Tarifario.h

```
#ifndef TARIFARIO_H
#define TARIFARIO_H

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

// Alinea A
class Tarifario {
	unsigned int * treinos;
	unsigned int quantos;
	virtual unsigned int calculaPagamento()const = 0;  // a implementar pelas derivadas
	 // esta funcao apenas se destina a calcular o pagamento
	 // a funcao calculaPagamentoEApagaTreinos() invoca esta e apaga os treinos
public:
	void acrescentaTreino(unsigned int t);
	unsigned int calculaPagamentoEApagaTreinos();

	// ---- funcoes seguintes: nao mencionadas, obrigatorias na mesma

	// e preciso um construtor para inicializar o ponteiro e a variavel quantos
	Tarifario();

	// Vai ser preciso ter o destrutor para devolver a memoria ocupada pela matriz dinamica
	virtual ~Tarifario();

	// a matriz dinamica de inteiros (treinos) pertence exclusivamente a cada objecto
	// de Tarifario, mas nao e automaticamente copiada quando se copiam ou atribuem estes
	// objectos (apenas o ponteiro e copiado levando a partilha de matriz entre objectos
	// e mais tarde a deletes repetidos da mesma matriz, entre outras incoerencias)
	// => e preciso fazer o operador de atribuicao e o construtor por copia:

	Tarifario(const Tarifario & x);
	Tarifario & operator=(const Tarifario & x);

	// Funcoes que nao foram pedidas mas vao dar jeito ao resto do codigo
	unsigned int getNumTreinos() const;

	unsigned int getTreino(unsigned int i) const;

	virtual Tarifario * duplica()const = 0;  // tem que ser abstracta tb. Nao se pode construir
	                        // um Tarifario ainda porque e classe abstracta
};

#endif
```

Tarifario.cpp

```
#include "Tarifario.h"

Tarifario::Tarifario(){
	treinos = nullptr; // inicialmente nao ter treinos nenhuns
	quantos = 0;
}
Tarifario::~Tarifario() {
		delete [] treinos;
}
```

```
void Tarifario::acrescentaTreino(unsigned int t) {
	unsigned int * aux = new unsigned int [quantos + 1];
	if (aux == nullptr) return;
	for (unsigned int i=0; i< quantos; i++)
		aux[i] = treinos[i];
	aux[quantos] = t;
	quantos++;
	delete []treinos;
	treinos = aux;
}
unsigned int Tarifario::calculaPagamentoEApagaTreinos() {
	unsigned int conta = calculaPagamento();  // vai chamar a funcao correspondente
// a classe do objecto que invoca
	delete []treinos;
	treinos = nullptr;
	quantos = 0;
	return conta;
}
Tarifario::Tarifario(const Tarifario & x) {
	// 1.  Preinicializacao tipicamente necessaria
	treinos = nullptr;  // pre-inicializacao para que o operador de atribuicao
	quantos = 0;        // consiga trabalhar bem (precisa de ter o objecto num estado
                        // coerente, dai a inicializacao previa)
                          // para acontecer delete apenas a um valor de um ponteiro
                          // nulo ou com um valor dado por new ...
	// 2. Usar ooperador de atribuicao
	*this = x;
}

Tarifario & Tarifario::operator=(const Tarifario & x) {
	// 1. Testa auto-atribuicao para evitar trabalho desnecessario e possivel incoerencia
	//    e libertacao dos treinos da origem da copia (que coincide com o destino)
	if (this == & x) // na mem. dinamica. Se sao o mesmo -> ja sao iguais = concluido
		return *this;

	// 2. Devolve os recursos ocupados pelo objecto alvo da atribuicao.
	delete []treinos;
	treinos = nullptr; // coloca objecto ja num estado coerente com o significado
	quantos = 0;       // "tenho 0 treinos"

	// 3. Se o outro objecto tambem tem 0 treinos, ja estamos iguais. Concluido
	if (x.treinos == nullptr) return *this;

	// 4. Criar recursos iguais (copiar) aos do outro objecto
	// 4.1. Alocar matriz dinamica para tantos treinos como os do outro objecto
	// 4.2. Copiar dos dados: o conteudo da matriz e restantes membros
	treinos = new unsigned int[x.quantos];
	if (treinos == nullptr)
		return *this;      // se nao houve memoria fica com 0 treinos
	quantos = x.quantos;
	for (unsigned int i = 0; i<quantos; i++)
		treinos[i] = x.treinos[i];
	// 5. concluido
	return *this;
}
unsigned int Tarifario::getTreino(unsigned int i) const {
	if ( (i<0) || (i>=quantos) ){
		return 0;
	 }
	return treinos[i];
}
unsigned int Tarifario::getNumTreinos() const { return quantos; }
```

