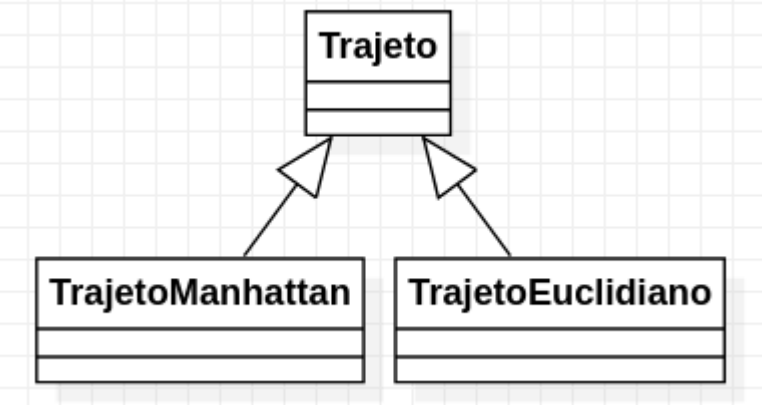
Funções Puramente Virtuais e Classes Abstratas

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Funções puramente virtuais

- Muitas vezes declaramos classes que não devem ser instanciadas
  - Classes incompletas
    - Servem como base para outras classes, mas não deveriam ser usadas para gerar objetos
    - Exemplo: a classe trajeto do projeto disponibilizado
      - Serve como base para classes como TrajetoEuclidiano e TrajetoManhattan
        - Mas tentar calcular a distância de um trajeto com essa classe é um erro



Funções puramente virtuais

- No momento podemos instanciar *Trajeto*
  - A única forma de indicarmos que não podemos calcular uma distância a partir dessa classe é pelo retorno da função *calcularDistanciaPontos*
    - Retorna zero
    - Mas isso não é muito informativo, nem útil

Trajeto.hpp

```
//...
class Trajeto{
	//...
	protected:
		virtual double calcularDistanciaPontos(const Ponto* const p1, const Ponto* const p2) const;
	//...
};
```

Trajeto.cpp

```
//...
double Trajeto::calcularDistanciaPontos(const Ponto* const p1, const Ponto* const p2) const{
	return 0;
}
```

Funções puramente virtuais

- Podemos declarar uma função **puramente virtual** em uma classe base
  - **Não damos implementação alguma para a função**
- Uma classe que possui **uma ou mais** funções puramente virtuais é chamada de **classe abstrata**
  - Não podemos criar instâncias de classes abstratas
    - Classes abstratas são consideradas **classes incompletas**
      - Exigem que outras classes herdem delas para que as implementações necessárias sejam concluídas

Funções puramente virtuais

- Para declarar uma **função** como **puramente virtual**, basta adicionar *= 0* no final da sua declaração no .hpp
  - E não a implementar no .cpp
  - Isso inicializa o ponteiro de função interno para NULL.
    - Veremos adiante

Trajeto.hpp

```
//...
class Trajeto{
        //...
        protected:
                virtual double calcularDistanciaPontos(const Ponto* const p1, const Ponto* const p2) const = 0;
        //...
};
```

Teste você mesmo

- No main
  - O que está errado?
  - Teste você mesmo.

```
#include <iostream>

#include "Trajeto.hpp"
#include "TrajetoManhattan.hpp"
#include "TrajetoEuclidiano.hpp"

int main(){
	Trajeto* tm{new TrajetoManhattan};
	Trajeto* te{new TrajetoEuclidiano};
	Trajeto* t{new Trajeto};

	//..

	delete tm;
	delete te;
	delete t;

	return 0;
}
```

Teste você mesmo

```
#include <iostream>

#include "Trajeto.hpp"
#include "TrajetoManhattan.hpp"
#include "TrajetoEuclidiano.hpp"

int main(){
    Trajeto* tm{new TrajetoManhattan};
    Trajeto* te{new TrajetoEuclidiano};
    Trajeto* t{new Trajeto};

    //..

    delete tm;
    delete te;
    delete t;

    return 0;
}
```

Erro de compilação. Impossível instanciar uma classe abstrata.

Funções puramente virtuais

- Ao declarar uma função puramente virtual
  - Você força as classes que herdam da classe base a implementarem a função
    - A sobrescrita **não é opcional** como em uma função virtual “comum”
- As classes que herdam da classe abstrata são chamadas de **classes concretas**

Funções e Classes “Finais”

- A partir do C++11
  - Uma **função** pode ser declarada como *final* em seu protótipo
    - Indica que a função não pode ser sobrescrita nas classes derivadas
    - Exemplo:

      virtual someFunction(parameters) ***final****;*
  - Uma **classe** pode ser declarada como *final*
    - Indica que a classe não pode ser derivada
    - Exemplo:

      class MyClass ***final*** *{*

      //...

