Makefile e Princípios de Projeto Orientado a Objetos

Roland Teodorowitsch

Programação Orientada a Objetos - ECo - Curso de Engenharia de Computação - PUCRS

25 de setembro de 2023

Makefile

Idéia Geral

- Automatiza a compilação e a geração do executável ("linkar") de projetos de *software*
- Comando:
  `make`
- Arquivo texto contendo as diretivas de compilação e "linkagem":
  `Makefile`
- Ao ser chamado, o comando `make` procura por um arquivo chamado `Makefile` que diz o que o `make` deve executar
- É padrão no Unix, mas está disponível em outros sistemas
- Referência:
  `http://mrbook.org/tutorials/make/`

Uso

- Caso `make` seja executado SEM um alvo (`make`):
  - Se houver `Makefile`, o rótulo (ou alvo) `all` será buscado no `Makefile` para se determinar o que fazer
  - Se não houver, ocorrerá um erro
- Caso `make` seja executado COM um alvo (`make` *`alvo`*):
  - Se houver `Makefile`, o rótulo `alvo` será buscado no `Makefile` para se determinar o que fazer
  - Se não houver, `make` usará regras internas para criar um executável (`alvo`) a partir de códigos-fontes (`alvo.c`, `alvo.cpp`, etc.) – ocorrerá um erro se nenhum código-fonte reconhecido for encontrado
- Pode-se indicar um arquivo com outro nome (diferente de `Makefile`) para o `make`:

```
make -f outro_makefile
```

Exemplo de um Projeto de *Software*

- Considere um projeto em C++ formado pelos seguintes arquivos:
  - `main.cpp`: programa principal
  - `Data.hpp` e `Data.cpp`: definição e implementação de uma classe para modelar uma data formada por dia, mês e ano
  - `Produto.hpp` e `Produto.cpp`: definição e implementação da classe para modelar produtos com data de validade
- Pode-se gerar um executável chamado app usando:

```
g++ -std=c++11 main.cpp Data.cpp Produto.cpp -o app
```

- Isto implica compilar sempre tudo a qualquer alteração, o que diminui a escalabilidade
- Uma solução mais interessante é criar um `Makefile` que:
  - Compila arquivos `.cpp` com a opção `-c` gerando arquivos-objeto (extensão `.o`)
  - Gera o executável ("linkagem") a partir dos arquivos-objeto

Formato do Makefile

- Comentários iniciam com #:

```
# Esta eh uma linha de comentario
# Esta eh outra linha de comentario
```

- É possível criar variáveis usando VARIAVEL=conteudo, como no *shell script*:

```
# Comando de compilacao
CC=g++
# Flags de compilacao
CFLAGS=-c -std=c++11
```

- Para acessar o valor da variável basta usar ${VARIAVEL}

Formato do Makefile

- Basicamente o Makefile é formado por regras com o seguinte formato:
  rótulo: *dependências*
  comandos (precedidos de pelo menos um TAB)
- Onde:
  - O *rótulo* geralmente é o que se deseja construir e pode ser usado como nome na linha de comando do make
  - As *dependências* são o que é preciso para se construir o que foi especificado no rótulo
  - Os *comandos* indicam como construir o rótulo a partir das dependências
- Exemplo: gerar um arquivo Conta.o a partir de Conta.cpp e Conta.hpp

```
Produto.o:  Produto.cpp Produto.hpp Data.hpp
            g++ ${CFLAGS} Produto.cpp
```

- Para ocultar a exibição dos comandos que o make executa, pode-se precedê-los com @

Exemplo Completo

```
# Makefile (Roland Teodorowitsch; 9 set. 2019)

EXECUTAVEL=app
CFLAGS=-c -std=c++11

all:            ${EXECUTAVEL}

${EXECUTAVEL}:  main.o Data.o Produto.o
                @g++ main.o Data.o Produto.o -o ${EXECUTAVEL}

main.o:         main.cpp Data.hpp Produto.hpp
                @g++ ${CFLAGS} main.cpp

Produto.o:      Produto.cpp Produto.hpp Data.hpp
                @g++ ${CFLAGS} Produto.cpp

Data.o:         Data.cpp Data.hpp
                @g++ ${CFLAGS} Data.cpp

run:            ${EXECUTAVEL}
                @./${EXECUTAVEL}

clean:
                @rm -f Produto.o Data.o main.o ${EXECUTAVEL}
```

