## Um exercício (herança, exceções, ...)

- Considere um mercado onde são comercializados dois tipos de produtos:
  - produtos sem necessidade especial de conservação ou validade (classe Produto)
  - produtos que exigem uma determinada temperatura de conservação e validade (classe **ProdutoCons**)
- · Observações:
  - Existem clientes e fornecedores (classe Participante), caracterizados por:
    - · um código
    - o conjunto de produtos de interesse (classe **ProdutoMercado**)
  - Um produto de interesse para um participante é composto por:
    - produto que este quer comprar (ou vender)
    - quantidade
    - preço



AED = 2019/20

|•••••

```
Um exercício (herança, exceções, ...)
O ficheiro mercado.txt tem informação inicial sobre o mercado: participantes e produtos.
      A informação contida no ficheiro é a seguinte:
                   // string "cliente" ou "fornecedor" ou "produto" ou "produtoCons"
       //*** se tipo= "cliente" ou "fornecedor", as linhas seguintes são:

    codigo_participante // string

    codigo_produto // inteiro

    quantidade

                                // quanto está disposto a comprar ou vender
      preco
                                 // float: preço máximo de compra se cliente, preço de venda se fornecedor
         codigo_produto
          quantidade
          preco
       //*** se tipo = "produto", a linha seguinte é:
          codigo_produto
                                  // inteiro
      //**** se tipo="produtoCons", as linhas seguintes são:
          codigo_produto
                                  // inteiro
                                  // inteiro
          temperatura_cons
          validade
                                  // inteiro
```

## Um exercício (herança, exceções, ...)

- · Considere as classes:
  - Mercado, Participante, Produto (subclasse: ProdutoCons), ProdutoMercado, ProdutoNaoDisp
- Implemente na classe **Mercado** o método:
  - void inicia(string nomeFich)

lê o ficheiro *nomeFich* e preenche a informação de produtos e participantes no mercado.

- Implemente na classe Mercado o método:
  - Fornecedor \*precoMaisBaixo(int codProduto, int quant)
     retorna o fornecedor que possui preço mais baixo para o produto de código codProduto. Este deve possuir quant quantidades desse produto. Se não existir nenhum fornecedor nestas condições lança uma exceção do tipo ProdutoNaoDisp.
- No período de promoções, os produtos normais (Produto) sofrem uma redução de 2% no preço e os produtos com validade (ProdutoCons) sofrem uma redução de 2% se a validade é superior a 20 e 5% se a validade é inferior ou igual a 20. Todos os fornecedores devem efetuar esta redução nos preços. Implemente na classe Mercado o método:
  - void promocao()



NED = 2019/20

```
Um exercício (herança, exceções, ...)
```

```
class Produto {
    int codigo;
public:
    Produto(int cod);
    int getCodigo();
    virtuaDint valorPromocao();
    virtuaDvoid imprime();
};
int Produto::valorPromocao()
```

```
int Produto::valorPromocao(
{ return 2; }

void Produto::imprime()
{ cout << codigo << endl; }</pre>
```



70 2010/20

```
class ProdutoCons: public Produto {
    int temptCons, validade;
    public:
        ProdutoCons(int cod, int tc, int val);
        int valorPromocao();
        void imprime();
    };

int ProdutoCons::valorPromocao()
{ if (validade<20) return 5; else return 2; }

void ProdutoCons::imprime()
{ cout << getCodigo() << " : temp=" << temptCons << " : val=" << validade << endl; }

AED-201920
```

```
Um exercício (herança, exceções, ...)
class ProdutoMercado {
   Produto *produto;
   int quantidade;
   float preco;
public:
   ProdutoMercado(Produto *prod, int qt=0, float p=0);
   Produto *getProduto() const;
   int getQuantidade() const;
   float getPreco() const;
   float setPreco(float p) ;
   void imprime();
};
  void ProdutoMercado::imprime() {
     cout << produto->getCodigo() << ": quant= " << quantidade << " , preco= " <<
      preco << endl;
```

