 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()

DCC UFMG

PDS 2 - Programação Orientada a Objetos (Polimorfismo – 2/2) 4

## 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'.

DCC UFMG

PDS 2 - Programação Orientada a Objetos (Polimorfismo – 2/2) 5

## 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](https://en.wikipedia.org/wiki/Dynamic_dispatch)

DCC UFMG

PDS 2 - Programação Orientada a Objetos (Polimorfismo – 2/2) 6

## Static vs. Dynamic Dispatch

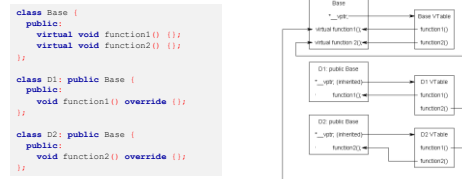
### ■ Quando C++ usa Dynamic Dispatch?

| Tipo   | Valor ou Referência? | Chamada | Como é 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.



[https://en.wikipedia.org/wiki/Virtual\\_method\\_table](https://en.wikipedia.org/wiki/Virtual_method_table)  
g++ -dump-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;
}

```

### Saída:

```

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

```

<https://wandbox.org/permlink/nehSI064v649eV/s>

## 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)

## 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/c78FZz#tcesoB>

Saída:  
C::m(A)  
C::m(A)

Por que será que  
isso acontece?!

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

13

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

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

14

## 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;
    }
};
```

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

15

## 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

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

16

## 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

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

17

## 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

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

18

## 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/dMYHYDcnW6ZYM4V>

Asteroid hit an ApolloSpacecraft  
ExplodingAsteroid hit an ApolloSpacecraft

[https://en.wikipedia.org/wiki/Visitor\\_pattern](https://en.wikipedia.org/wiki/Visitor_pattern)

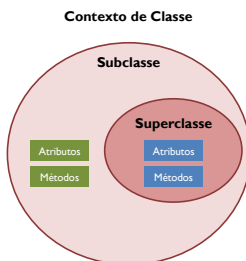
## Conversão de tipo

- 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.cppplus.com/doc/tutorial/typecasting/>  
[https://en.cppreference.com/w/cpp/language/dynamic\\_cast](https://en.cppreference.com/w/cpp/language/dynamic_cast)



PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

25

## 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;
}
```



PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

26

## 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;
}
```



PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

27

## Exercício



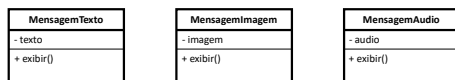
- Qual o domínio do problema? Elementos?
  - Quais atributos devem existir?
  - Quais métodos devem existir?



PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

28

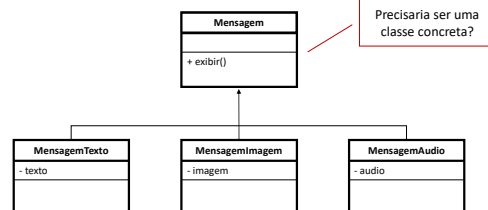
## Exercício



PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

29

## Exercício



PDS 2 - Programação Orientada a Objetos (Polimorfismo - 2/2)

30



