**Requisitos funcionais**

***Todos os atributos e funções membros devem estar relacionados a classe***

1. Pelo menos 4 atributos

Em Picareta.h

int tipoN;

string tipoS;

Bloco matMine;

Em Ferramenta.h

int tipoFerramenta;

1. Pelo menos 4 funções membros sem incluir get e set

Em Ferramenta.h

virtual void jogarNoChao();

virtual void checarEstado() const;

virtual inline void destruir();

virtual bool quebrarBloco(Bloco &);

**Requisitos de implementação**

1. Todos os atributos devem ser inicializados. Fez validação de dados

Atributos da classe Bloco:

string nome;

vector<float> resMat;

vector<bool> colherMat;

Inicialização dos atributos no construtor:

Bloco::Bloco(string nome, float resMat[5], bool colherMat[5]) {

this->setNome(nome);

this->setResMat(resMat);

this->setColherMat(colherMat);

}

Atributos da classe Ferramenta:

int tipoFerramenta;

int maxRes;

int atualRes;

Inicialização dos atributos no construtor:

Ferramenta::Ferramenta(int tipoFerramenta, int maxRes, int atualRes, Data & dataDeCriacao) {

this->tipoFerramenta = tipoFerramenta;

this->setMaxRes(maxRes);

this->setAtualRes(atualRes);

this->dataDeCriacao = new Data(dataDeCriacao);

}

Atributos da classe Picareta:

int tipoN;

string tipoS;

Bloco matMine;

static int numPicaretasQuebradas;

1. Três construtores, incluindo um construtor de cópia e construtor com parâmetros defaults. Verifica alocação dentro do construtor de cópia.

Três construtores para Picareta:

Picareta(string = "madeira", int = 0, int = 60, int = 60, Bloco = Bloco());

Picareta(const Picareta &);

Picareta(int);

1. Deve ter um atributo string

Atributo string em Picareta.h:

string tipoS;

1. Um atributo static. Correta modelagem dos statics?

Inicialização do atributo static(Na classe Picareta):

int Picareta::numPicaretasQuebradas = 0;

Utilização do atributo static:

virtual inline void destruir(){

numPicaretasQuebradas++;

cout << "Sua Picareta de " << this->tipoS << " quebrou." << endl;

cout << "Construa uma picareta Nova" << endl;

}

1. Um atributo const static

Atributo const static em Ferramenta.h:

const static int TIPO\_PICARETA = 0;

1. Dois métodos constantes (não pode ser get)

Na classe Picareta:

void Picareta::checarEstado() const {

cout << "Sua picareta de " << this->tipoS << " esta com " << (this->atualRes / this->maxRes) \* 100 << "% de resistencia ";

cout << "(" << this->atualRes << " de " << this->maxRes << ")" << endl;

}

Na classe Ferramenta:

void Ferramenta::infoItem() const{

cout << "Este Item e uma ferramenta:" << endl;

}

E sobrecarregado na classe Picareta:

void Picareta::infoItem() const{

Ferramenta::infoItem();

cout << "Picareta de " << this->tipoS << endl;

}

1. Um array
2. Uma função inline (não pode ser get ou set)

Método inline em Picareta.h:

virtual inline void destruir() {

numPicaretasQuebradas++;

cout << "Sua Picareta de " << this->tipoS << " quebrou." << endl;

cout << "Construa uma picareta Nova" << endl;

}

1. Método com passagem por referência usando ponteiro

Método da classe Bloco:

bool Bloco::quebrarBloco(const Ferramenta \* f) {

int tipo = f->getTipoN();

cout << resMat[tipo] << "sec para quebrar o Bloco" << endl << "Para cancelar aperte C" << endl;

int oldTime;

for(int i = 0; i < resMat[tipo] \* 1000; i++)

{

Sleep(0.1);

if(kbhit())

if(getch() == 'c')

{

cout << "Bloco nao foi quebrado" << endl;

return false;

}

int time = i / 1000;

if(time != oldTime) cout << time + 1 << endl;

oldTime = time;

}

cout << "Bloco quebrado" << endl;

return this->colherMat[tipo];

}

1. Método static – deve ser chamado no main

Método static em Picareta:

int Picareta::menuPicareta(){

int opcao = -1;

cout << "Menu de criação de Picareta" << endl;

cout << "Voce deseja criar uma picareta de qual material?" << endl;

cout << "1 - Madeira" << endl;

cout << "2 - Pedra" << endl;

cout << "3 - Ferro" << endl;

cout << "4 - Diamante" << endl;

cout << "5 - Ouro" << endl;

cin >> opcao;

while(opcao > 5 || opcao < 1) {

cout << "Por favor, escolha uma opcao valida" << endl;

cout << "Menu de criação de Picareta" << endl;

cout << "Voce deseja criar uma picareta de qual material?" << endl;

cout << "1 - Madeira" << endl;

cout << "2 - Pedra" << endl;

cout << "3 - Ferro" << endl;

cout << "4 - Diamante" << endl;

cout << "5 - Ouro" << endl;

cin >> opcao;

}

return opcao - 1;

}

1. Composição com a classe Data. Fez uso do objeto criado?

Criado um ponteiro pra classe Data na classe Ferramenta que utiliza a data para registrar o dia da criação da ferramenta:

Ferramenta::Ferramenta(int tipoFerramenta, int maxRes, int atualRes, Data & dataDeCriacao) {

this->tipoFerramenta = tipoFerramenta;

this->setMaxRes(maxRes);

this->setAtualRes(atualRes);

this->dataDeCriacao = new Data(dataDeCriacao);

}

E utilizado no método infoItem(), também na classe Ferramenta:

void Ferramenta::infoItem() const {

cout << "Ferramenta criada no dia: ";

dataDeCriacao->print();

cout << "Este Item e ";

}

1. O que é const deve ser const.
2. Alocação dinâmica de memória. A memória é desalocada?

Atributos:

Data \* dataDeCriacao; (classe Ferramenta)

Bloco \* matMine; (classe Picareta)

Ambos destruídos nos destrutores de suas respectivas classes.

1. friend Operator<<

Na classe Bloco:

ostream &operator<< (ostream &output, const Bloco &bloco){

output << bloco.nome;

return output;

}

Na classe Ferramenta:

ostream &operator<< (ostream &output, const Ferramenta &ferramenta){

output << ferramenta.getTipoDaFerramenta();

return output;

}

Na classe Picareta:

ostream &operator<< (ostream &output, const Picareta &picareta){

output << static\_cast<Ferramenta>(picareta);

output << " de " << picareta.getTipoS();

return output;

}

1. Operator=
2. vector push\_back

Requisitos para as classes adicionais (pelo menos duas)

* Operator =
* Alocação dinâmica - se houver vazamento de memória a classe toda é desconsiderada
* Usar o destrutor
* Construtor de cópia
* Operator << friend
* Um const static

**Requisitos herança**

* Diagrama de classes (obrigatório salvar também o png do diagrama no gitHub)
  + Herança pública
  + Construtor de cópia, e sobrecargas dos operadores de atribuição (=) e << (cout << base) para a classe base e derivada
  + Usar Protected acessando diretamente os atributos na classe derivada
  + Alocação dinâmica de memória na classe base e derivada
  + Sobrescrita de método: chamar dentro do método da classe derivada o método correspondente da classe base usando ::
  + No main: criar um ponteiro da classe base para alocar memória para a classe derivada e chamar os vários métodos implementados