Apressado.h

```
#ifndef APRESSADO_H
#define APRESSADO_H

#include "tarifario.h"
// Alinea B


// Apenas e preciso redefinir a funcao calcula pagamentos
// O acesso aos treinos tem que ser feito via funcoes getXXX pois os dados sao
// privados na classe base.

class Apressado : public Tarifario {
	unsigned int calculaPagamento()const;
public:
	Tarifario * duplica()const;
};

#endif
```

Apressado.cpp

```
#include "Apressado.h"

unsigned int Apressado::calculaPagamento()const {
	unsigned int custo = 0;
	unsigned int numTreinos = getNumTreinos();

	for(unsigned int i=0; i< numTreinos; i++) {
		unsigned int treino = getTreino(i);
		if (treino <= 10)
			custo += 10;
		else if (treino <= 20)
			custo += 15;
		else
			custo += 25;
	}
	return custo;
}
Tarifario * Apressado::duplica() const{
	return new Apressado(*this);
}
```

Cliente.h

```
#ifndef CLIENTE_H
#define CLIENTE_H

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

// Alinea C
class Tarifario;
class Ginasio;

class Cliente {
	string nome;
	unsigned int bi;
	Tarifario * tarif;
```

```
        unsigned int horainicio;// hora de inicio do treino
         // -1 se o cliente nao estiver a treinar

        //     Faz parte da alinea C original
    //se fosse proibido copiar e atribuir objectos da classe Cliente
        //Cliente(const Cliente & x) = delete;
        //Cliente & operator= (const Cliente & x) = delete;

public:
        // Construtor e necessario. Deve receber informacao do tarifario
    // Nao pode haver clientes sem tarifario
        Cliente(string n,unsigned int b, Tarifario * t);

        // e necessario um destrutor para apagar o Tarifario
        virtual ~Cliente();

        void iniciaTreino(unsigned int hora);
        void terminaTreino(unsigned int hora);
        unsigned int paga();

        virtual void reageEntrada(Ginasio * g) = 0;
        virtual void reageSaida(Ginasio * g) = 0;

        void mudaTarifario(Tarifario * t);

    // Necessarias mas nao foram explicitamente pedidas

        unsigned int getBI()const { return bi; }

        bool estaATreinar() const{ return horainicio != -1; }

        // Alinea D - "Afinal e para copiar"

        //  Passa a ser necessario ter operador de atribuicao
        // e construtor por copia porque o Tarifario pertence ao cliente e tem que ser
        // tambem copiado mas como reside fora do objecto cliente (via ponteiro) so sera
        // copiado se for atraves de um construtor por copia e operador de atribuicao
        // definidos pelo progamador

        // => Fazer: construtor por copia e operador de atribuicao

        Cliente & operator= (const Cliente & x);

        Cliente(const Cliente & x);

        virtual Cliente * duplica()const = 0;  // Tem que ser virtual pura (Cliente e abstracta)
        virtual string getAsString() const;
};


#endif
```

Cliente.cpp

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

#include "Cliente.h"
#include "Tarifario.h"
```

```
Cliente::~Cliente() {
    delete tarif;
}

Cliente::Cliente(string n, unsigned int b, Tarifario * t) : nome(n), bi(b), tarif(t) {
    horainicio = -1; // significa nao esta a treinar - necessario
}

void Cliente::iniciaTreino(unsigned int hora) {
    if (horainicio == -1)
        horainicio = hora;
}

void Cliente::terminaTreino(unsigned int hora) {
    if (horainicio != -1) {
        tarif->acrescentaTreino(hora - horainicio);
        horainicio = -1;
    }
}

unsigned int Cliente::paga() {
    return tarif->calculaPagamentoEApagaTreinos();
}

void Cliente::mudaTarifario(Tarifario * t) {
    if (t != nullptr) { // So muda se o novo tarifario existir mesmo
        delete tarif; // Apaga tarifario anterior (deixa de servir)
        tarif = t; // (obs: os treinos do antigo tarifario ficaram
    } // por pagar.... devia-se pagar antes de mudar de tarif)
}

Cliente & Cliente::operator=(const Cliente & x) {

    // 1. Testa auto-atribuicao para evitar trabalho desnecessario
    // e libertacao do tarifario da origem da copia (que coincide com o destino)
    if (this == &x) return * this;

    // 2. Liberta recursos actuais do objecto alvo da atribuicao
    //    Neste caso so o objecto tarifario
    delete tarif;

