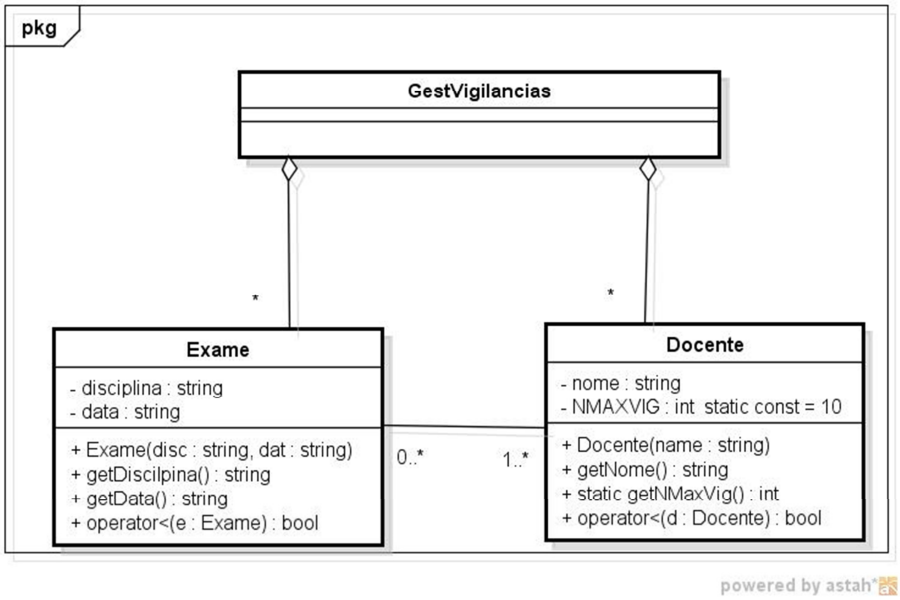
Grupo I

a)



b)

```
class GestVigilancias{
      Coleccao<Exame> exames;
      Coleccao<Docente> docentes;
public:
};

class Exame{
      string disciplina;
      string data;
      Coleccao<Docente*> vigilantes;
public:
      Exame(const string &disc, const string &dat) {disciplina=disc; data=dat;}
      string getDisciplina() const {return disciplina;}
      string getData() const {return data;}
      bool operator<(const Exame& exam) const {
            return disciplina+data<exam.disciplina+exam.data;
      }
};

class Docente{
      string nome;
      Coleccao<Exame*> vigilancias;
      static const int NMAXVIG;
public:
      Docente(const string &name): nome(name){}
      string getNome()const{return nome;}
      static int getNMaxVig(){return NMAXVIG;}
      bool operator<(const Docente& doc) const {return nome<doc.nome;}
};
const int Docente::NMAXVIG=2;
```

c)

```
bool GestVigilancias::addExame(const string &disc, const string &dat){
      Exame e(disc,dat);
      return exames.insert(e);
}
```

d)

```
void GestVigilancias::listarVigilanciasDeDocente(const string &name){
      Docente *fd=findDocente(name);
      if(fd!=NULL) fd->listarVigilancias();
      else cout << "Docente " << name << " nao existe!\n";
}

void Docente::listarVigilancias(){
      Coleccao<Exame*>::iterator it;
      cout<<"Vigilancias do docente "<<nome<<":\n";
      for(it=vigilancias.begin(); it!=vigilancias.end(); it++)
            cout<<(*it)->getDisciplina()<<" "<<(*it)->getData()<<endl;
}
```

e)

```
void GestVigilancias::removerExame(const string &disc, const string &dat) {
      Exame *fe=findExame(disc, dat);
      if(fe!=NULL){
            fe->removerVigilantes();
            Exame e(disc,dat);
            exames.erase(e);
      }else cout << "Exame de " << disc <<" na data "<<dat<< " nao existe!\n";
}

void Exame::removerVigilantes(){
      Coleccao<Docente*>::iterator it;
      for(it=vigilantes.begin(); it!=vigilantes.end(); it++)
            (*it)->removerVigilancia(this);
      vigilantes.clear();
}

void Docente::removerVigilancia(Exame *e) {vigilancias.erase(e);}
```

Grupo II

a) Identifique o tipo de relação existente entre as diferentes classes definidas no programa. [1 val.]

Herança entre Conta e ContaAPrazo
Agregação/Associação entre Conta e Cliente

b) Apresente o resultado que será visualizado na saída standard com a execução do programa. [3 val.]

```
Criada conta n.1
Fechada
Criada conta n.2
<Conta a Prazo>
Deposito invalido
Conta n.2 saldo:0
com 1o titular: Ana
com 2o titular: Rita
Conta a Prazo Fechada
```

c) Diga que alterações é que teria no resultado visualizado se na classe `Conta` [2 val.]

i) o método `depositar(.)` não fosse virtual.
não aparecia "Deposito invalido"
em vez de "saldo:0" aparecia "saldo:40"

ii) o método `levantar(.)` não fosse virtual.
não haviam alterações

iii) o método `print(.)` não fosse virtual.
não era mostrado o 2º titular

iv) o método `~Conta(.)` não fosse virtual.
em vez de "Conta a Prazo Fechada" aparecia apenas "Fechada"

d) Se acrescentássemos à função main() a linha de código que se segue, resultaria daí algum erro na nossa aplicação? Justifique. [1 val.]

```
ContaAPrazo contas[10000];
```

Sim. Só é possível criar arrays de instâncias de classes se estas dispuserem de um construtor por defeito, algo que não acontece com a `ContaAPrazo`.

e) Acrescente ao programa uma classe para um novo tipo de conta, de um só titular, denominada ContaAOrdem. Esse novo tipo de conta deverá ser caracterizado por não permitir que o saldo fique, em nenhum momento, a zero e pela obrigatoriedade de se efectuar um depósito (não importa o montante) no momento da sua abertura (criação da conta). [3 val.]

```
class ContaAOrdem: public Conta{
public:
      ContaAOrdem(Cliente *t1, double dep): Conta(t1){depositar(dep);}
      void levantar(double val) {if(val<saldo) levantar(val);}
      ~ContaAOrdem(){cout<<"Conta a Ordem ";}
};
```