Construtores

Construtores

- Quando um objeto da classe *GradeBook* é criado, a sua cópia do atributo *courseName* é inicializada como vazia
  - Por padrão.
- Mas e se quiséssemos que o atributo fosse inicializado com um valor padrão?
  - Podemos criar um método **construtor**, para inicializar cada objeto criado.

Construtores

- Um construtor é um método especial, definido com o **mesmo nome da classe** e executado automaticamente quando um objeto é criado
  - Não retorna valores;
  - Não possui valor de retorno;
  - Deve ser declarado como público.
- Se não especificarmos um construtor, o compilador utilizará o construtor padrão
  - No nosso exemplo, foi utilizado o construtor padrão da classe *string*, que a torna vazia.
- Vejamos nosso exemplo, agora com um construtor.

Construtores

```
class GradeBook
{
public:
   // o construtor inicializa courseName com a string fornecida como argumento
   GradeBook( string name )
   {
      setCourseName( name ); // chama a função set para inicializar courseName
   } // fim do construtor GradeBook

   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;
};
```

Construtores

```
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;
}
```

Construtores

- Notem que um construtor pode possuir parâmetros ou não
  - Por exemplo, poderíamos não passar nenhum parâmetro e definir um valor padrão dentro do próprio construtor.
- Quando um atributo for objeto de outra classe, podemos chamar o construtor da outra classe em um construtor definido por nós
  - E opcionalmente, especificar inicializações adicionais.
- Todas nossas classes devem possuir construtores, para evitarmos lixo em nossos atributos.

Construtores

- É possível criarmos mais de um construtor na mesma classe
  - Sobrecarga de construtores
    - O **construtor *default*** não possui parâmetros.
  - Da mesma forma que sobrecarregamos funções;
  - A diferenciação é feita pelo número de parâmetros enviados no momento da criação do objeto
    - Diferentes objetos de uma mesma classe podem ser inicializados por construtores diferentes.
  - Escolhemos qual construtor é mais adequado a cada momento.

Construtores

- Podemos ainda ter construtores com parâmetros padronizados
  - O construtor recebe parâmetros para inicializar atributos;
  - Porém, define parâmetros padronizados, caso não receba nenhum parâmetro.
- Suponha uma classe *Venda*, em que temos os atributos valor e peças
  - Ao criar um objeto, o programador pode definir a quantidade de peças e o valor da venda;
  - Porém, se nada for informado, inicializaremos os atributos com o valor -1, usando o mesmo construtor;
  - É uma forma de economizar o trabalho de sobrecarregar um construtor.

Construtores

```
class Vendas
{
  public:
  //parametros padrão
  Vendas(int p= -1, float v= -1.0)
  {
     valor = v;
     pecas = p;
  }
  float getValor()
  {
     return valor;
  }
  int getPecas()
  {
     return pecas;
  }
  private:
  float valor;
  int pecas;
};
```

```
int main()
{
  //inicializa um objeto com -1 e outro com 10
   Vendas a, b(10, 10);

   cout <<a.getPecas()<<endl
        <<a.getValor()<<endl
        <<b.getPecas()<<endl
        <<b.getValor()<<endl;
     return 0;
}
```

Destrutores

Destrutores

- De forma análoga aos construtores, que inicializam objetos, temos os **destrutores**, que **finalizam objetos**
  - São chamados automaticamente quando um objeto for destruído, por exemplo, ao terminar o seu bloco de código;
  - São indicados por um ~ antes do nome do método, que deve ser igual ao da classe.

Destrutores

- **Destrutores**:
  - Não possuem valor de retorno;
  - Não podem receber argumentos;
  - Não podem ser chamados explicitamente pelo programador.
- **Atenção!**
  - Se um programa terminar por uma chamada ***exit()*** ou ***abort()***, o destrutor não será chamado.

Destrutores

- Vejamos um exemplo em que o construtor de uma classe incrementa um atributo a cada vez que um objeto é criado e o destrutor decrementa o mesmo atributo a cada vez que um objeto é destruído;
- Como seria possível se cada objeto possui uma cópia diferente de cada atributo?
  - Usamos o modificador ***static***, que faz com que haja apenas um atributo compartilhado por todos os objetos.

Destrutores

```
class Rec
{
        private:
                static int n;//cria um único item para todos os objetos
        public:
                Rec()
                {
                        n++;
                }

                int getRec()
                {
                        return n;
                }

                ~Rec()
                {
                        n--;
                }
};
```

Destrutores

```
int Rec::n=0;//necessário para que o compilador crie a variável

int main()
{
      Rec r1, r2, r3;
      cout<<r1.getRec()<<endl;

      {
            Rec r4, r5, r6;//só valem dentro deste bloco
            cout<<r1.getRec()<<endl;
      }

        cout<<r1.getRec();

      return 0;
}
```

Destrutores

- Construtores e destrutores são especialmente úteis quando os objetos utilizam alocação dinâmica de memória
  - Alocamos a memória no construtor;
  - Desalocamos a memória no destrutor.
- Novamente, destrutores são geralmente omitidos em diagramas de classes UML.

Construtores Parametrizados e Vetores de Objetos

Construtores Parametrizados e Vetores de Objetos

- No caso de termos um vetor de objetos, recomenda-se não utilizar construtores parametrizados
  - Ou então utilizar construtores com parâmetros padronizados.
- Caso seja realmente necessário, no momento da declaração do vetor é necessário inicializá-lo, fazendo a atribuição de **objetos anônimos**
  - De forma parecida com a inicialização de vetores de tipos primitivos;
  - Cada objeto anônimo deve enviar seus parâmetros para o construtor.

Construtores Parametrizados e Vetores de Objetos

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

class Numero
{
	public:
		Numero(int n)
		{
			valor = n;
		}
		int getNumero()
		{
			return valor;
		}
	private:
		int valor;
};
```

```
int main()
{
	Numero vet[3] = {Numero(0), Numero(1), Numero(2)};
	int i;

	for(i=0; i<3; i++)

cout<<vet[i].getNumero()<<endl;

	return 0;
}
```