INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e de Gestão

Departamento de Informática e Comunicações

Engenharia Informática / Informática de Gestão
2º Ano
Época Recurso

Programação Orientada por Objectos — Data: 12/02/2011
1º Semestre — Duração: 2h00

resolva os 2 grupos em folhas de exame separadas

Grupo I
(12 valores)

Uma empresa transportadora de grande dimensão necessita de uma pequena aplicação que lhe permita facilmente saber, a qualquer momento, que motorista é que conduz cada uma das viaturas que se encontrem em trânsito e qual o seu número de telemóvel (basta que a viatura esteja afecta a um motorista para que se considere que a mesma se encontra em trânsito). A empresa dispõe de uma frota considerável de viaturas, quer de passageiros, quer de mercadorias, sendo as primeiras caracterizadas pela matrícula e lotação, e as segundas pela matrícula e peso bruto, e derivado do estrangulamento financeiro que a afecta, todas as viaturas circulam com um único motorista.
Repare que, ainda que todos os motoristas estejam habilitados a conduzir qualquer viatura, num dado momento só uma viatura pode estar afecta a um motorista (teria que ser muito habilidoso para conseguir conduzir várias ao mesmo tempo…).

a) Apresente, através de um diagrama de classes em UML, a solução por si idealizada para sustentar a aplicação descrita, indicando para cada classe apenas os atributos estritamente necessários, um método construtor e, nas classes em que se justifique, o operador que permita que os objectos sejam coleccionáveis. [3 val.]
Não indique quaisquer outros métodos.

b) Codifique em C++ a sua solução, tal como a descreveu no diagrama da alínea anterior. [2.5 val.]

NOTA: As colecções deverão ser implementadas com base no template de classes *Coleccao* ou *ColeccaoHibrida* que utilizou nas aulas de POO. Para efeitos de consulta, disponibilizam-se os protótipos dos seus principais métodos no final do enunciado.

c) Implemente os métodos necessários para adicionar (registar) à aplicação um novo motorista e um novo veículo de passageiros. [1 val.]

d) Implemente o(s) método(s) necessário(s) para afectar uma viatura a um motorista. [2.7 val.]

e) Implemente o(s) método(s) que permita(m) obter o número de telemóvel do motorista que conduza uma viatura com uma dada matrícula. [1.4 val.]

f) Implemente o(s) método(s) necessário(s) para escrever na saída standard uma listagem das viaturas em circulação com indicação das matrículas e do nome dos respectivos condutores. [1.4 val.]

Grupo II
(8 valores)

O programa que se segue implementa um sistema de gestão (muito simples) de um prédio habitacional formado por habitações e garagens.

```
class Fraccao{
protected:
	char id;
	int permilagem;
public:
	Fraccao(char i, int p): id(i), permilagem(p){
		cout << "Criada Fraccao " << id;
	}
	virtual void mostrar(){
		cout << id << " com permilagem "<< permilagem << endl;
	}
	virtual ~Fraccao(){}
};

class Habitacao: public Fraccao{
public:
	Habitacao(char i): Fraccao(i, 40){
		cout << " do tipo Habitacao" << endl;
	}
	void mostrar(){
		cout << "Habitacao ";
		Fraccao::mostrar();
	}
	~Habitacao(){ cout << "Removida Habitacao "<< id << endl;}
};

class Garagem: public Fraccao{
public:
	Garagem(char i): Fraccao(i, 12){
		cout << " do tipo Garagem" << endl;
	}
	void mostrar(){
		cout << "Garagem ";
		Fraccao::mostrar();
	}
	~Garagem(){ cout << "Removida Garagem "<< id <<endl;}
};
```

```
class Predio{
	static const int MAXFRAC=20;
	Fraccao *fraccoes[MAXFRAC];
	int nfrac;
public:
	Predio():nfrac(0){
		cout << "Criado Predio." << endl;
	}
	void addGaragem(char i){
		if(nfrac<MAXFRAC) fraccoes[nfrac++]=new Garagem(i);
	}
	void addHabitacao(char i){
		if(nfrac<MAXFRAC) fraccoes[nfrac++]=new Habitacao(i);
	}
	void mostrarFraccoes(){
		cout << endl << "Fraccoes do Predio:" << endl;
		for(int i=0; i<nfrac; i++) fraccoes[i]->mostrar();
	}
	~Predio(){}
};

void main(){
	Predio p;
	p.addHabitacao('A');
	p.addGaragem('K');
	p.addHabitacao('B');
	p.mostrarFraccoes();
}
```

a) Identifique o tipo de relação existente entre `Predio` e `Habitacao`, entre `Fraccao` e `Garagem`, e entre `Predio` e `Fraccao`. [0.6 val.]

b) Alguma das classes é abstracta? [0.2 val.]

c) Numa das classes o destrutor não tem a implementação mais adequada. Diga qual é esse destrutor e dê-lhe a implementação desejada. [1.6 val.]

d) Apresente o resultado que será visualizado na saída standard após a execução do programa e depois de resolvida a inconformidade referenciada na alínea anterior.[2.6]

e) E o que seria visualizado se o método `Fraccao::Mostrar()` não fosse virtual? Refira-se apenas à parte da visualização que resultaria diferente em relação ao output da alínea anterior. [1 val.]

f) Admita agora que o prédio passou a poder incorporar, para além de habitações e garagens, lojas comerciais com diferentes permilagens. Acrescente ao programa os métodos e/ou classes necessárias, para que passe a reflectir esta nova realidade. [2 v.]

Anexo

Class Coleccao

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

template<class K>
class Coleccao: public set<K>{
public:
	bool insert(const K &c);
	K *find(const K &c);
	int size() const;
	void erase(const K &);
	//void clear();
	//bool empty() const;
	//iterator begin();
	//iterator end();
};
```

Class ColeccaoHibrida

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

template<class T>
class less_pointers
{
public:
	bool operator()(const T &left, const T &right) const
	{
		return (*left < *right);
	}
};

template<class K>
class ColeccaoHibrida: public set<K, less_pointers<K>>
{
public:
	bool insert(const K &c);
	K find(const K &c);
	int size() const;
	void erase(const K &);
	//void clear();
	//bool empty() const;
	//iterator begin();
	//iterator end();
};
```