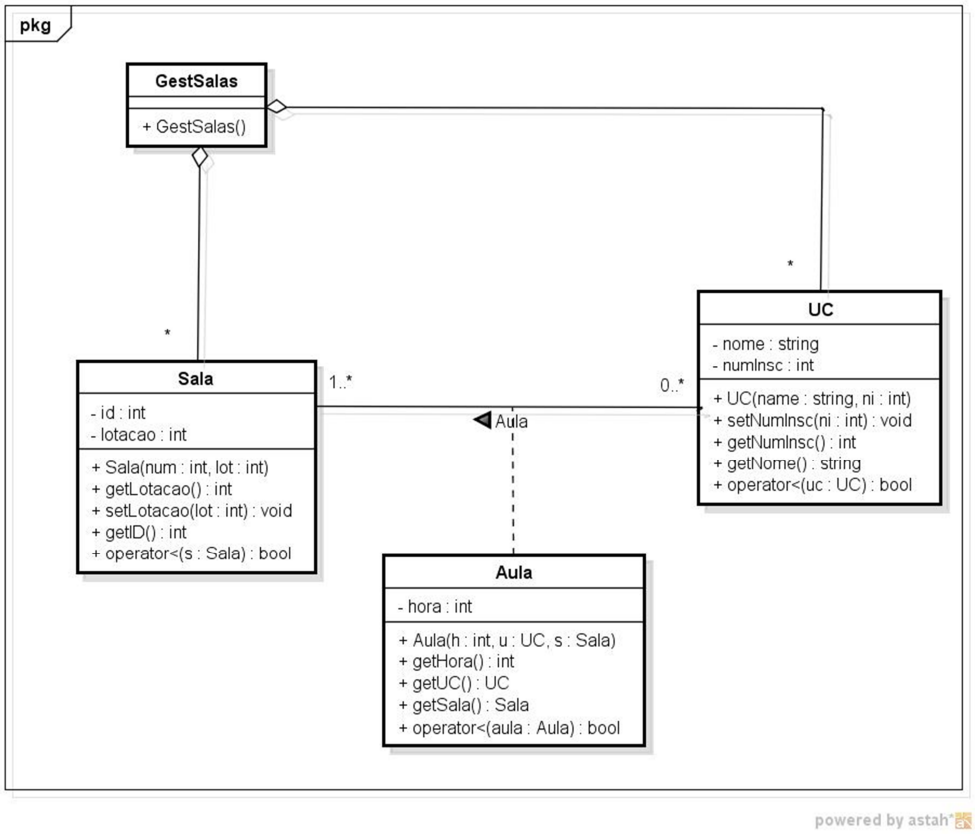
Grupo I

a)