## Class Participante { string codigo; vector<ProdutoMercado> produtos; public: Participante(int cod); string getCodigo() const; float getPreco(int codProduto, int quant) const; void addProduto(ProdutoMercado &prod); void imprime() const; void promocao(); };

```
void Participante::promocao() {
    for (unsigned int i=0; iprodutos.size(); i++)
    {
        Produto * prod = produtos[i].getCodigo();
        int decPreco=prod->valorPromocao();
        float preco= produtos[i].getPreco()*(1.0-decPreco/100.0);
        produtos[i].setPreco(preco);
    }
}

void Participante::addProduto(ProdutoMercado &prod) {
        produtos.push_back(prod);
}
```

```
### Cout <= ""; produtos.ize(); i++) {

| cout <= ""; produtos.ij.imprime(); }

| AED = 2019/20
```

```
class ProdutoNaoDisp {
    string razao;
    int valor;
    public:
        ProdutoNaoDisp(string rz, int v);
        string getRazao() const;
        friend ostream & operator << (ostream &out, ProdutoNaoDisp &nd);
    };

ProdutoNaoDisp::ProdutoNaoDisp(string rz, int v): razao(rz), valor(v) { }
    string ProdutoNaoDisp::getRazao() const { return razao; }
        ostream & operator << (ostream &out, ProdutoNaoDisp &nd) {
            out << nd.razao << ":" << nd.valor << endl;
            return out;
    }
    FEUP

AED-2019/20
```

```
class Mercado
{
    vector<Participante> fornecedores;
    vector<Participante> clientes;
    static vector<Produto*> produtos;
    public:
    void inicia(string nomeF);
    void imprime();
    Participante *precoMaisBaixo(int codProduto, int quant);
    void promocao();
    static Produto* getProduto(int codProduto);
};

polimorfismo
```

```
Produto* Mercado::getProduto(int codProduto) {
    for (int i=0; iprodutos.size(); i++)
        if ( produtos[i]->getCodigo() == codProduto )
            return produtos[i];
    return NULL;
    }

void Mercado::promocao() {
    for (int i=0; i<fornecedores.size(); i++)
        fornecedores[i].promocao();
    }

FEUP

AED=2019/20
</pre>
```

```
Participante *Mercado::precoMaisBaixo(int codProduto, int quant) {
    float pmin=0;
    int indF=-1;
    ProdutoNaoDisp p1("produto inexistente", codProduto);
    for (int i=0; i<fornecedores.size(); i++)
    {
        try{
            float p = fornecedores[i].getPreco(codProduto, quant);
            if (pmin == 0) { pmin=p; indF=i; }
            else if (p < pmin) { pmin=p; indF=i; }
            catch(ProdutoNaoDisp nd) {
                if (ind.getRazao()=="quantidade insuficiente") p1=nd; }
            }
            if (indF != -1) return &fornecedores[indF];
            else throw p1;
        }
            FEUP
```

```
Um exercício (herança, exceções, ...)
 void Mercado::inicia(string nomeF) {
    ifstream fich(nomeF.c_str()); string linha;
    Participante *p;
    while ( !fich.eof() ) {
        fich >> linha; int cod=atoi(linha.c_str());
        if (cod==0) { // linha contém uma string
             if (linha=="cliente") {
             fich >> linha;
             Participante p1(linha);
             clientes.push_back(p1);
             p=&clientes.back();
