

Conceito de herança

Polígono

Retângulo

```
class Poligono {  
    private:  
        int nroVertices;  
        float *xCord, *yCord;  
    public:  
        void set(float *x, float *y,  
int nV);  
};
```

```
class Retangulo {  
    private:  
        int nroVertices;  
        float *xCord, *yCord;  
    public:  
        void set(float *x, float *y,  
int nV);  
        float area();  
};
```

Conceito de herança

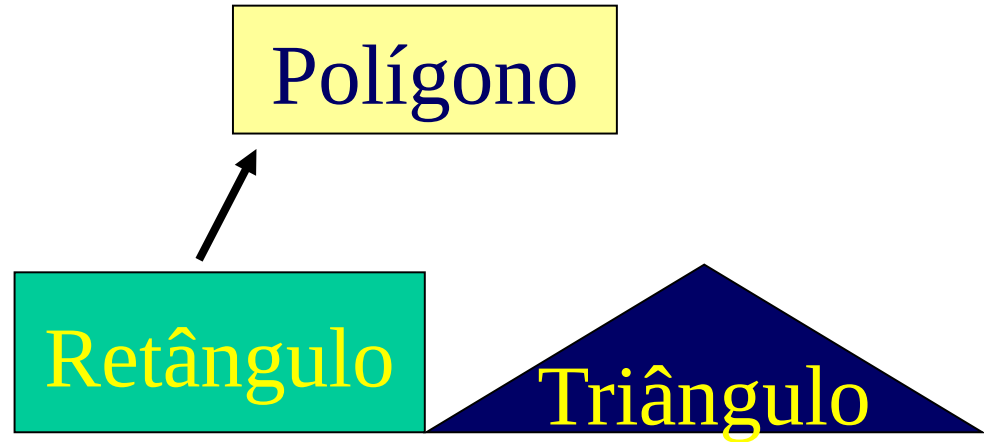


```
class Triangulo {  
    private:  
        int nroVertices;  
        float *xCord, *yCord;  
    public:  
        void set(float *x, float *y, int nV);  
        float area();  
};
```

Conceito de herança

```
class Poligono {  
    protected:  
        int nroVertices;  
        float * xCord, float  
        *yCord;  
    public:  
        void set(float *x, float *y,  
        int nV);  
};
```

```
class Retangulo: public  
    Poligono{  
    public:  
        float area();  
};
```

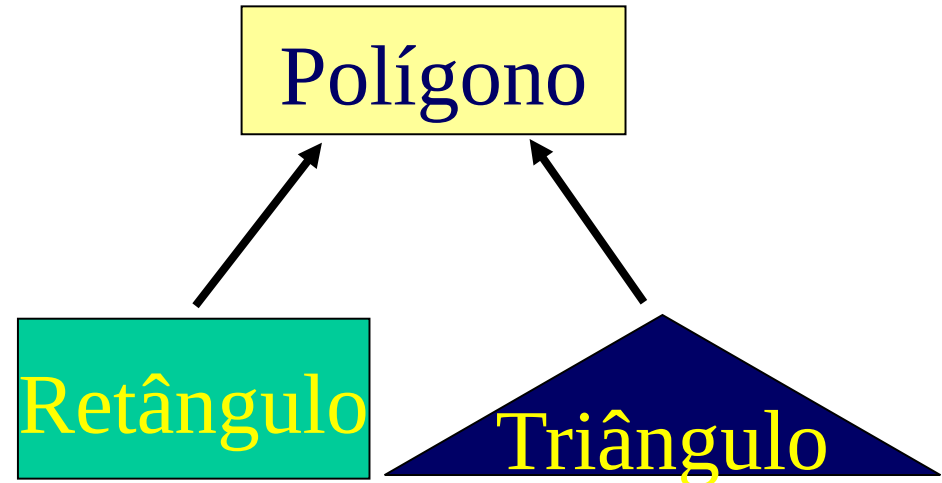


```
class Retangulo {  
    protected:  
        int nroVertices;  
        float * xCord, float *yCord;  
    public:  
        void set(float *x, float *y,  
        int nV);  
        float area ();  
};
```