Princípios de Projeto Orientado a Objetos

Princípios de Projeto Orientado a Objetos (POO)

- Os **objetivos** do projeto de uma classe são construir classes:
  - De alta qualidade
  - Com interfaces públicas consistentes
  - Úteis e reutilizáveis
- Critérios de qualidade da **interface pública** de uma classe:
  - **Coesão**
  - Completude
  - Conveniência
  - Clareza
  - Consistência
  - Acoplamento
  - **Encapsulamento**

Coesão

Coesão

- Medida da inter-relação dos membros (atributos e métodos) da interface pública de uma classe
- Indica o grau em que uma classe possui uma finalidade única e bem orientada, ou seja, uma classe não deve assumir responsabilidades que não são suas
- Na prática:
  - Alta coesão é o estado desejável de uma classe cujos membros suportam uma única regra ou responsabilidade bem focada
  - Baixa coesão é o estado indesejável de uma classe cujos membros suportam múltiplas regras ou responsabilidades desfocadas

Exemplo 1 - Baixa Coesão

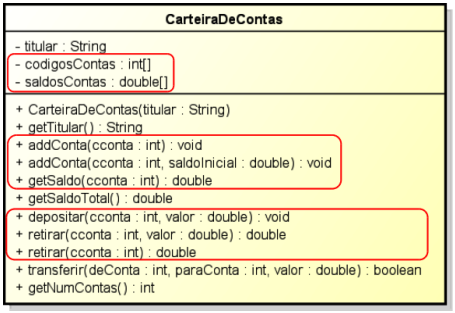
```
#include <string>
using namespace std;
class Aluno {
  private:
    string nome;
    double notas[3];
  public:
    Aluno(string nome, double notas[3]) {
      this->nome = nome;
      defineNotas(notas);
    }
    string obtemNome() {
      return nome;
    }
    void defineNotas(double notas[3]) {
      for (int i=0; i<3; ++i)
          this->notas[i] = notas[i];
    }
    double calculaMedia() {
      double soma = 0.0;
      for (int i=0; i<3; ++i)
          soma += notas[i];
      return soma / 3.0;
    }
};
```

Exemplo 1 - Alta Coesão

```
#include <string>

using namespace std;

class Aluno {
  private:
    string nome;
    Notas notas;
  public:
    Aluno(string nome, double notas[3]) {
      this->nome = nome;
      this->notas.defineNotas(notas);
    }

    string obtemNome() {
      return nome;
    }

    void defineNotas(double notas[3]) {
      this->notas.defineNotas(notas);
    }

    double calculaMedia() {
      return notas.calculaMedia();
    }
};
```

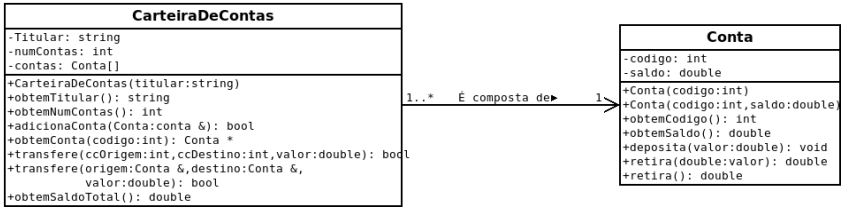
```
class Notas {
  private:
    double notas[3];
  public:
    Notas() {
      for (int i=0; i<3; ++i)
          notas[i] = 0.0;
    }

    Notas(double notas[3]) {
      defineNotas(notas);
    }

    void defineNotas(double notas[3]) {
      for (int i=0; i<3; ++i)
          this->notas[i] = notas[i];
    }

    double calculaMedia() {
      double soma = 0.0;
      for (int i=0; i<3; ++i)
          soma += notas[i];
      return soma / 3.0;
    }
};
```

Exemplo 2 - Baixa Coesão



- Esta classe tende a ter dificuldades de manutenção e de reuso!
- Devemos ao máximo evitar essas "super-classes" que fazem tudo no seu sistema

Exemplo 2 - Alta Coesão



- Classes mais especializadas!
- Classes mais fáceis de manter (e menos frequentemente alteradas)
- Tendem a ser mais reutilizadas