    // 3. Cria/Copia recursos (iguais ao) do outro objeto
    // 3.1 No caso do tarifario, pode ser de qualquer tipo
    //     Usar o new nao e viavel (nao se sabe o tipo). O Tarifario que se duplique
    //     Tem que ser pela via do "duplica" (implementar no tarifario e derivadas)
    nome = x.nome;
    bi = x.bi;
    horainicio = x.horainicio;
    tarif = x.tarif->duplica();

    return *this;
}

Cliente::Cliente(const Cliente & x) {
    // 1. Efectua pre-inicializacao para compatibilizar com o operador
    //    neste caso basta colocar o ponteiro tarif num estado inicial coerente
    tarif = nullptr;

    // 2. Usa operador de atribuicao
    *this = x;
}
```

```
string Cliente::getAsString() const {
    ostringstream oss;
    oss << nome << " - BI: " << bi;
    return oss.str();
}
```

Sociavel.h

```
#ifndef SOCIAVEL_H
#define SOCIAVEL_H

#include "cliente.h"
// Alinea E

class Sociavel : public Cliente {
public:
	Sociavel(string n,unsigned int b, Tarifario * t);
	virtual void reageEntrada(Ginasio * g) {}
	virtual void reageSaida(Ginasio * g);

	Cliente * duplica()const {
		return new Sociavel(*this);
	}
	virtual string getAsString()const;
};

#endif
```

Sociavel.cpp

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

#include "Sociavel.h"
#include "Ginasio.h"

Sociavel::Sociavel(string n,unsigned int b, Tarifario * t) : Cliente(n, b,  t) {
}
void Sociavel::reageSaida(Ginasio * g) {
	if (g->getNumClientesATreinar() == 1) // se so sobrou este
		g->saiClienteDoTreino(getBI());  // pede para sair
}
string Sociavel::getAsString() const {
	ostringstream oss;
	oss << "Sociavel - " << Cliente::getAsString();
	return oss.str();
}
```

Ginasio.h

```
#ifndef GINASIO_H
#define GINASIO_H

#include <iostream>
#include <string>
#include <vector>
```

```
using namespace std;


class Cliente;
// Alinea F

class Ginasio {
	vector<Cliente *> clientes; // vector - ninguem proibiu. * porque e preciso p/ polimorf.
	unsigned int relogio;

    unsigned int pesquisaClienteDadoBI(unsigned int bi);
public:
	void avancaRelogio(unsigned int tempo);

	bool acrescentaCliente(Cliente * c);

	unsigned int paga(unsigned int bi);

	void entraClienteNoTreino(unsigned int bi);

	void saiClienteDoTreino(unsigned int bi);

	void removeCliente(unsigned int bi);

	// funcoes que nao foram explicitamente pedidas mas sao mesmo necessarias
	Ginasio() {
		relogio = 0;
	}

	~Ginasio();
	// Nao foram pedidas, fazem falta ou dao jeito

	unsigned int getNumClientesATreinar() const;

	// Alinea G - Duplicar o Ginasio e todos os dados que contem (clientes e os seus tarifarios)
	Ginasio & operator=(const Ginasio & x);

	Ginasio(const Ginasio & x);
	string getAsString() const;
};

#endif
```

Ginasio.cpp

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

#include "Ginasio.h"
#include "Cliente.h"

void Ginasio::avancaRelogio(unsigned int tempo) {
    relogio += tempo;
}
```

```
bool Ginasio::acrescentaCliente(Cliente * c) {
    if (c == nullptr) {
        return false;
    }
    // se existe um cliente com o mesmo bi
    if (pesquisaClienteDadoBI(c->getBI()) != -1) {
        return false;
    }
    clientes.push_back(c);
    return true;
}

unsigned int Ginasio::paga(unsigned int bi) {
    // Procura o cliente. E devolvida a sua posicao no vector clientes
    unsigned int ind = pesquisaClienteDadoBI(bi);
    // Se nao foi encontrado foi devolvido -1
    if (ind != -1) {
        // Se encontrou, chama a paga()
        return clientes[ind]->paga();
    }
    return 0;
}

void Ginasio::entraClienteNoTreino(unsigned int bi) {
    unsigned int ind = pesquisaClienteDadoBI(bi);

    if (ind == -1) return; // pode acontecer ser um Cliente "nao encontrado"

    // 1. Verifica se o cliente ainda nao estava a treinar
    if (clientes[ind]->estaATreinar() == true) {
        return;
    }