Conceito de herança

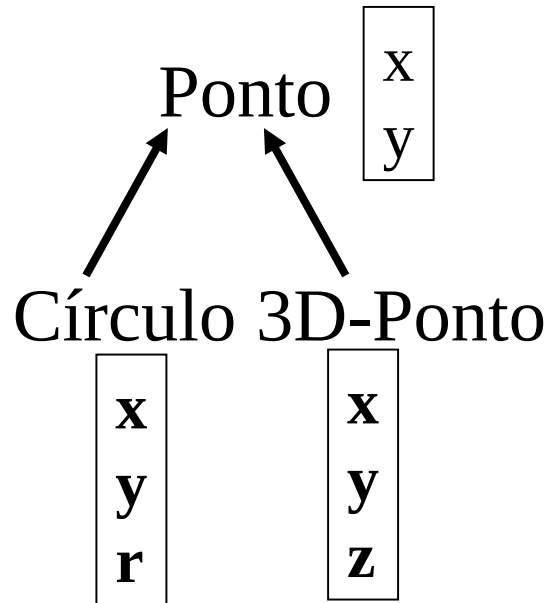
```
class Poligono {  
    protected:  
        int nroVertices;  
        float *xCord, float  
        *yCord;  
    public:  
        void set(float *x, float  
        *y, int nV);  
};
```

```
class Triangulo : public  
    Poligono{  
    public:  
        float area();  
};
```



```
class Triangulo {  
    protected:  
        int nroVertices;  
        float *xCord, float *yCord;  
    public:  
        void set(float *x, float *y, int  
        nV);  
        float area();  
};
```

Conceito de herança



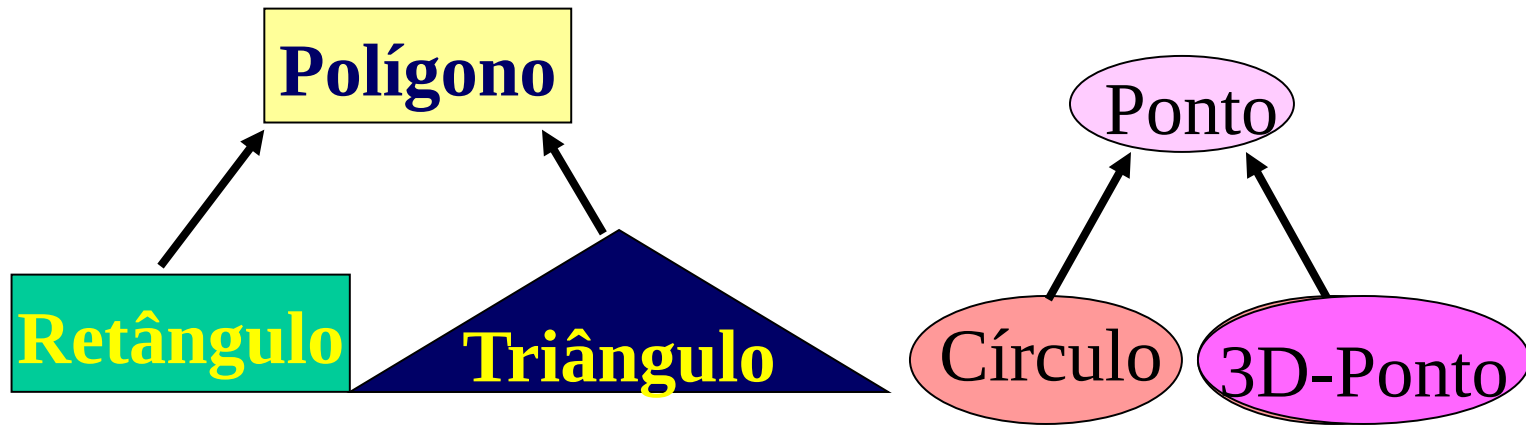
```
class Circulo: public Ponto{  
    private:  
        double r;  
};
```

```
class Ponto {  
    protected:  
        int x, y;  
    public:  
        void set(int a, int b);  
};
```

