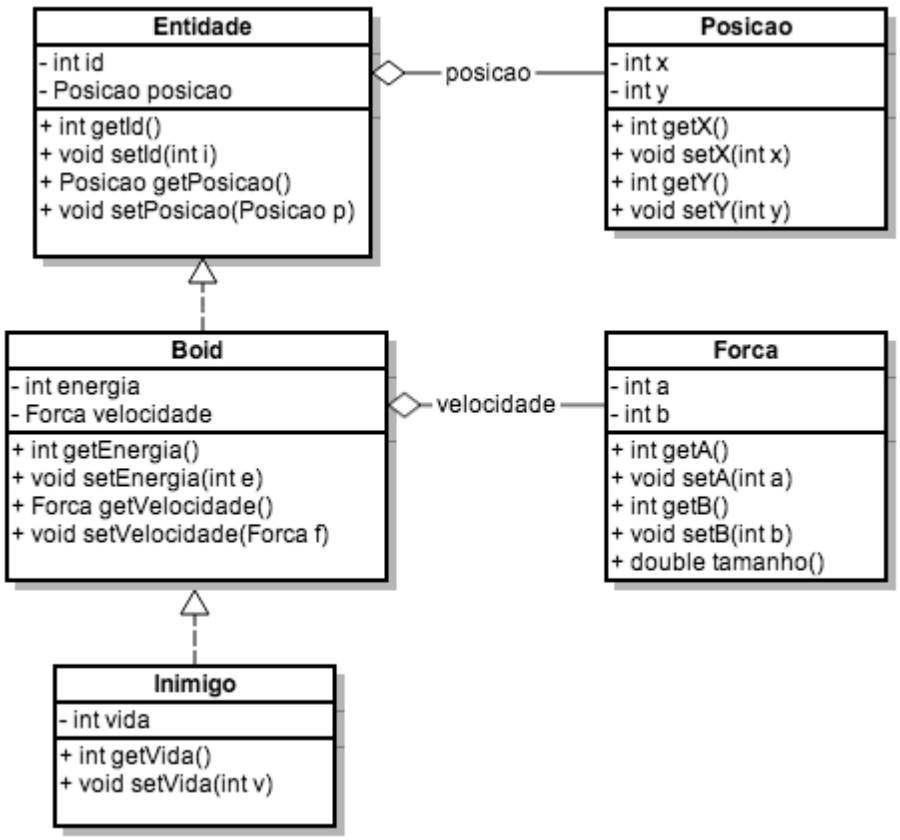
Universidade Federal da Fronteira Sul
Ciência da Computação
Programação I

Exercícios sobre modelagem e herança

1) Dado o diagrama UML abaixo:



Implemente os seguintes métodos:

a) `int contaPossiveisCandidatos(Inimigo i[])` - método *estático* pertencente à classe `Services` (crie essa classe também). Retorna o número de objetos do vetor que tenham: posição `x` ou `y` maiores que `100`, `energia` menor ou igual a `50` e `velocidade` com tamanho igual a `200`.

b) `Inimigo achaAlvoMaisProximo(Inimigo i[])` - método pertencente à classe `Inimigo`. O método itera no vetor de inimigos recebido como parâmetro, retornando o inimigo cujas componentes `x,y` da posição E as componentes `a,b` da velocidade sejam iguais aos do objeto que invocou o método.

2) Uma das habilidades mais importantes para se programar no paradigma de orientação a objetos é a capacidade de criar as classes com base na interpretação e abstração de uma cena/ambiente. Imaginando que você recebeu a tarefa de modelar um jogo de Xadrez, faça o diagrama UML com as classes, métodos e atributos necessários para cumprir a tarefa. Uma peça pode estar vida ou morta e pode se mover pelo tabuleiro; o tabuleiro deve ser capaz de informar qual peça está numa determinada posição (com base na linha e coluna informadas), bem como informar quantas peças estão vivas (ou mortas).

Universidade Federal da Fronteira Sul

Ciência da Computação

Programação I

3) O código abaixo foi feito por um membro da sua equipe, que alegou que o resultado é modular e encapsulado. A classe `FlxSprite`, segundo seu colega, ***não pode ser alterada*** pois é código legado da aplicação. Ele mostrou a você alguns casos de uso da classe `FlxSprite`: casos #1, #2 e #3 (veja eles abaixo). Você argumentou que a classe `FlxSprite` não parecia encapsulada ou modular, e que uma alternativa seria criar uma nova classe, chamada `Character`, que herda de `FlxSprite` e faz a inicialização de todos os atributos no momento da ***instanciação***.

```
class FlxObject {
  private String label;

  public FlxObject(String h) {
    setLabel(h);
  }

  public void setLabel(String h) {
    label = h;
  }
}

class FlxSprite extends FlxObject {
  private String graphic;
  private boolean alive;
  public int x,y;

  public FlxSprite(String j) {
    super(j);
  }

  void setAlive(boolean b) {
    alive = b;
  }

  void loadGraphics(String n) {
    graphic = n;
  }

  void reset(int x, int y) {
    this.x = x; this.y = y;
  }
}
```