      };
  - Tentar herdar de uma classe final, ou sobrescrever uma função final, resulta em um **erro em tempo de compilação**

O custo de uma função virtual

- Funções virtuais são implementadas internamente via três níveis de indireção (ponteiros)
  - Os níveis são ocultos e implementados pelo compilador
  - Mas você precisa conhecê-los para saber dos **custos computacionais** envolvidos

Primeiro nível de ponteiros

- Para toda classe que possui **ao menos uma** função virtual
  - O compilador monta na memória uma tabela chamada **virtual function table** – *vtable*
    - Contém os endereços na memória das funções virtuais implementadas nessa classe
    - Esse pode ser considerado o primeiro nível de ponteiros (indireção)

Segundo nível de ponteiros

- Quando um **objeto** de uma classe que possui **ao menos uma** função virtual **é instanciado**
  - O compilador adiciona internamente a esse objeto um **ponteiro** que aponta para a **vtable** correta
  - Esse é considerado o segundo nível de ponteiros (indireção)

Terceiro nível de ponteiros

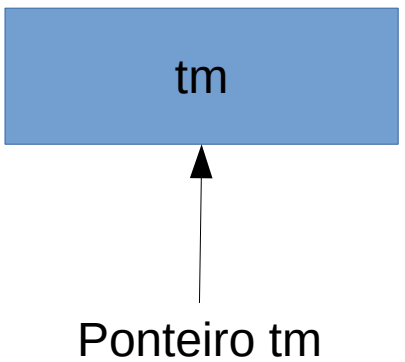
- O terceiro nível é o próprio *handle* do objeto (ponteiro ou referência)
  - Exemplo: Trajeto* tm{new TrajetoManhattan};

Exemplo de execução

```
//...
int main(){
        Trajeto* tm{new TrajetoManhattan};

        std::cout << tm->getDistanciaPercorrida() << std::endl;

        delete tm;
        return 0;
}
```



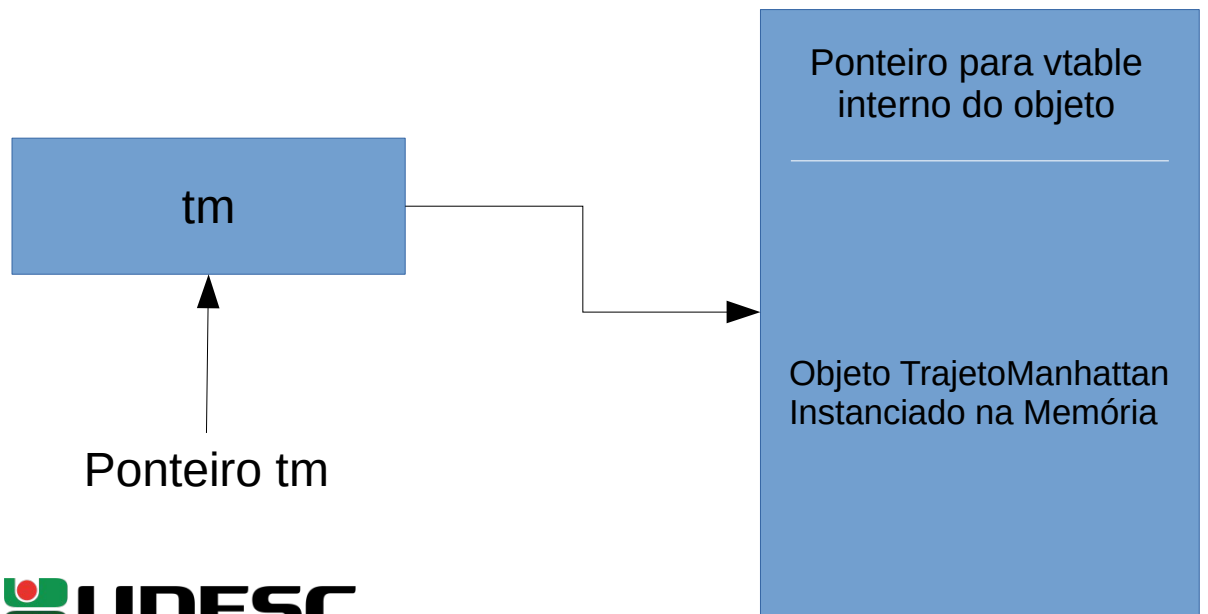
UDESC

Exemplo de execução

```
//...
int main(){
        Trajeto* tm{new TrajetoManhattan};

        std::cout << tm->getDistanciaPercorrida() << std::endl;

        delete tm;
        return 0;
}
```