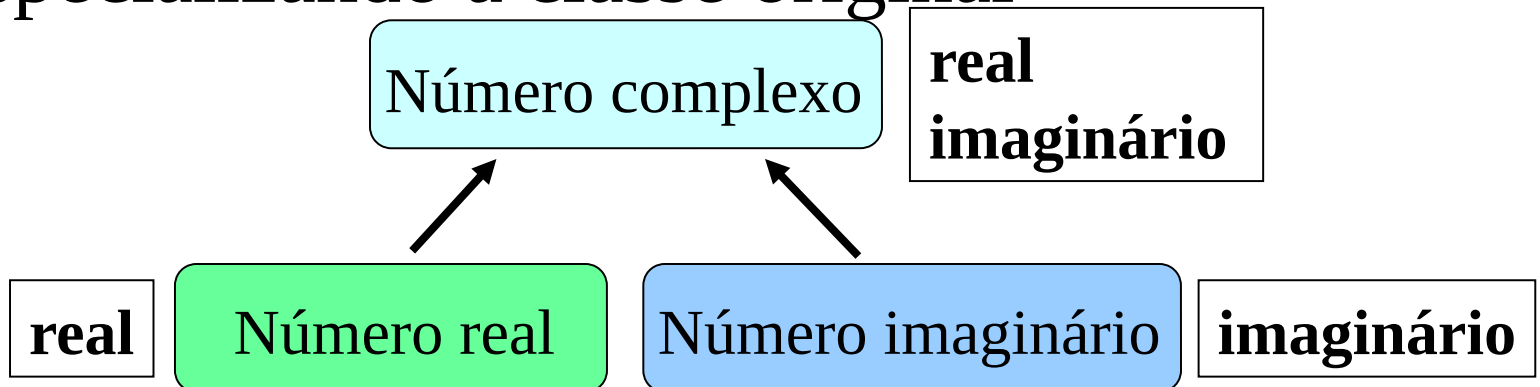
```
class 3D-Ponto: public Ponto{  
    private:  
        int z;  
};
```

Conceito de herança

- Aumentando a classe original



- Especializando a classe original



Porque herança?

A herança é um mecanismo para

- construção de classes de tipos de classes já existentes
- definição de novos tipos de classe para ser uma
 - especialização
 - aumentode tipos já existentes

Definir uma hierarquia de classes

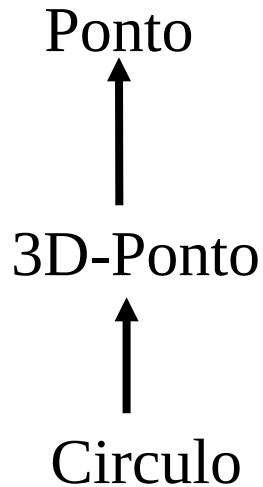
- Sintaxe:

```
class NomeClasseDerivada : nível de acesso NomeClasseBase
```

Onde

- nível de acesso especifica o tipo de derivação
 - por padrão são privados, ou
 - publicos
- Qualquer classe pode servir como uma classe base
 - Assim, uma classe derivada também pode ser uma classe base.

Classe Derivada



```
class Ponto {  
    protected:  
        int x, y;  
    public:  
        void set(int a, int b);  
};
```

```
class 3D-Ponto: public Ponto{  
    private:  
        double z;  
    ... ..  
};
```

```
class Circulo: public 3D-Ponto {  
    private:  
        double r;  
    ... ..  
};
```

Ponto é a classe base de **3D-Ponto**, enquanto **3D-Ponto** é a classe base de **Circulo**

O que herda?