Caso de uso #1

```
FlxSprite a = new FlxSprite("um");

a.setAlive(true);
a.loadGraphics("player1.png");
a.reset(20, 20);
```

Caso de uso #2

```
FlxSprite b = new FlxSprite("dois");

b.setAlive(true);
b.loadGraphics("another.png");
b.reset(-40, 13);
```

Caso de uso #3

```
FlxSprite c = new FlxSprite("tres");

c.setAlive(true);
c.loadGraphics("newone.png");
c.reset(10, 5);
```

Escreva o código da classe `Character` de tal forma que ela herde da classe `FlxSprite`, seja encapsulada e faça todas as inicializações dos casos de uso #1, #2 e #3 diretamente no construtor.

Universidade Federal da Fronteira Sul

Ciência da Computação

Programação I

4) Dadas as classes abaixo, mostre o que será impresso na tela quando o programa `Main` for executado.

```
class Sprite {
 private int x;
 public int scaleY;

 public Sprite (int x, int s) {
   this.x = x;
   this.scaleY = s;
 }
 public int getX() {
   return this.x;
 }
 public void setX(int i) {
   this.x = i;
 }
 public void oi() {
   imprime();
   System.out.println("Soi");
 }
 public void inverte() {
   this.x = -this.scaleY;
   System.out.println("x = " + getX());
 }
 public String resumo() {
   return x + " " + scaleY;
 }
 public void imprime() {
   System.out.println(getX() + " " + scaleY);
 }
}
```

```
class MovieClip extends Sprite {
 private float alpha;

 public MovieClip(float h) {
   super(10, 22);
   alpha = h;
   setX(12);
 }

 public void negativo() {
   inverte();
   alpha = 0;
   System.out.println("Negativo!");
 }

 public String resumo() {
   return "Movie: " + super.resumo();
 }

 public void imprime() {
   System.out.println(resumo() + " " + alpha);
 }
}
```

```
class MovieBlock extends MovieClip {
  public MovieBlock(float h) {
    super(h);
    setX(40);
  }

  public String resumo() {
    return "Block: resumo";
  }

  public void imprime() {
    super.imprime();
    System.out.println("Block "+getX());
  }

  public void inverte() {
    setX(900);
    System.out.println("b inverte");
  }
}
```

```
class Main {
 public static void main(String args[]) {
  Sprite s = new Sprite(1, 2);
  MovieClip m = new MovieClip(5.5f);
  MovieBlock b = new MovieBlock(7.3f);

  System.out.println(s.resumo());
  s.imprime();
  m.negativo();
  m.inverte();
  m.imprime();
  m.oi();
  System.out.println("b.x = " + b.getX());
  b.imprime();
  b.negativo();
 }
}
```

Universidade Federal da Fronteira Sul

Ciência da Computação

Programação I

5) Utilizando o diagrama UML da questão **1)**, imagine que você foi incumbido de criar uma classe nova chamada `InimigoVolatil`, que é filha da classe `Inimigo`. Ao contrário de sua classe pai, a classe `InimigoVolatil` não utiliza a propriedade `vida` quando os métodos `getVida()` e `setVida()` são invocados, ela usa uma combinação das propriedades `posicao` e `velocidade`. O método `getVida()`, por exemplo, retorna a soma dos valores `x` e `y` da propriedade `posicao` do objeto; o método `setVida(int v)` coloca o valor de `v` nos atributos `a` e `b` da propriedade `velocidade` do objeto. O método `getEnergia()` da classe `InimigoVolatil` também funciona de forma diferente: ele retorna o valor da propriedade `vida` somada com a propriedade `energia`.

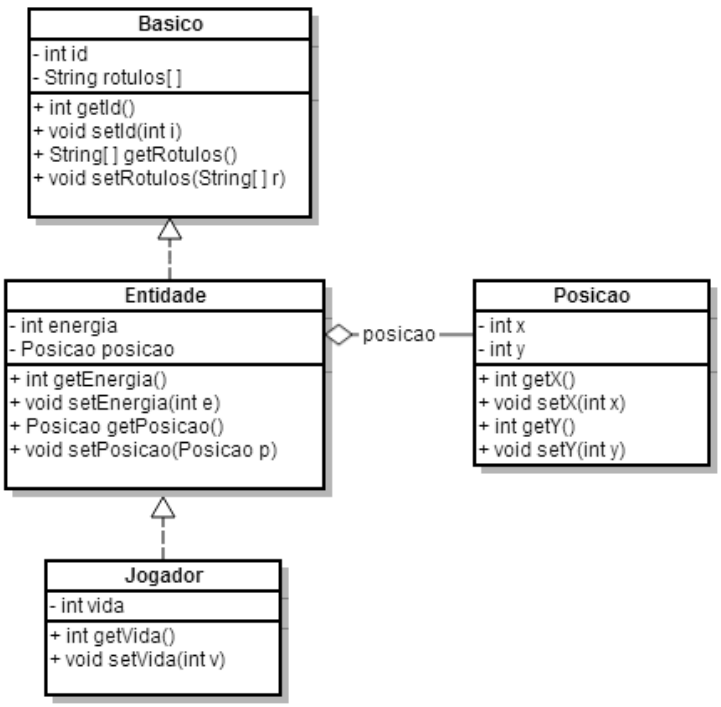
Implemente a classe `InimigoVolatil`, fazendo as alterações necessárias para que os métodos `getVida()`, `setVida()` e `getEnergia()` funcionem conforme o que foi descrito.