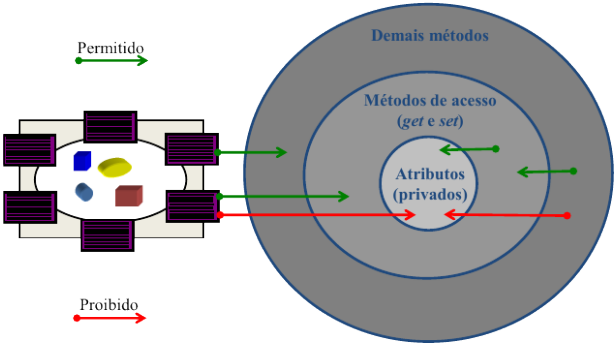
Encapsulamento

Encapsulamento

- Isolamento do estado interno de uma estrutura de software através de uma interface pública
- Encapsular é sinônimo de esconder. Logo, um código encapsulado é aquele código que diz o que faz mas não diz como se faz no primeiro momento. Para saber como faz você deve acessar outro trecho de código. Por isso no encapsulamento devemos prezar pela boa semântica ao darmos nomes aos métodos. Quem lê o nome do método deve saber o que ele faz
- Classes bem encapsuladas têm:
  - Definição clara da interface pública
  - Estado interno escondido
- A vantagem de uma classe bem encapsulada é que ela simplifica o entendimento do sistema e a manutenção

Técnica de Anéis de Operações

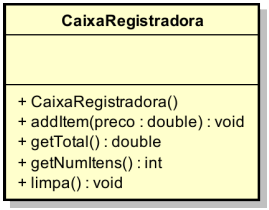
- Ajuda a manter um bom encapsulamento interno da classe:
  - O uso dessa técnica não afeta o acesso externo (que continua sendo regido por modificadores de visibilidade)
  - Nessa técnica são criados três anéis fictícios na classe
  - Os métodos de anéis externos acessam sempre métodos (ou atributos) de anéis internos consecutivos



Exemplo

- Exemplo de bom encapsulamento...

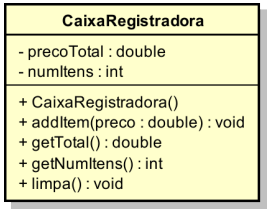
- Interface Pública bem definida!



- Exemplo de uso:

```
int main() {
  CaixaRegistradora *caixa = new CaixaRegistradora();

  //adiciona tres itens
  caixa->addItem(1.99);
  caixa->addItem(2.99);
  caixa->addItem(1.50);

  cout << caixa->getTotal() << endl;     // saida: 6.48
  cout << caixa->getNumItens() << endl; // saida: 3

  caixa->limpa();

  cout << caixa->getTotal() << endl;     // saida: 0
  cout << caixa->getNumItens() << endl; // saida: 0

  delete caixa;
  return 0;
}
```

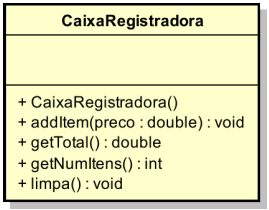
Exemplo: Implementação 1




```
class CaixaRegistradora {
  private:
    double precoTotal;
    int numItens;
  public:
    CaixaRegistradora() {
      precoTotal = 0.0;
      numItens = 0;
    }
    void addItem(double preco) {
      precoTotal += preco;
      numItens++;
    }
    double getTotal() {
      return precoTotal;
    }
    int getNumItens() {
      return numItens;
    }
    void limpa() {
      precoTotal = 0.0;
      numItens = 0;
    }
};
```

Implementação 2

- Quando o encapsulamento é bem feito, pode-se alterar a implementação sem alterar a Interface Pública
- O mesmo programa de teste funcionará para as duas implementações




```
class CaixaRegistradora {
private:
   double itens[100];
   int numItens;
public:
   CaixaRegistradora(){
      numItens = 0;
   }
   void addItem(double preco) {
      itens[numItens] = preco;
      numItens++;
   }
   double getTotal() {
      double total = 0;
      for (int i = 0; i < numItens; i++)
          total += itens[i];
      return total;
   }
   int getNumItens() {
      return numItens;
   }
   void limpa() {
      numItens = 0;
   }
};
```

Comparando as 2 Implementações...