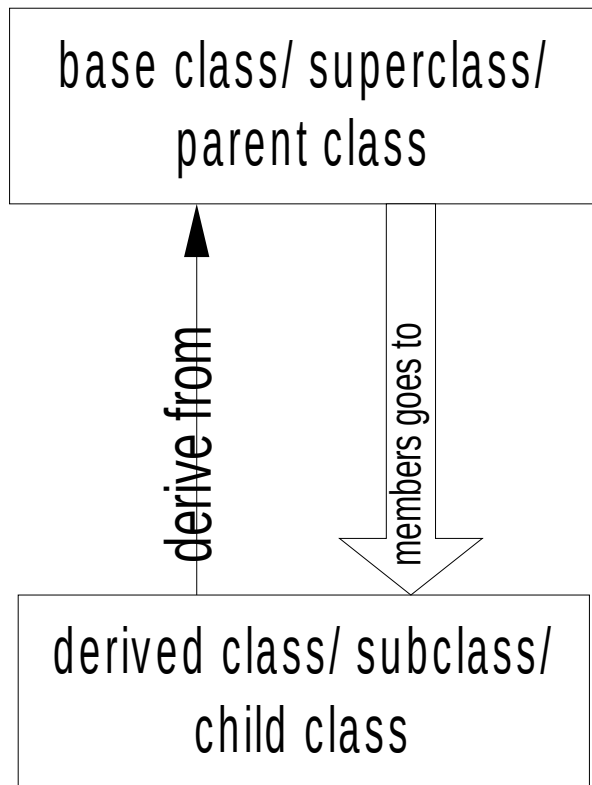
- Em princípio, cada membro de uma classe base é herdado por uma classe derivada
 - apenas com permissões de acesso diferentes

Controle de acesso sobre os membros

Há dois níveis de controle de acesso para os membros da classe

– definição de classe

– tipo de herança



```
class Ponto {  
    protected: int x, y;  
    public: void set(int a, int b);  
};  
class Circulo: public Ponto{  
    ... ..  
};
```

Direitos de Acesso de classes derivadas


Tipo de herança				
Controle de acesso para os deputados		private	protected	public
	private	-	-	-
	protected	private	protected	protected
	public	private	protected	public

- O tipo de herança define o nível de acesso para os membros da classe derivada que são herdados a partir da classe base

Classe Derivada

```
class mae {  
    protected: int mProt;  
    public: int mPubl;  
    private: int mPriv;  
};
```

private/protected/ public



```
class filha : ----- mae{  
    private: double dPriv;  
    pública: void dFoo();  
};
```

```
class neta : public filha {  
    private: double nPriv;  
    public: void nFoo();  
};
```

```
int main () {  
    /*....*/  
}
```

```
void filha :: dFoo() {  
    mPriv = 10; //erro  
    mProt = 20;  
};
```

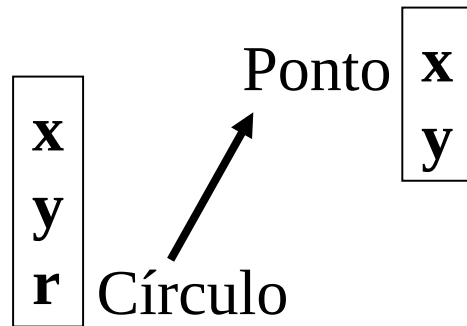
O que herda?

- Em princípio, cada membro de uma classe base é herdado por uma classe derivada
 - Apenas com permissões de acesso diferentes
- Contudo, há exceções para
 - construtor e destruidor
 - operador = () membro
 - amigos

Uma vez que todas estas funções são específicas de cada classe

Defina seus próprios membros

A classe derivada também pode definir seus próprios membros, além dos membros herdados da classe base



```
class Circulo: public Ponto{  
    private:  
        double r;  
    public:  
        void set_r(double c);  
};
```

```
class Ponto {  
    protected: int x, y;  
    public:  
        void set(int a, int b);  
};
```

```
class Circulo {  
    protected: int x, y;  
    private: double r;  
    public:  
        void set(int a, int b);  
        void set_r(double c);  
};
```

Ainda mais ...

- Uma classe derivada pode **sobrepôr** os métodos definidos em sua classe pai. Com sobrescrita,
 - o método na subclasse tem a assinatura idêntico ao método na classe base.
 - uma subclasse implementa sua própria versão de um método de classe base.

```
class A {  
    protected:  
        int x, y;  
    public:  
        void imprime()  
        {cout << "De A" << endl;}  
};
```

```
class B: public A{  
    public:  
        void imprime()  
        {cout << "De B" << endl;}  
};
```