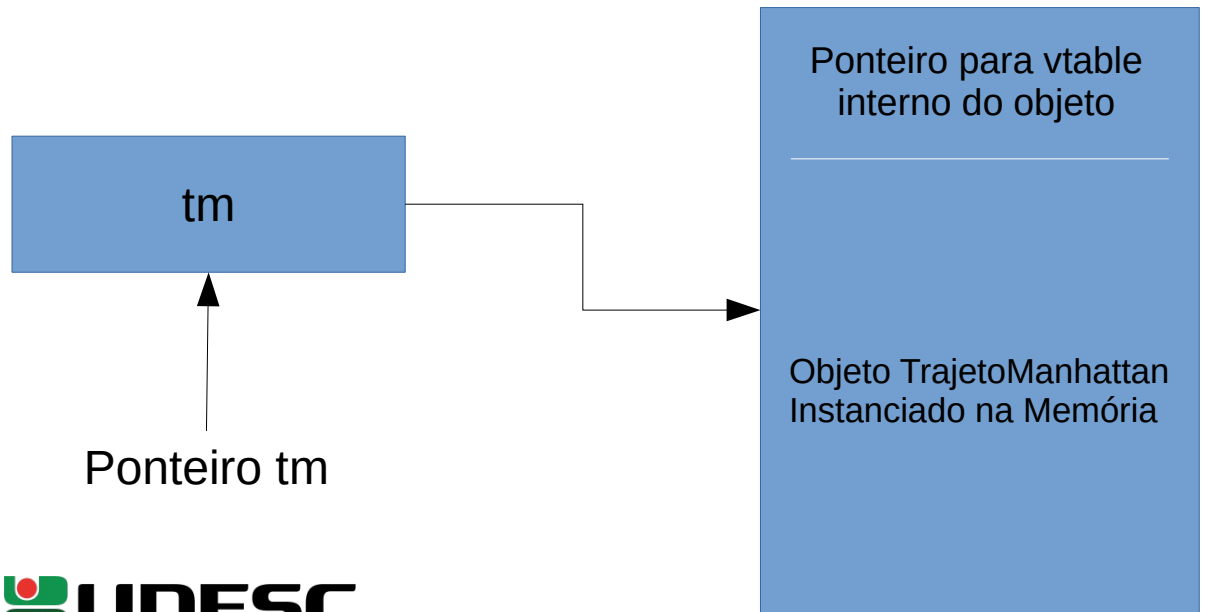
UDESC

Exemplo de execução

```
//...
int main(){
        Trajeto* tm{new TrajetoManhattan};

        std::cout << tm->getDistanciaPercorrida() << std::endl;

        delete tm;
        return 0;
}
```



Até aqui o custo é o mesmo para uma função virtual ou “comum”

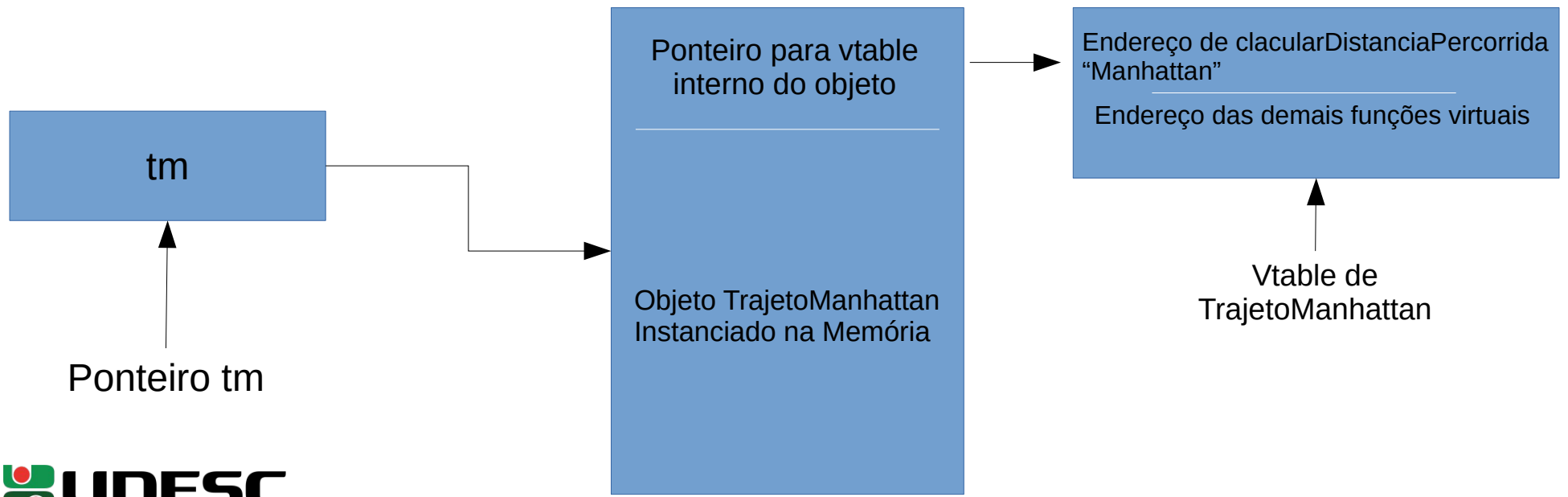
UDESC

Exemplo de execução

```
//...
int main(){
        Trajeto* tm{new TrajetoManhattan};

        std::cout << tm->getDistanciaPercorrida() << std::endl;

        delete tm;
        return 0;
}
```



