Separando a Interface da Implementação

Separando a Interface da Implementação

- Uma vantagen de definir classes é que, quando empacotadas apropriadamente, elas podem ser reutilizadas
  - Como por exemplo, a classe ***string***.
- Nosso exemplo não pode ser reutilizado em outro programa
  - Já contém um *main*, e todo programa deve possuir apenas um *main*.
- Claramente, colocar um *main* no mesmo arquivo que contém uma classe impede sua reutilização
  - Para resolver isto, separamos os arquivos.

Headers

- A classe fica em um arquivo de cabeçalhos **.h** (*header*)
  - Lembre-se que ao final da classe colocamos **;**
- O *main* fica em um arquivo de código-fonte **.cpp** (*source*);
- Desta forma, o arquivo **.cpp** deve incluir o arquivo **.h** para reutilizar o código
  - Compilamos apenas o arquivo **.cpp**.
- Vejamos como fica nosso exemplo.

GradeBook.h

```
#include <iostream>
#include <string>
using namespace std;

class GradeBook
{
public:
   GradeBook( string name )
   {
      setCourseName( name );
   }

    void setCourseName( string name )
   {
      courseName = name;
   }

   string getCourseName()
   {
      return courseName;
   }

   void displayMessage()
   {
      cout<<"Welcome to the grade book for\n"<<getCourseName() <<"!"<<endl;
   }
private:
   string courseName;
};
```

Main.cpp

```
#include <iostream>
using namespace std;
#include "GradeBook.h" // inclui a definição de classe GradeBook

int main()
{
   // cria dois objetos GradeBook
   GradeBook gradeBook1("BCC221 - POO");
   GradeBook gradeBook2("BCC202 – AED's I");

   cout << "gradeBook1 created for course: "
        << gradeBook1.getCourseName()
        << "\ngradeBook2 created for course: "
        << gradeBook2.getCourseName()<< endl;
   return 0;
}
```

Separando a Interface da Implementação

- Um problema relacionado a esta divisão de arquivos é que o usuário da classe vai conhecer a implementação
  - O que não é recomendável.
- Permite que o usuário escreva programas baseado em detalhes da implementação da classe
  - Quando na verdade deveria apenas saber quais métodos chamar, sem saber seu funcionamento;
  - Se a implementação da classe for alterada, o usuário também precisará alterar seu programa.
- Podemos então separar a **interface** da **implementação**.

Interfaces

- A **interface** de uma classe especifica quais serviços podem ser *utilizados* e como *requisitar* estes serviços
  - E não como os serviços são realizados.
- A interface pública de uma classe consiste dos métodos públicos
  - Em nosso exemplo, o construtor, o *getter*, o *setter* e o método *displayMessage*.

Interfaces

- Separamos então nossa classe em dois arquivos:
  - A definição da classe e protótipos dos métodos são feitos no arquivo **.h**;
  - A implementação dos métodos é definida em um arquivo **.cpp** separado;
  - Por convenção os arquivos possuem o mesmo nome, diferenciados apenas pela extensão.
- O *main* é criado em um terceiro arquivo, também com extensão **.cpp**
  - *GradeBook.h*
  - *GradeBook.cpp*
  - *Main.cpp*

GradeBook.h

```
#include <string>
using namespace std;

// Definição da classe GradeBook
class GradeBook
{
 public:
   GradeBook( string );
   void setCourseName( string );
   string getCourseName();
   void displayMessage();
 private:
   string courseName;
};
```

GradeBook.h

- Na definição da classe, temos apenas a declaração dos atributos e dos protótipos dos métodos
  - Apenas o cabeçalho dos métodos
    - Sempre terminados com **;**
  - Note que não é necessário definir um nome para os atributos, apenas o tipo.

GradeBook.cpp

```
#include <iostream>
using namespace std;
#include "GradeBook.h" // inclui a definição de classe GradeBook

GradeBook::GradeBook( string name )
{
   setCourseName( name );
}
void GradeBook::setCourseName( string name )
{
   courseName = name;
}
string GradeBook::getCourseName()
{
   return courseName;
}
void GradeBook::displayMessage()
{
   cout << "Welcome to the grade book for\n" << getCourseName()
      << "!" << endl;
}
```

GradeBook.cpp

- No arquivo de definição dos métodos deve ser incluído o arquivo **.h** com a definição da classe;
- Note que após o tipo de cada método, incluímos o nome da classe seguido de **::**
  - Operador de resolução de escopo
    - "Amarra" a implementação ao protótipo definido no outro arquivo;
    - Sem isto, serão considerados como funções, não relacionadas à classe.
- Na definição dos métodos é necessário dar nomes aos parâmetros.