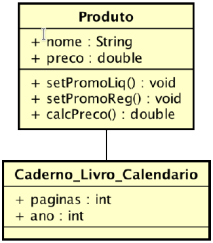
```
class CaixaRegistradora {
private:
   double precoTotal;
   int numItens;
public:
   CaixaRegistradora() {
      precoTotal = 0.0;
      numItens = 0;
   }
   void addItem(double preco) {
      precoTotal += preco;
      numItens++;
   }
   double getTotal() {
      return precoTotal;
   }
   int getNumItens() {
      return numItens;
   }
   void limpa() {
      precoTotal = 0.0;
      numItens = 0;
   }
};
```

```
class CaixaRegistradora {
private:
   double itens[100];
   int numItens;
public:
   CaixaRegistradora(){
      numItens = 0;
   }
   void addItem(double preco) {
      itens[numItens] = preco;
      numItens++;
   }
   double getTotal() {
      double total = 0;
      for (int i = 0; i < numItens; i++)
          total += itens[i];
      return total;
   }
   int getNumItens() {
      return numItens;
   }
   void limpa() {
      numItens = 0;
   }
};
```

Lista de Exercícios

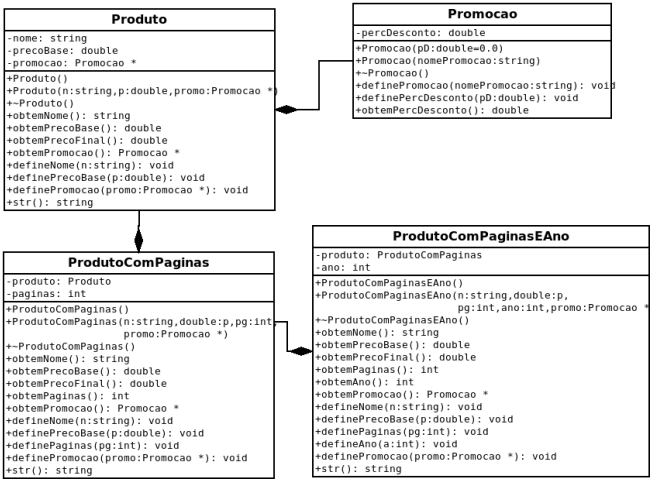
Exercício 1

1. Uma livraria possui alguns produtos à venda, como livros, cadernos, calendários e canetas. Todos os produtos possuem nome e preço. Cadernos possuem também o número de páginas, como os livros e calendários, mas estes últimos se diferenciam por ainda terem o atributo ano. Esta loja é conhecida pelas promoções que oferece. Uma promoção Regular oferece um desconto de 10% em cada produto. Já a Liquidação oferece um desconto de 30% no preço de cada produto.



- Um colega apresentou a modelagem UML ao lado, e pediu para você verifica-la. Logo, você deve melhorar a modelagem do seu colega sempre pensando em termos classes encapsuladas com alta coesão! Além de implementa-las, é claro!