          else if (linha=="fornecedor") {
             fich >> linha;
             Participante p1(linha);
             fornecedores.push_back(p1);
             p=&fornecedores.back();
```

```
Um exercício (herança, exceções, ...)
            else if (linha=="produto") {
                fich >> cod;
               Produto *prod1= new Produto(cod);
               produtos.push_back(prod1);
            else if (linha=="produtoCons") {
               int tempt, val;
               fich >> cod; fich >> tempt; fich >> val;
               Produto *prod1= new ProdutoCons(cod,tempt,val);
               produtos.push_back(prod1);
          else {
                    // linha contem codigo produto
              int quant; float preco;
              fich >> quant; fich >> preco;
              Produto *prod1=Mercado::getProduto(cod);
              ProdutoMercado prod1(cod,quant,preco);
              p->addProduto(prod1);
       } fich.close(); }
```

```
void Mercado::imprime()
{
    cout << "clientes:" << endl;
    for (int i=0; i<clientes.size(); i++)
        clientes[i].imprime();
    cout << endl << "fornecedores:" << endl;
    for (int i=0; i<fornecedores.size(); i++)
        fornecedores[i].imprime();
    cout << "produtos:" << endl;
    for(int i=0; i<produtos.size(); i++)
        produtos[i]->imprime();
    cout << endl;
}
</pre>
```

```
Um exercício (herança, exceções, ...)
    int main()
      Mercado m1; char op;
      ml.inicia("mercado.txt");
      do {
        cout << "i - imprime informação mercado" << endl;
        cout << "f - pesquisa fornecedor mais barato" << endl;</pre>
        cout << "p - promoção" << endl; cout << "s - sair" << endl;
        cin >>op;
        switch (op) {
          case 'i':
             m1.imprime(); break;
          case 'p':
             m1.promocao();
             cout << endl << "promocao:" << endl << endl;</pre>
             m1.imprime(); break;
```

```
case 'f':

try {

int codP, qt;

cout << "qual o código do produto? "; cin >> codP;

cout << "qual a quantidade? "; cin >> qt;

Participante *f1=m1.precoMaisBaixo(codP,qt);

cout << "fornecedor com preco mais baixo para " << qt;

cout << "fornecedor com preco mais baixo para " << qt;

cout << " unidades do artigo " << codP << " : " << f1->getCodigo() << endl;

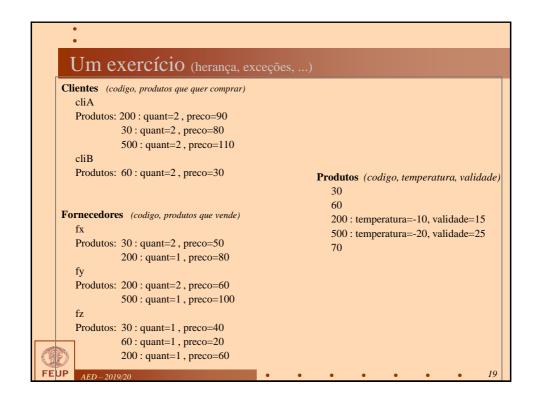
}

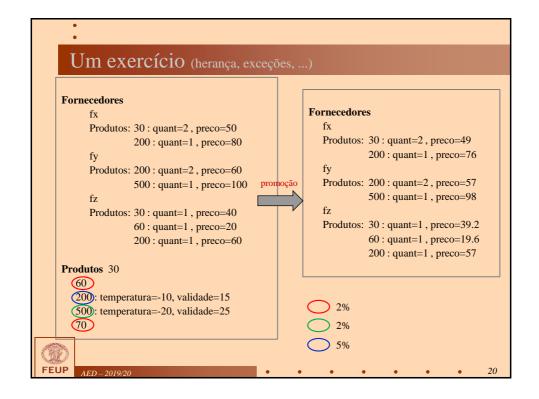
catch (ProdutoNaoDisp &nd) { cout << nd ; }

break;

}
} while (op!='s');

return 1;
```





## Outro exercício: especificação

- Um programa de gestão de um clube desportivo guarda informação sobre o seu <u>pessoal</u>, que inclui *atletas*, *professores* e *sócios*.
- Todo o pessoal do clube desportivo é identificado por um código único e sequencial.
  - Os atletas estão inscritos em uma modalidade, e em uma turma.
  - Os professores lecionam uma determinada modalidade a várias turmas.
  - Os sócios pagam uma determinada quota mensal.
  - A turma deve especificar o seu horário semanal.

Implemente a hierarquia de classes que na sua opinião melhor descreve o cenário acima, especificando os dados e métodos de cada classe.



D\_2019/20