Main.cpp

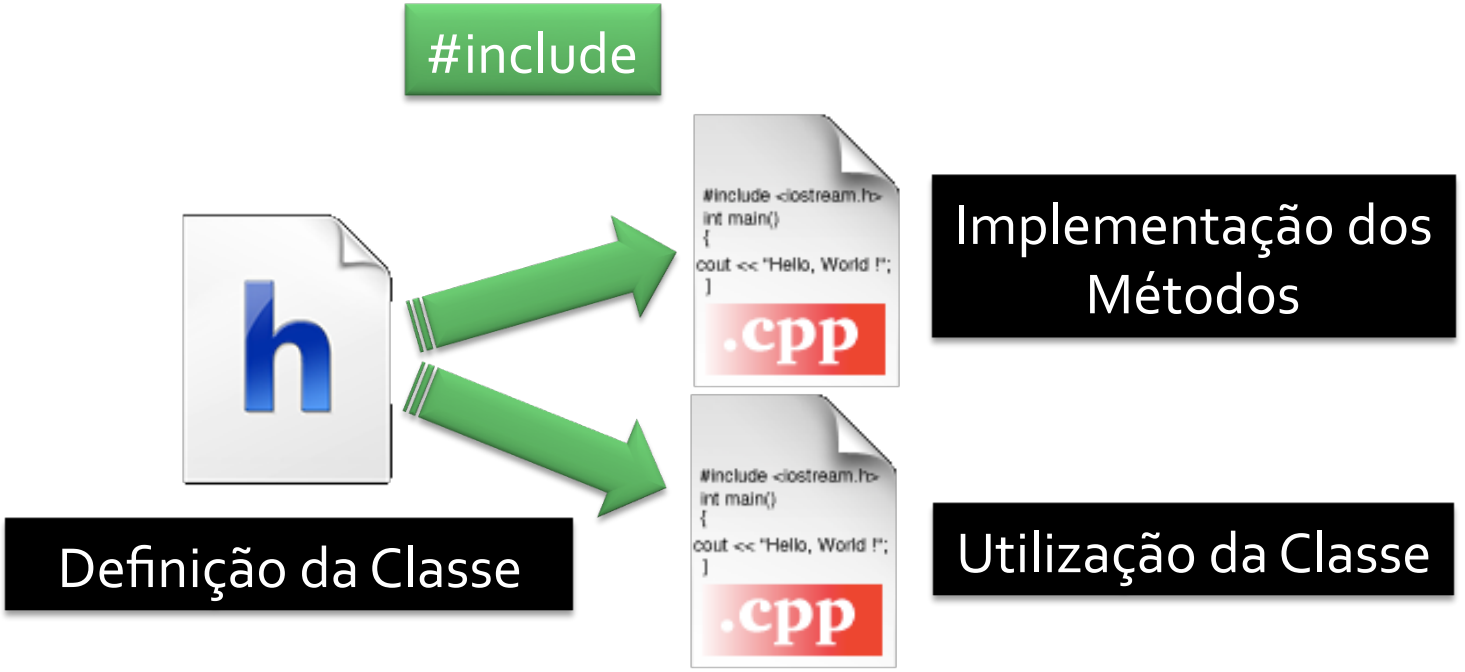
```
#include <iostream>
using namespace std;

#include "GradeBook.h" //inclui a definição de classe GradeBook

int main()
{
   GradeBook gradeBook1("BCC221 - POO");
   GradeBook gradeBook2("BCC202 – AED's I");

   cout << "gradeBook1 created for course: "
        <<gradeBook1.getCourseName()
        << "\ngradeBook2 created for course: "
        <<gradeBook2.getCourseName()<< endl;
   return 0;
}
```