Acessar um método

```
class Ponto {  
    protected:  
    int x, y;  
    public:  
    void set(int a, int b)  
        {x = a; y = b};  
    void foo ();  
    void imprime();  
};
```

```
Ponto A;  
A.set(30,50); // da classe base Ponto  
A.imprime(); // da classe base Ponto
```

```
class Circulo: public Ponto{  
    private: double r;  
    public:  
    void set (int a, int b, double c) {  
        Ponto :: set (a, b); // chamada de  
        função mesmo nome  
        r = c;  
    }  
    void imprime(); };
```

```
Circulo C;  
C.set(10,10,100); // da classe Circulo  
C.foo(); // do classe base Ponto  
C.imprime(); // da classe Circulo
```

Regras de construtor para classes derivadas

O construtor padrão e o destrutor da classe base são sempre chamados quando um novo objeto de uma classe derivada é criado ou destruído.

```
class A {  
    public:  
    A ()  
        {cout << "A:padrão" << endl;}  
    A (int a)  
        {cout << "A:parâmetro" <<  
        endl;}  
};
```

```
class B: public A{  
    public:  
    B (int a)  
        {cout << "B" << endl;}  
};
```

```
B teste(1);
```

saída:

```
A:padrão  
B
```

Regras de construtor para classes derivadas

Você também pode especificar um construtor da classe base que não seja o construtor padrão

```
ConstrDerivada (args derivada): ConstrBase (args base)  
{ ClasseDerivada corpo do construtor }
```

Regras de construtor para classes derivadas

```
class A {  
    public:  
    A ()  
        {cout << "A:padrao" << endl;}  
    A (int a)  
        {cout << "A:parâmetro" << endl;}  
};
```

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

C teste(1);

saída:

A:parâmetro
C

Exercício

Crie um programa que permita gerenciar pessoas físicas e pessoas jurídicas. Toda pessoa física possui nome, CPF e endereço, e toda pessoa jurídica possui nome, CNPJ e endereço. Crie a classe principal do programa para instanciar uma pessoa física com base em valores predefinidos, e imprima seus dados. Faça a reutilização de código usando herança

Exercício 2

Escreva um conjunto de classes a serem usadas em uma ferramenta matemática. Essa ferramenta deve gerenciar figuras geométricas. Considere apenas quadrados, triângulos e losangos. Crie classes que representem as figuras geométricas, considerando-se os atributos necessários para cada tipo de figura. Além disso, crie um método responsável em calcular e retornar a área para cada tipo de figura.

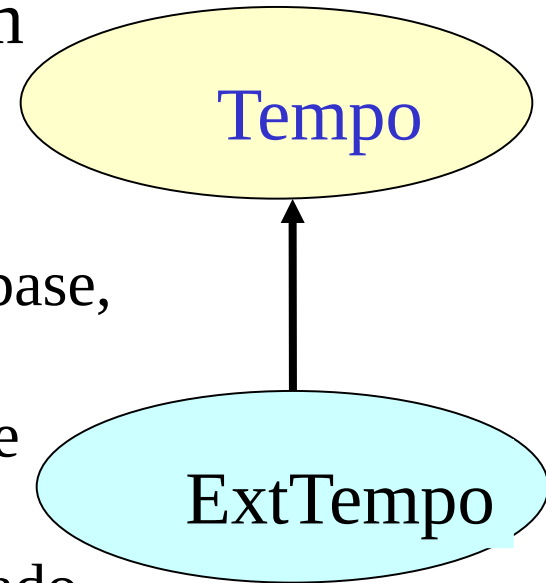
Observação: a área do quadrado é dada por: $\text{lado} * \text{lado}$; a área do triângulo é dada por: $\text{base} * \text{altura} / 2$; e a área do losango é dada por: $D * d / 2$, onde D é a diagonal maior e d é a diagonal menor

Exercício 3

Escreva um conjunto de classes a serem usadas em uma ferramenta matemática. Essa ferramenta deve gerenciar figuras geométricas. Considere apenas quadrados, triângulos e losangos. Crie classes que representem as figuras geométricas, considerando-se os atributos necessários para cada tipo de figura. Além disso, crie um método responsável em calcular e retornar a área para cada tipo de figura. Observação: a área do quadrado é dada por: $\text{lado} * \text{lado}$; a área do triângulo é dada por: $\text{base} * \text{altura} / 2$; e a área do losango é dada por: $D * d / 2$, onde D é a diagonal maior e d é a diagonal menor. Cada classe filha deverá ter um construtor recebendo as dimensões da figura e um construtor padrão. Também criar destrutores para as classes filhas que, quando invocados, exibam a mensagem: “Destruindo <tipo da figura>”. Exemplo: se for a classe Quadrado, exibir “Destruindo quadrado”. Fazer uma função main que teste todas as funcionalidades