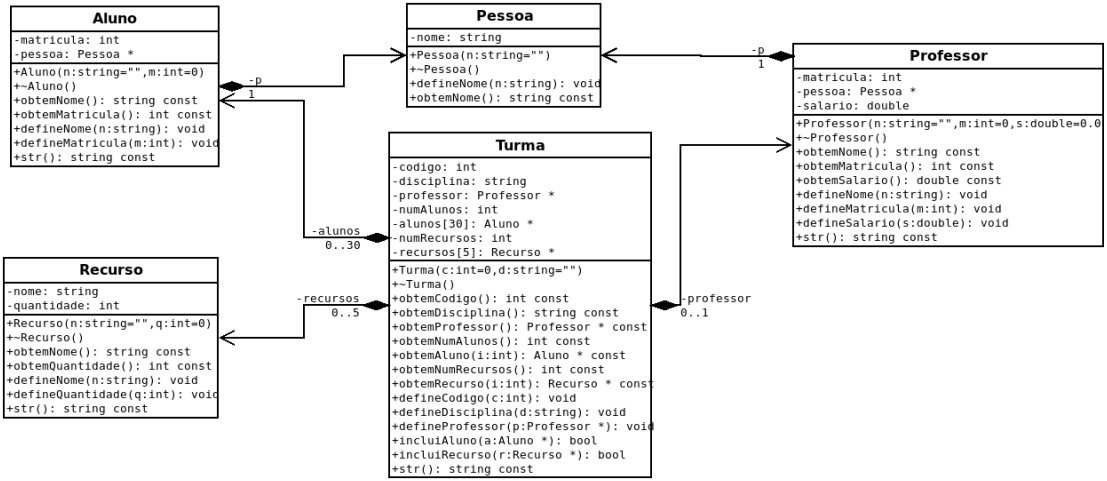
Exercício 1 - Exemplo de Modelagem



Exercício 2

2. Crie um programa que gerencie uma turma de determinada disciplina, contendo professor, alunos e recursos utilizados. O relacionamento entre as classes que deverão ser implementadas é mostrado no diagrama de classes da próxima página.
   - Uma turma pode ter no máximo 30 alunos matriculados.
   - Uma turma pode ter no máximo 5 recursos alocados.
   - Para testar a implementação, no programa principal (função `main()`), crie uma instância da classe `Turma`, e preencha-a com referências para objetos das classes `Professor`, `Aluno` (4 alunos) e `Recurso` (3 recursos). Inicialize estes objetoss na função `main()` e implemente os métodos `str()` das classes para que sejam mostradas as informações dos objetos como ilustrado no exemplo de execução que aparece logo após o diagrama de classes.

Exercício 2: Diagrama de Classes



Exercício 2: Exemplo de Execução

```
Turma: 30 - Programacao Orientada a Objetos (ECo)
Professor: Professor 1 (234; R$567.89)
Alunos:
- Aluno 1 (1)
- Aluno 2 (2)
- Aluno 3 (3)
- Aluno 4 (4)
Recursos:
- Computador (4)
- Projetor (1)
```

Créditos

Créditos

- Estas lâminas contêm trechos de materiais disponibilizados pelo professor Rafael Garibotti.

Soluções

Exercício 2: Pessoa.hpp

```
#ifndef _PESSOA_HPP
#define _PESSOA_HPP

#include<string>

using namespace std;

class Pessoa{
private:
  string nome;
public:
  Pessoa(string n = "");
  ~Pessoa();
  string obtemNome() const;
  void defineNome(string n);
};
#endif
```

Exercício 2: Pessoa.cpp

```cpp
#ifdef DEBUG
#include <iostream>
#endif
#include"Pessoa.hpp"

Pessoa::Pessoa(string n) {
  nome = n;
  #ifdef DEBUG
  cout << "+ Pessoa(" << nome << ") criada..." << endl;
  #endif
}

Pessoa::~Pessoa() {
  #ifdef DEBUG
  cout << "- Pessoa(" << nome << ") destruida..." << endl;
  #endif
}

string Pessoa::obtemNome() const {
  return nome;
}

void Pessoa::defineNome(string n) {
  nome = n;
}
```

Exercício 2: Aluno.hpp

```
#ifndef _ALUNO_HPP
#define _ALUNO_HPP

#include "Pessoa.hpp"

class Aluno {
private:
  int matricula;
  Pessoa *pessoa;
public:
  Aluno(string n = "", int m = 0);
  ~Aluno();
  string obtemNome() const;
  int obtemMatricula() const;
  void defineNome(string n);
  void defineMatricula(int m);
  string str() const;
};
#endif
```

Exercício 2: Aluno.cpp

```
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "Aluno.hpp"

Aluno::Aluno(string n, int m) {
  pessoa = new Pessoa(n);
  matricula = m;
  #ifdef DEBUG
  cout << "+ Aluno(" << pessoa->obtemNome() << "," << matricula << ") criado..." << endl;
  #endif
}

Aluno::~Aluno(){
  #ifdef DEBUG
  cout << "- Aluno(" << pessoa->obtemNome() << "," << matricula << ") destruido..." << endl;
  #endif
  delete pessoa;
}

string Aluno::obtemNome() const {  return pessoa->obtemNome();  }
int Aluno::obtemMatricula() const {  return matricula;  }
void Aluno::defineNome(string n) {  pessoa->defineNome(n);  }
void Aluno::defineMatricula(int m){  matricula = m;  }

string Aluno::str() const {
  stringstream ss;
  ss << pessoa->obtemNome() << " (" << matricula << ")";
  return ss.str();
}
```

Exercício 2: Professor.hpp

```
#ifndef _PROFESSOR_HPP
#define _PROFESSOR_HPP

#include "Pessoa.hpp"

class Professor {
private:
  Pessoa *pessoa;
  int matricula;
  double salario;
public:
  Professor(string n = "", int m = 0, double s = 0.0);
  ~Professor();
  string obtemNome() const;
  int obtemMatricula() const;
  double obtemSalario() const;
  void defineNome(string n);
  void defineMatricula(int m);
  void defineSalario(double s);
  string str() const;
};
#endif
```