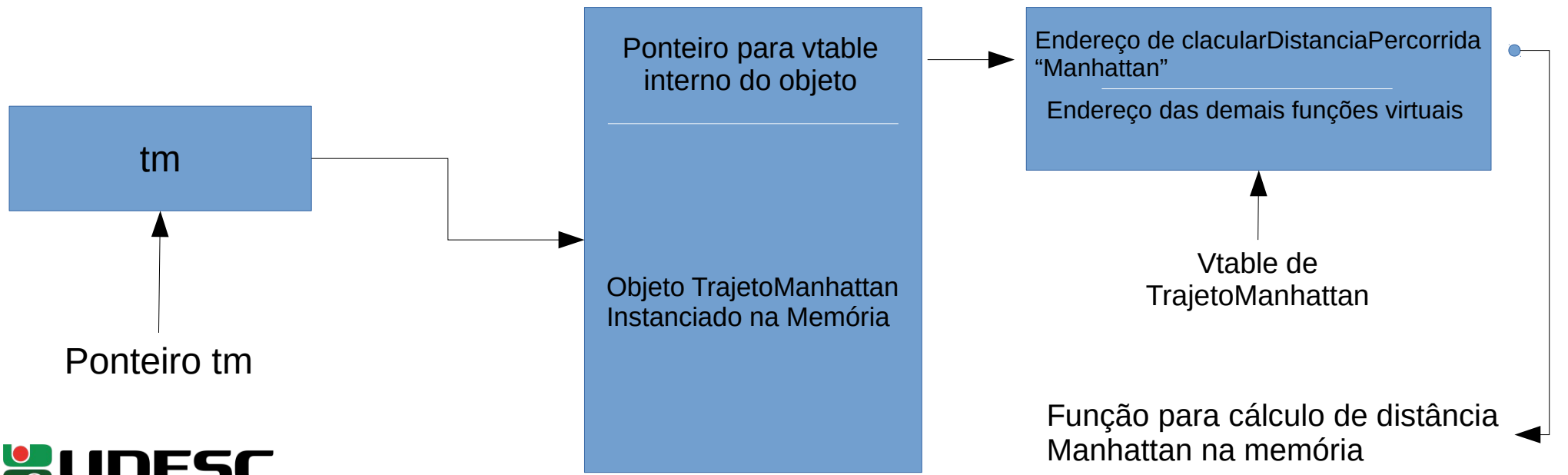
UDESC

Exemplo de execução

```
//...
int main(){
        Trajeto* tm{new TrajetoManhattan};

        std::cout << tm->getDistanciaPercorrida() << std::endl;

        delete tm;
        return 0;
}
```



UDESC

Curiosidades

- Muitas classes da STL, como *array* e *vector*, são implementadas sem usar funções virtuais
  - Não desejam pagar o *overhead* do polimorfismo
  - Geram problemas caso você precise herdar dessas classes para modificar algum comportamento

Dicas de Engenharia de Software e Performance

- Do ponto de vista de engenharia de software, idealmente todas as funções deveriam ser virtuais
  - Leia sobre o princípio Open Closed
- Quanto ao custo computacional
  - O seu compilador faz o possível para tentar resolver tudo em tempo de compilação e evitar as indireções múltiplas
  - Mesmo com as indireções, o custo é baixo
    - O seu processador possui mecanismos internos para tratar as indireções extras
      - Caches grandes (especialmente cache de instruções)
      - Mecanismos de predição de salto
        - Veja mais em Arquitetura de Computadores
  - **Remova as declarações virtuais em último caso**
    - Geralmente você pode otimizar o desempenho de diversas outras formas e com resultados melhores
    - Remova as chamadas virtuais apenas quando você não tiver mais opções

Exercício

1. Pesquise sobre o princípio Open Closed.

2. Modifique a hierarquia de classes solicitada na última aula
   - A classe Forma deve ser abstrata
   - Declare as funções necessárias da classe Forma como puramente virtuais

Referências

- DEITEL, P.; DEITEL, H. **C++ how to Program**. [S.l.]: Pearson, 2017. ISBN 9780134448237
- STROUSTRUP, B. **The C++ Programming Language**. Pearson Education, 2013. ISBN 9780133522853.