Exemplo de Herança

- **Tempo** é a classe base
- **ExtTempo** é a classe derivada com herança pública
- A classe **derivada** pode
 - herdar todos os membros da classe base, exceto o construtor
 - acessar todos os membros publicos e protegidos da classe base
 - definir o seu membro de dados privado
 - fornecer seu próprio construtor
 - definir as suas funções de membros publicos
 - substituir funções herdadas da classe base



Especificação da classe **Tempo**

// ESPECIFICAÇÃO ARQUIVO

(Tempo.h)

class Tempo{

public:

void definir (int h, int m, int s);

void incremento ();

void escrever () const;

Tempo (int initH, int initM, int initS); // construtor

Tempo (); // construtor padrão

protected :

int horas;

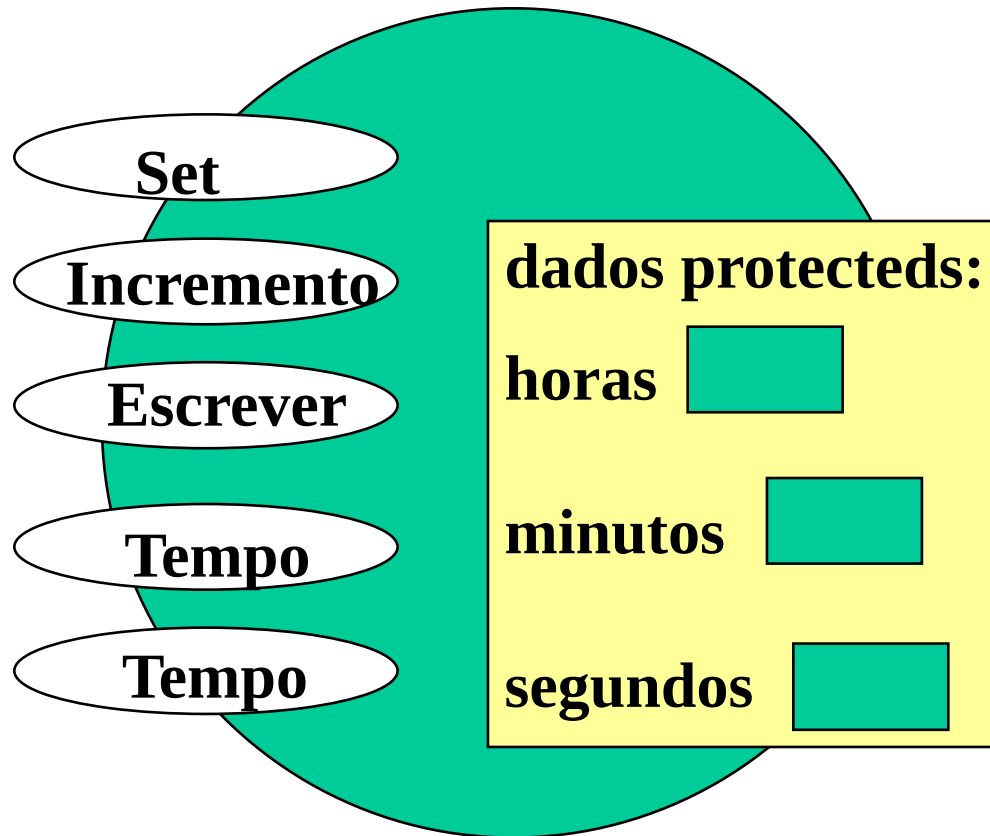
int minutos;

int segundos ;

};

Diagrama de Interface de Classe

Classe Tempo



Classe derivada **ExtTempo**

// ESPECIFICAÇÃO ARQUIVO

(ExtTempo.h)