Exercício 2: Professor.cpp

```cpp
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include <iomanip>
#include"Professor.hpp"

Professor::Professor(string n, int m, double s) {
  pessoa = new Pessoa(n);
  matricula = m;
  salario = s;
  #ifdef DEBUG
  cout << "+ Professor(" << pessoa->obtemNome() << "," << matricula << "," << salario << ") criado..." << endl;
  #endif
}

Professor::~Professor() {
  #ifdef DEBUG
  cout << "- Professor(" << pessoa->obtemNome() << "," << matricula << "," << salario << ") destruido..." << endl;
  #endif
  delete pessoa;
}

string Professor::obtemNome() const {  return pessoa->obtemNome();  }
int Professor::obtemMatricula() const {  return matricula;  }
double Professor::obtemSalario() const {  return salario;  }
void Professor::defineNome(string n) {  pessoa->defineNome(n);  }
void Professor::defineMatricula(int m) {  matricula = m;  }
void Professor::defineSalario(double s) {  salario = s;  }

string Professor::str() const {
  stringstream ss;
  ss << pessoa->obtemNome() << " (" << matricula << "; R$";
  ss << fixed << setprecision(2) << salario << ")";
  return ss.str();
}
```

Exercício 2: Recurso.hpp

```
#ifndef _RECURSO_HPP
#define _RECURSO_HPP

#include<string>

using namespace std;

class Recurso{
private:
  string nome;
  int quantidade;
public:
  Recurso(string n="",int q=0);
  ~Recurso();
  string obtemNome() const;
  int obtemQuantidade() const;
  void defineNome(string n);
  void defineQuantidade(int q);
  string str() const;
};
#endif
```

Exercício 2: Recurso.cpp

```
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include"Recurso.hpp"

Recurso::Recurso(string n,int q) {
  nome = n;
  quantidade = q;
  #ifdef DEBUG
  cout << "+ Recurso(" << nome << "," << quantidade << ") criado..." << endl;
  #endif
}

Recurso::~Recurso() {
  #ifdef DEBUG
  cout << "- Recurso(" << nome << "," << quantidade << ") destruido..." << endl;
  #endif
}

string Recurso::obtemNome() const {  return nome;  }
int Recurso::obtemQuantidade() const {  return quantidade;  }
void Recurso::defineNome(string n) {  nome = n;  }
void Recurso::defineQuantidade(int q) {  quantidade = q;  }

string Recurso::str() const {
  stringstream ss;
  ss << nome << " (" << quantidade << ")";
  return ss.str();
}
```

Exercício 2: Turma.hpp

```
#ifndef _TURMA_HPP
#define _TURMA_HPP

#include "Professor.hpp"
#include "Aluno.hpp"
#include "Recurso.hpp"

#define MAX_ALUNOS   30
#define MAX_RECURSOS 5

using namespace std;

class Turma {
private:
    int codigo;
    string disciplina;
    Professor *professor;
    int numAlunos;     Aluno *alunos[MAX_ALUNOS];
    int numRecursos;   Recurso *recursos[MAX_RECURSOS];
public:
    Turma(int c = 0, string d = "");
    ~Turma();
    int obtemCodigo() const;
    string obtemDisciplina() const;
    Professor *obtemProfessor() const;
    int obtemNumAlunos() const;
    Aluno *obtemAluno(int i) const;
    int obtemNumRecursos() const;
    Recurso *obtemRecurso(int i) const;
    void defineCodigo(int c);
    void defineDisciplina(string d);
    void defineProfessor(Professor *p);
    bool incluiAluno(Aluno *a);
    bool incluiRecurso(Recurso *r);
    string str() const;
};
#endif
```