b)

```
class Aula{
      int hora;
      Sala *sala;
      UC *uc;
public:
      Aula(int h, UC *u, Sala *s) {
            hora=h;
            sala=s;
            uc=u;
      }
      int getHora() { return hora;}
      UC *getUC() {return uc;}
      Sala *getSala() {return sala;}
      bool operator<(const Aula& aula) const { return hora<aula.hora;}
};

class Sala{
      int id;
      int lotacao;
      Coleccao<Aula> aulas;
public:
      Sala(int num, int lot) {id=num; lotacao=lot;}
```

```
	int getLotacao() {return lotacao;}
	void setLotacao(int lot) {lotacao=lot;}
	int getID() {return id;}
	bool operator<(const Sala& s) const { return id<s.id;}
};

class UC{
	string nome;
	int numInsc;
	Coleccao<Aula*> aulas;
public:
	UC(string name, int ni): nome(name) {numInsc=ni;}
	void setNumInsc(int ni) {numInsc=ni;}
	int getNumInsc() {return numInsc;}
	string getNome() {return nome;}
	bool operator<(const UC& uc) const { return nome<uc.nome;}
};

class GestSalas{
	Coleccao<Sala> salas;
	Coleccao<UC> ucs;
public:
};


c)
	bool GestSalas::addSala(int id, int lot) {
		if(findSala(id)==NULL){
			Sala s(id,lot);
			salas.insert(s);
			return true;
		}else return false;
	}
	Sala *GestSalas::findSala(int id) {
		Sala s(id,0);
		return salas.find(s);
	}

d)
	void GestSalas::listarAulasDeUmaSala(int nsala) {
		Sala *fs=findSala(nsala);
		if(fs!=NULL) fs->listarAulas();
		else cout<<"Sala "<<nsala<<" nao existe!\n";
	}
	void Sala::listarAulas() {
		Coleccao<Aula>::iterator it;
		cout<<"Aulas na Sala "<<id<<":\n";
		for(it=aulas.begin(); it!=aulas.end(); it++)
			cout<<it->getHora()<<"h - "<<it->getUC()->getNome()<<endl;
	}

e)
	bool GestSalas::alocarAula(string nomeUC, int hora, int nsala) {
		UC *fu=findUC(nomeUC);
```

```
        if(fu){
                Sala *fs=findSala(nsala);
                if(fs) return fs->alocarAula(hora,fu);
                else{
                        cout << "Sala: " << nsala << " nao existe.\n";
                        return false;
                }
        }else {
                cout << "Unidade Curricular: " << nomeUC << " nao existe.\n";
                return false;
        }
}
UC *GestSalas::findUC(string name) {
        UC u(name,0);
        return ucs.find(u);
}
bool Sala::alocarAula(int hora, UC* uc){
        if(disponivel(hora)){
                if(lotacao>=uc->getNumInsc()){
                        Aula a(hora,uc,this);
                        add(a);
                        return true;
                }else cout << "A sala indicada nao tem lotacao suficiente\n";
        }else cout << "A sala indicada esta ocupada na hora pretendida\n";
        return false;
};
void Sala::add(Aula &a){
        if(aulas.insert(a)) a.getUC()->add(aulas.find(a));
        else cout << "Já está marcada uma aula na hora pretendida\n";
}
```

Grupo II

a) Identifique o tipo de relação existente entre *Figura e Circunferencia* e entre *Figura* e *Ponto*.

Figura e Circunferencia: HERANÇA
Figura e Ponto: AGREGAÇÃO

b) Na função main(), a invocação de um dos métodos resulta em erro. Identifique o erro e proponha uma solução adequada para que essa invocação passe a ser possível.

A invocação **f->ampliar(3);** requer que o métotdo ampliar seja membro da classe Figura, uma vez que f está declarado como apontador para Figura.

Solução:

```
virtual void Figura::ampliar(int)=0;
```

c) Apresente o resultado que será visualizado na saída standard após a execução do programa e depois corrigido o erro referenciado na alínea anterior.

```
Circunferencia de raio 5 com centro em (0.1,10)
Circunferencia de raio 15 com centro em (0.1,10)
Circunferencia de raio 15 com centro em (0.2,20)
```

```
Circunferencia diz Xau
Figura diz Xau
Ponto diz Xau
Ponto diz Xau
Ponto diz Xau
```

d) Se acrescentarmos à função main() a linha de código que se segue, daí resultará algum erro na nossa aplicação? Se sim, proponha uma solução adequada para que essa linha de código deixe de dar erro.

```
Circunferencia varias[10];
```

SIM

solução:
```
Figura::Figura(){}
Circunferencia::Circunferencia(){raio=1;}
```

e) Acrescente ao programa uma nova classe de nome Circulo que, para alem de um centro e raio, tenha ainda como atributo a cor de preenchimento.

```
class Circulo: public Circunferencia{
      int cor_preench;
public:
      Circulo(){cor_preench=0;}
      Circulo(const Ponto& c, int r, int cor): Circunferencia(c,r){
            cor_preench=cor;
      }


      void mostrar() const{
            cout<<"Circulo de raio "<<raio<<" com centro em "<<posic;
            cout<<" e com a cor "<<cor_preench<<endl;
      }
      ~Circulo(){cout<<"Circulo diz Xau\n";}
};
```