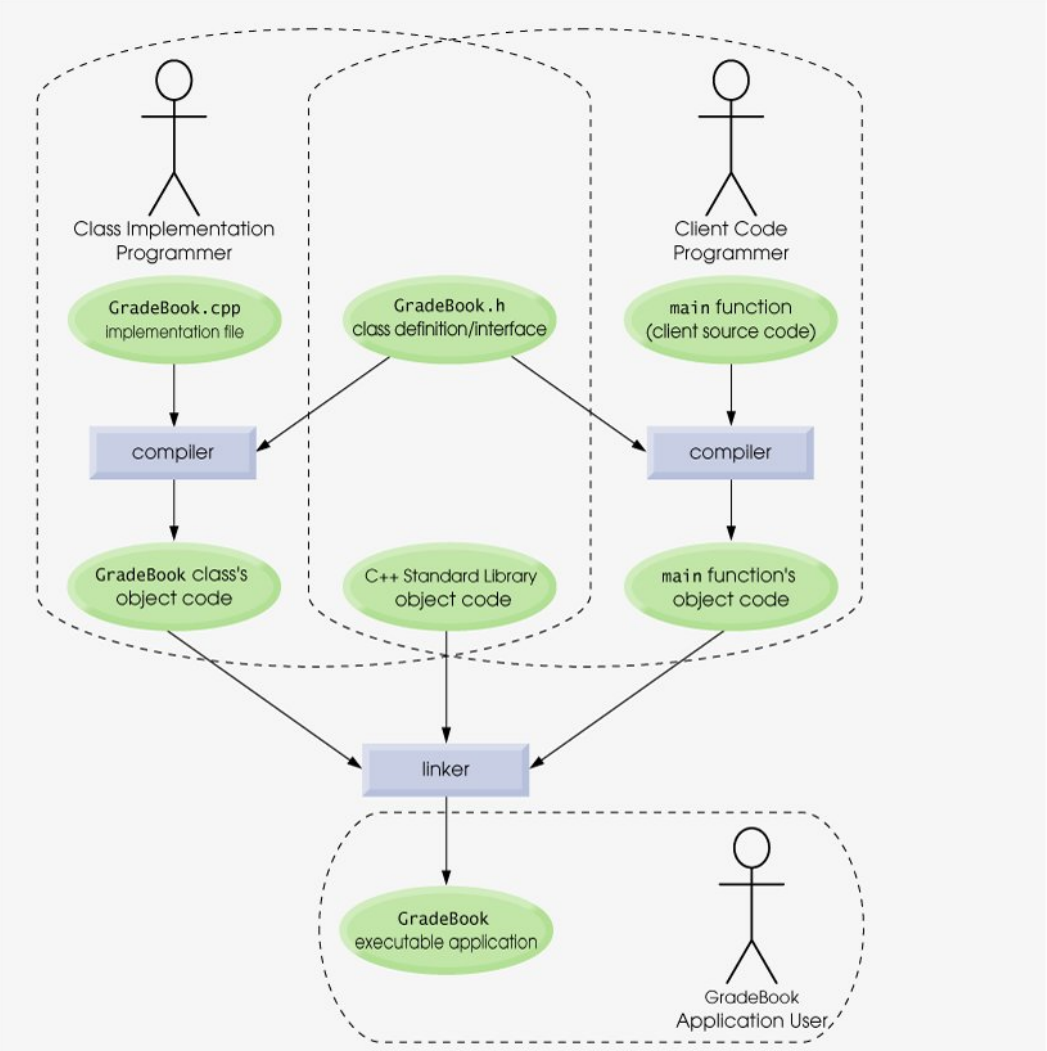
Interfaces – Inclusão de Arquivos



Interfaces - Compilação

- Quando separados desta forma, precisamos compilar três vezes
  - g++ -c Classe.cpp
    - Gera um arquivo **Classe.o**, que deve ser entregue ao usuário da classe;
    - Precisa dos arquivos **Classe.h** e **Classe.cpp**.
  - g++ -c Main.cpp
    - Gera um arquivo **main.o**;
    - Precisa dos arquivos **Classe.h** e **main.cpp**.
  - g++ Classe.o main.o –o programa
    - Gera o executável;
    - Não precisa do arquivo **Classe.cpp**, escondendo a implementação.

Interfaces - Compilação



Interfaces - Compilação

- Uma alternativa para compilar múltiplos arquivos é utilizar ***makefiles***
  - É uma forma automatizada para executar tarefas em terminal;
  - No próximo slide há um *Makefile* para realizar a compilação exemplificada anteriormente
    - O conteúdo deve ser salvo em um arquivo chamado ***Makefile*** (sem extensão);
    - Os arquivos devem se chamar ***main.cpp*** e ***metodos.cpp***;
    - Para executar, digite ***make*** no terminal, dentro da pasta onde estão os arquivos;
    - O resultado será um arquivo executável chamado **programa**.

Makefile

```
OBJS=main.o metodos.o
all: programa
programa: $(OBJS)
	g++ $(OBJS) -o $@
main.o: main.cpp
	g++ -c main.cpp -o main.o -Wall
metodos.o: metodos.cpp
	g++ -c $< -o $@ -Wall
```