Exercício 2: Turma.cpp (primeira parte)

```
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "Turma.hpp"

Turma::Turma(int c, string d) {
  codigo = c;
  disciplina = d;
  professor = nullptr;
  numAlunos = numRecursos = 0;
  #ifdef DEBUG
  cout << "+ Turma(" << codigo << "," << disciplina << ") criada..." << endl;
  #endif
}

Turma::~Turma() {
  #ifdef DEBUG
  cout << "- Turma(" << codigo << "," << disciplina << ") destruida..." << endl;
  #endif
}

int Turma::obtemCodigo() const {  return codigo;  }
string Turma::obtemDisciplina() const {  return disciplina;  }
Professor *Turma::obtemProfessor() const {  return professor;  }
int Turma::obtemNumAlunos() const {  return numAlunos;  }
Aluno *Turma::obtemAluno(int i) const {  return (i<0 || i>=numAlunos)?nullptr:alunos[i];  }
int Turma::obtemNumRecursos() const {  return numRecursos;  }
Recurso *Turma::obtemRecurso(int i) const {  return (i<0 || i>=numRecursos)?nullptr:recursos[i];  }
void Turma::defineCodigo(int c) {   codigo = c;  }
void Turma::defineDisciplina(string d) {  disciplina = d;  }
void Turma::defineProfessor(Professor *p) {  professor = p;  }
```

Exercício 2: Turma.cpp (segunda parte)

```
bool Turma::incluiAluno(Aluno *a) {
  if (numAlunos >= MAX_ALUNOS) return false;
  alunos[numAlunos++] = a;
  return true;
}

bool Turma::incluiRecurso(Recurso *r) {
  if (numRecursos >= MAX_RECURSOS) return false;
  recursos[numRecursos++] = r;
  return true;
}

string Turma::str() const {
  stringstream ss;
  ss << "Turma: " << codigo << " - " << disciplina << endl;
  if (professor != nullptr) ss << "Professor: " << professor->str() << endl;
  if (numAlunos > 0) {
     ss << "Alunos:" << endl;
     for (int i=0; i< numAlunos; ++i)
         ss << "- " << alunos[i]->str() << endl;
  }
  if (numRecursos > 0) {
     ss << "Recursos:" << endl;
     for (int i=0; i< numRecursos; ++i)
         ss << "- " << recursos[i]->str() << endl;
  }
  return ss.str();
}
```

Exercício 2: main.cpp

```
#include <iostream>
#include "Turma.hpp"

using namespace std;

int main () {
  Aluno *alunos[] = {
    new Aluno("Aluno 1", 1), new Aluno("Aluno 2", 2), new Aluno("Aluno 3", 3), new Aluno("Aluno 4", 4)
  };
  int numAlunos = sizeof(alunos) / sizeof(Aluno *);
  Recurso *recursos[] = {
    new Recurso("Computador", 4), new Recurso("Projetor",1)
  };
  int numRecursos = sizeof(recursos) / sizeof(Recurso *);
  Professor *professor = new Professor("Professor 1", 234, 567.89);
  Turma *turma = new Turma(30,"Programacao Orientada a Objetos (ECo)");
  turma->defineProfessor(professor);
  for (int i=0; i<numAlunos; ++i)
      turma->incluiAluno(alunos [i]);
  for (int i=0; i<numRecursos; ++i)
      turma->incluiRecurso(recursos[i]);

  cout << turma->str();

  delete turma;
  delete professor;
  for (int i=0; i<numRecursos; ++i)
      delete recursos[i];
  for (int i=0; i<numAlunos; ++i)
      delete alunos[i];
  return 0;
}
```

Exercício 2: Makefile

```
CFLAGS = -c -std=c++11 #-DDEBUG

all:            main.o Pessoa.o Aluno.o Professor.o Recurso.o Turma.o
                g++ -o app main.o Pessoa.o Aluno.o Professor.o Recurso.o Turma.o
                ./app > app.out

main.o:         main.cpp Aluno.hpp Professor.hpp
                g++ ${CFLAGS} main.cpp

Pessoa.o:       Pessoa.cpp Pessoa.hpp
                g++ ${CFLAGS} Pessoa.cpp

Aluno.o:        Aluno.cpp Aluno.hpp Pessoa.hpp
                g++ ${CFLAGS} Aluno.cpp

Professor.o:    Professor.cpp Professor.hpp Pessoa.hpp
                g++ ${CFLAGS} Professor.cpp

Recursos.o:     Recurso.cpp Recurso.hpp
                g++ ${CFLAGS} Recurso.cpp

Turma.o:        Turma.cpp Turma.hpp Aluno.hpp Professor.hpp Recurso.hpp
                g++ ${CFLAGS} Turma.cpp

clean:
                rm -f main.o Pessoa.o Aluno.o Professor.o Recurso.o Turma.o app
```