



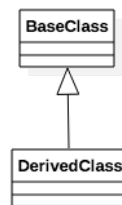
# Programação Orientada a Objetos

Prof. Hugo Marcondes  
hugo.marcondes@ifsc.edu.br

Aula 09

Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

- Herança permite a definição de uma classe em termos de outra classe
  - Classe Base
  - Classe Derivada



`class` DerivedClass: `access-specifier` BaseClass

# Exemplo



INSTITUTO FEDERAL  
SANTA CATARINA

```
// Base class
class Shape {
public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
protected:
    int width;
    int height;
};

// Derived class
class Rectangle: public Shape
{
public:
    int getArea() {
        return (width * height);
    }
};
```

```
int main(void)
{
    Rectangle Rect;

    Rect.setWidth(5);
    Rect.setHeight(7);










    // Print the area of the object.
    cout << "Total area: " << Rect.getArea() << endl;

    return 0;
}
```



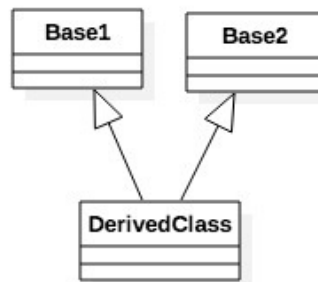
- Uma classe derivada herda todos os atributos e métodos da classe base, com as seguintes exceções
- Construtores, Destrutores e Construtores de cópia da classe Base
- Operadores sobrecarregados da classe Base
- Funções “amigas” (friend functions) da classe Base

- Os atributos e métodos de uma classe podem ser qualificadas de acordo com a sua acessibilidade.
- Os qualificadores de acesso em C++ podem ser sumarizados de acordo com a tabela abaixo:

Acesso	public	protected	private
Interno			
Classe Derivada			
Externo			



- Em C++ as heranças também podem ser qualificadas com qualificadores de acesso
  - Determinam a visibilidade dos métodos herdados pela classe Base, na classe Derivada
- Herança Pública (**public**)
  - Não alteram os qualificadores dos métodos da classe Base, na classe Derivada.
- Herança Protegida (**protected**)
  - Os métodos público e protegidos da classe Base, se tornam métodos protegidos na classe Derivada
- Herança Privada (**private**)
  - Os métodos públicos e protegidos da classe Base, se tornam métodos privados na classe Derivada



`class` **DerivedClass**: `access` **Base1**, `access` **Base2**, ...

# Exemplo



INSTITUTO FEDERAL  
SANTA CATARINA

```
// Base class Shape
class Shape {
public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
protected:
    int width;
    int height;
};

// Base class PaintCost
class PaintCost {
public:
    int getCost(int area) {
        return area * 70;
    }
};
```

```
// Derived class
class Rectangle: public Shape, public PaintCost {
public:
    int getArea() {
        return (width * height);
    }
};

int main(void) {
    Rectangle Rect;
    int area;

    Rect.setWidth(5);
    Rect.setHeight(7);

    area = Rect.getArea();
    // Print the area of the object.
    cout << "Total area: " << Rect.getArea() << endl;

    // Print the total cost of painting
    cout << "Total paint cost: $";
    cout << Rect.getCost(area) << endl;

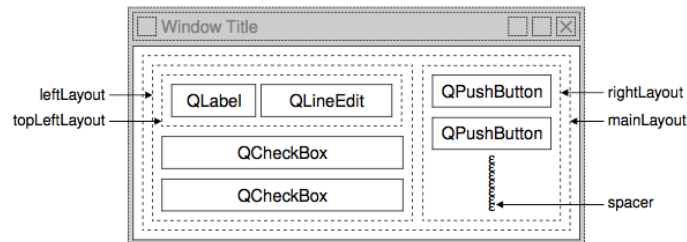
    return 0;
}
```



# QT: Dialogs



- Boa parte do desenvolvimento baseado no QT é feito através da herança de classes dos componentes providos pelo QT
- Vamos analisar a implementação do exemplo abaixo



**Figure 2.2.** The Find dialog's layouts



- Signals e Slots são mecanismos fundamentais da programação utilizando o QT
  - Permitem a interação de objetos de forma ALTAMENTE desacoplada
- Slots são similares a funções membro do C++, a diferença é que estas podem ser conectadas a um ou mais signal, sendo executada automaticamente toda vez que o signal é emitido.
  - Esta associação é realizada pelo método `connect()` da classe `QObject`

```
connect(sender, SIGNAL(signal), receiver, SLOT(slot));
```



- A conexão entre sinais e slots é muito versátil
- Um sinal pode ser conectado em diversos slots

```
connect(slider, SIGNAL(valueChanged(int)),
        spinBox, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)),
        this, SLOT(updateStatusBarIndicator(int)));
```

- Muitos sinais podem ser conectados a um mesmo slot

```
connect(lcd, SIGNAL(overflow()),
        this, SLOT(handleMathError()));
connect(calculator, SIGNAL(divisionByZero()),
        this, SLOT(handleMathError()));
```

- Um sinal pode ser conectado a outro sinal

```
connect(lineEdit, SIGNAL(textChanged(const QString &)),
        this, SIGNAL(updateRecord(const QString &)));
```

- Para conectar sinais e slots é fundamental
  - Ambos devem possuir a mesma assinatura de parâmetros (tipo e ordem)
  - Excepcionalmente, se um sinal possuir um número maior de parâmetros, os parâmetros adicionais são ignorados.

```
connect(ftp, SIGNAL(rawCommandReply(int, const QString &)),  
        this, SLOT(checkErrorCode(int)));
```

- Apesar de amplamente utilizado no conceito de Widgets, a infra-estrutura de signal e slots pode ser extensivamente utilizado em outros componentes e como uma ferramenta de implementação das suas próprias classes

```
class Employee: public QObject {  
  
    Q_OBJECT  
  
    public:  
        Employee() { mySalary = 0; }  
        int salary() const {  
            return mySalary; }  
    public slots:  
        void setSalary(int newSalary);  
    signals:  
        void salaryChanged(int newSalary);  
  
    private:  
        int mySalary;  
};
```

```
void Employee::setSalary(int newSalary) {  
    if (newSalary != mySalary) {  
        mySalary = newSalary;  
        emit salaryChanged(mySalary);  
    }  
}
```