    // 2. Avisa os outros clientes a treinar que ENTROU um na sala
    for (unsigned int i = 0; i < clientes.size(); i++) {
        if (clientes[i]->estaATreinar()){
            clientes[i]->reageEntrada(this);
        }
    }
    // o cliente inicia o treino
    clientes[ind]->iniciaTreino(relogio);
}

void Ginasio::saiClienteDoTreino(unsigned int bi) {
      // testa se encontrou
    unsigned int ind = pesquisaClienteDadoBI(bi);
    if (ind == -1) return; // pode acontecer ser um Cliente "nao encontrado"
    // 2. Nao estava a treinar? Entao esquece. Foi engano
    if (clientes[ind]->estaATreinar() == false) {
        return;
    }

    // primeiro tira da sala de treino e so depois avisa os que ficaram que saiu um
    clientes[ind]->terminaTreino(relogio);

    // 4. Avisa os outros clientes a treinar que saiu um da sala
    for (unsigned int i = 0; i < clientes.size(); i++) {
        if (clientes[i]->estaATreinar()) {
            clientes[i]->reageSaida(this);
        }
    }
}
```

```
void Ginasio::removeCliente(unsigned int bi) {
    // 1. procura o cliente na base de dados de clientes
    unsigned int ind = pesquisaClienteDadoBI(bi);
    if (ind == -1) return; // Nao existe aqui ninguem com esse BI

    // 2. Encontrou - manda-o sair do treino (pode nem estar a treinar = faz nada)
    saiClienteDoTreino(bi); // funcao de ginasio - evita-se repetir o codigo aqui

    // 3. Ja agora, antes de ires, paga a conta sff.
    clientes[ind]->paga(); // deseja num. de contribuinte na factura?

    // 5. Apaga (a ficha do) cliente - O enunciado diz que os clientes pertencem ao ginasio
    delete clientes[ind]; // importante que o cliente seja mesmo uma variavel dinamica
    clientes.erase(clientes.begin() + ind); // retira do vector o ponteiro para memoria ja libertada
}

Ginasio::~Ginasio() {
    // Apaga as fichas. Nao e preciso fazer mais nada.
    for (unsigned int i = 0; i < clientes.size(); i++)
        delete clientes[i]; // Enunciado diz que clientes pertencem ao Ginasio
} // (devem ser meras fichas de papel num arquivo qualquer)

// Nao foram pedidas, fazem falta ou dao jeito

unsigned int Ginasio::getNumClientesATreinar() const {
    unsigned int n = 0;
    for (unsigned int i = 0; i < clientes.size(); i++) {
        if (clientes[i]->estaATreinar())
            n++;
    }
    return n;
}

unsigned int Ginasio::pesquisaClienteDadoBI(unsigned int bi) {
    for (unsigned int i = 0; i < clientes.size(); i++)
        if (clientes[i]->getBI() == bi)
            return i;
    return -1;
} // retorna o indice se encontrou, ou -1 se nao encontrou

Ginasio & Ginasio::operator=(const Ginasio & x) {

    // 1. Testa a auto-atribuicao. Se sao o mesmo objecto, "ja sao iguais"
    // nao se libertam recursos do destino da atribuicao porque coincide com a origem
    if (this == &x) return *this;

    // 2. Apaga o conteudo ja existente no "alvo" da atibuicao (*this)
    for (unsigned int i = 0; i < clientes.size(); i++)
        delete clientes[i]; // apaga mesmo os clientes (eram do ginasio)
    // os clientes foram apagados, mas os ponteiros
    clientes.clear(); // ainda estao no vector, o qual deve ser esvaziado.

    // 3. Faz copia polimorfica dos clientes do Ginasio original
    for (unsigned int i = 0; i < x.clientes.size(); i++)
        clientes.push_back(x.clientes[i]->duplica()); // Seja la o que fores, duplica-te

    // 4. So falta copiar o valor do relogio
    relogio = x.relogio;
    return *this;
}
```

```cpp
Ginasio::Ginasio(const Ginasio & x) {
    // nao e preciso nenhuma pre-inicializacao explicita porque o vector e
    // inicializado automaticamente com o seu construtor por omissao (uma vez que nao
    // foi usado nenhum outro na lista de inicializaao deste constutor) e que o deixa
    // vazio e pronto a usar. Noutros casos poderia ser necesaria algo aqui.
    *this = x;
}

string Ginasio::getAsString() const {
    ostringstream oss;
    oss << "\nGinasio: "
            << " (tempo: " << relogio << ")"
            << "\nClientes" << endl;
    for (unsigned int i = 0; i < clientes.size(); i++)
        oss << clientes[i]->getAsString() << endl;

    oss << "\nA treinar " << endl;
    for (unsigned int i = 0; i < clientes.size(); i++) {
        if (clientes[i]->estaATreinar()) {
            oss << clientes[i]->getAsString() << endl;
        }
    }
    return oss.str();
}
```