#include “Tempo.h”

enum ZoneType {EST, CST, MST, PST, EDT, CDT, MDT, PDT};

classe ExtTempo : public Tempo

// Tempo é a classe base e usa a herança pública

{

public:

void definir (int h, int m, int s, timeZone ZoneType);

void escrever () const; //substituído

ExtTempo (int initH, int initM, int initS, ZoneType initZone);

ExtTempo (); // construtor padrão

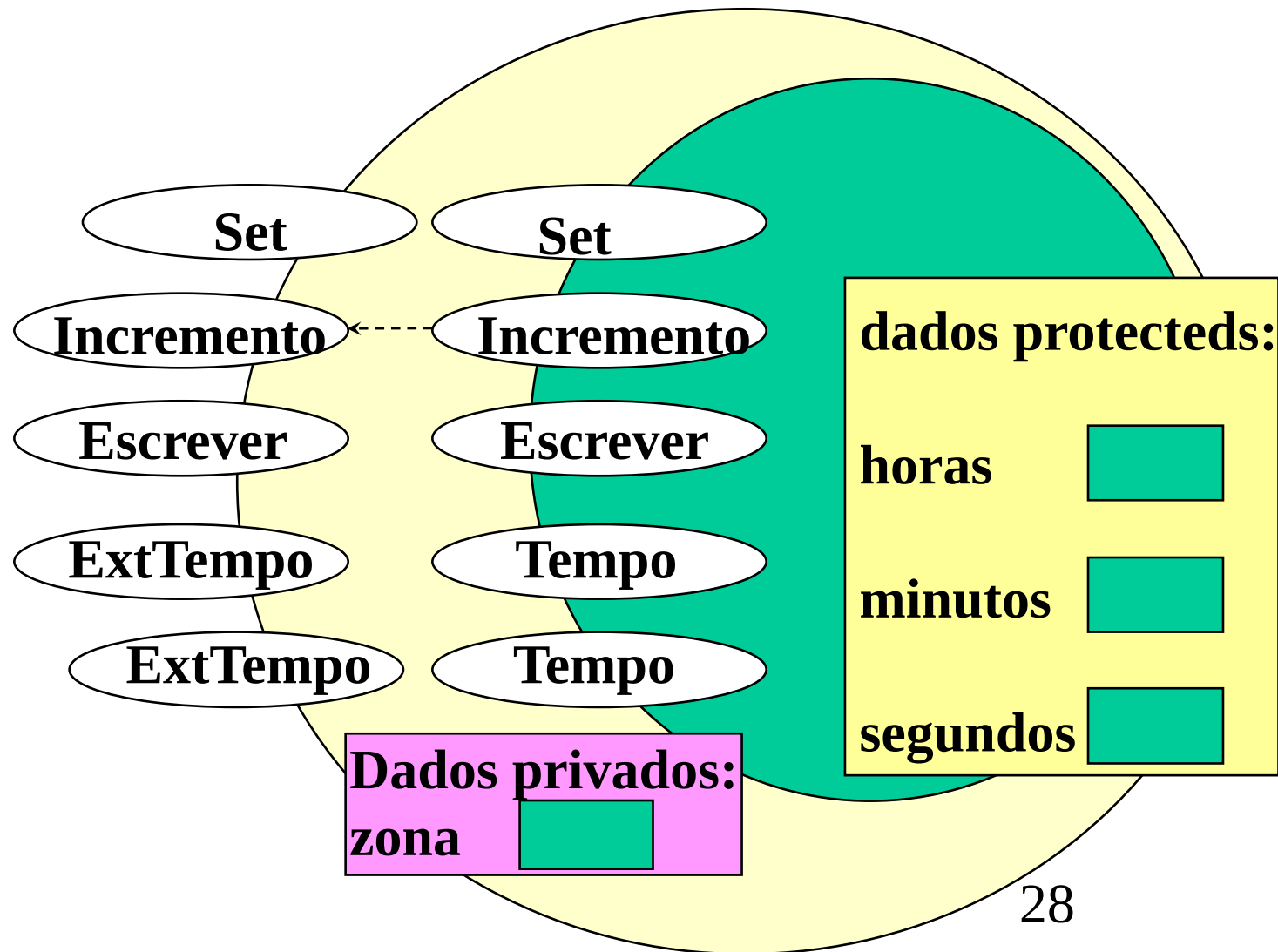
private :

ZoneType zona; // membro de dados adicionado

};

Diagrama de Interface de Classe

Classe ExtTempo



Implementação de **ExtTempo**

Construtor padrão

```
ExtTempo :: ExtTempo ( )  
{  
    zona = HUSA ;  
}
```

```
ExtTempo et1;
```

et1

<pre>h = 0 min = 0 seg = 0 zona = EST</pre>

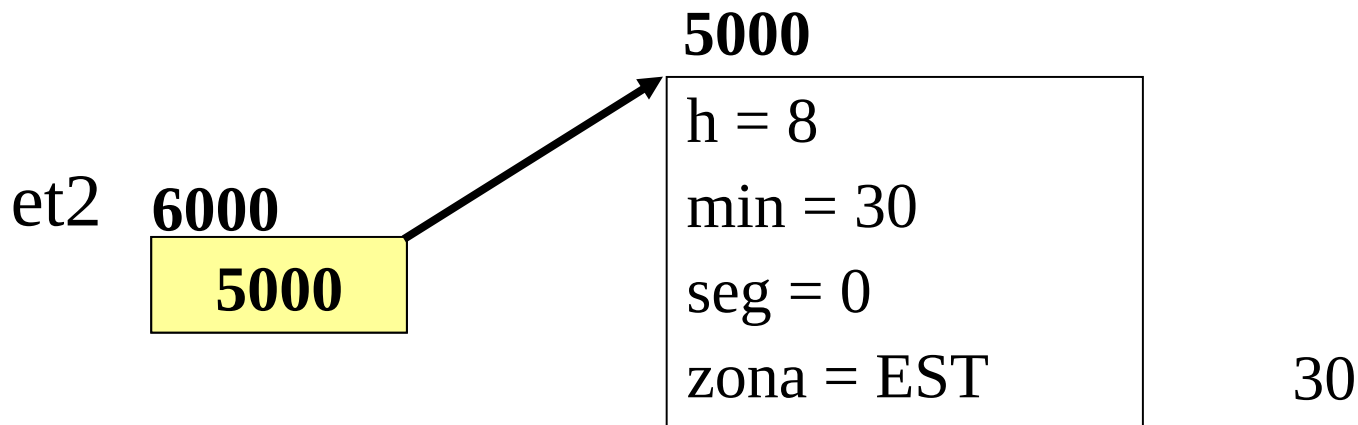
O construtor padrão da classe base, `Tempo ()`, é chamado automaticamente, quando um objeto `ExtTempo` é criado.

Implementação de ExtTempo

Outro Construtor

```
ExtTempo :: ExtTempo (int initH, int initM, int initS,  
    ZoneType initZone)  
    : Tempo (initH, initM, initS)  
    // constructor inicializador  
{  
    zona = initZone;  
}
```

```
ExtTempo *et2 = new ExtTempo (8,30,0, EST);
```



Implementação de ExtTempo

```
void ExtTempo :: Set (int h, int m, int s, ZoneType timeZone){  
    Tempo :: Set (horas, minutos, segundos); // chamada de função  
    mesmo nome  
    zona = timeZone;  
}
```

```
void ExtTempo :: Escrever () const // função primordial  
{  
    string zoneString[8] =  
        {"EST","OCS","MST","PST","EST","CDT","MST","PDT"};  
  
    Tempo :: Escrever ();  
    cout << ' ' << zoneString[zona] << endl;  
}
```

Trabalhando com **ExtTempo**

```
#include "ExtTempo.h"
```

```
... ..
```

```
int main ()
```

```
{
```

```
    ExtTempo esteTempo (8, 35, 0, PST);
```

```
    ExtTempo aqueleTempo;    //construtor padrão chamado
```

```
    aqueleTempo.Escrever ();    // saídas 00:00:00 EST
```

```
    aqueleTempo.Set (16, 49, 23, CDT);
```

```
    aqueleTempo.Escrever ();    // saídas 16:49:23 CDT
```

```
    esteTempo.incremento ();
```

```
    esteTempo.incremento ();
```

```
    esteTempo.Escrever ();    // saídas 08:35:02 PST
```

```
}
```