6) Modele um sistema que represente um PC. Um PC possui programas que rodam na CPU. Cada programa pode ler e escrever de um espaço de memória. Os espaços de memória são limitados (ex.: existem 30 deles) e controlados pelo SO (Sistema Operacional. O SO é o único capaz de dar espaços de memória para uso (e receber eles de volta quando um programa decide não usá-los mais). Um programa pode usar N espaços de memória. Além da memória, um programa pode ter acesso a um espaço de disco. Assim como um espaço de memória, um espaço no disco funciona na forma endereço/valor: com um endereço (ex. 5), pode-se escrever ou ler um dado daquele espaço. Espaço de memória e disco são exclusivos de um programa depois que esse os recebe do SO. A CPU possui uma lista de programas em execução.

7) Utilize o diagrama UML abaixo. Escreva os métodos **construtores** necessários para as classes para que um objeto `Jogador` possa ser instanciado e tenha todas as suas propriedades inicializadas através de parâmetros informados (vide exemplo abaixo). Você **não** pode criar métodos que não sejam construtores (como getters e setters), apenas usar os já existentes.

```
String h[];
int vida, energia, x, y, id;

Jogador j = new Jogador(vida, energia, x, y, id, h);
```



8) Dado o código das classes abaixo, qual será a saída do programa na tela?

```
class Grandfather {
      private float avg;
      public Grandfather()           {  this("Grandfather done");          }
      public Grandfather(String h)   {  System.out.println(h); }
}
class Father extends Grandfather {
      private String name;

      public Father(int i)           {  this("Father");                    }
      public Father(String h) {
            super("ok");
            name = h;
            System.out.println(h + " almost");
      }
}
class Son extends Father {
      public int id;

      public Son(int i)             {  this(3.14); id = i;                 }
      public Son(double t)          {  this();                              }
      public Son() {
            super(4);
            System.out.println("Son done!");
      }
}
class Main {
      public static void main(String[] args) {
            Son s = new Son(5);
            Father f = new Father("Hommer");
            Grandfather g = new Grandfather();
            System.out.println("s.id = " + s.id);
      }
}
```

9) Preencha as lacunas de acordo com os conceitos ou termos da programação orientada a objetos.

a. Uma _____ serve como um modelo para a criação de objetos.
b. O ato de alocar um espaço de memória para um objeto de uma determinada classe através do operador `new` chama-se _____.
c. Os membros de uma classe que definem seu estado/características são os _____.
d. Os membros de uma classe que definem seu comportamento e/ou funcionalidades são os _____.
e. O _____ é um método especial, público e sem retorno, executado sempre no momento da criação de um objeto.
f. O mecanismo de _____ permite que uma classe reutilize as propriedades e/ou métodos já definidos em outra classe mais genérica. A classe que recebeu as características chama-se _____ e a classe que foi estendida chama-se_____.
g. Um membro precedido do modificador _____ é visível somente dentro da classe onde foi declarado. Um membro _____ é visível na própria classe, em suas subclasses ou outras classes do mesmo pacote. Já um membro _____ é acessível por qualquer outra classe de qualquer pacote.

h. O conceito de _____ diz que uma classe deve funcionar como uma caixa-preta: não precisamos conhecer os detalhes internos de sua implementação, apenas conhecer sua interface pública.
i. Os atributos precedidos do modificador _____ possuem um valor que é compartilhado por todas as instâncias de uma classe. Já os métodos que possuem o mesmo modificador não exigem a instanciação de um objeto, podendo ser chamados diretamente através do nome da classe.
j. O operador _____ permite referenciar membros da própria classe, sendo utilizado, por exemplo, para resolver ambiguidades entre nomes do membro e de parâmetros com o mesmo nome.
k. Uma classe que possui a palavra-chave _____ na sua declaração não pode ser instanciada.
l. Um atributo precedido da palavra _____ não pode ter seu valor alterado em tempo de execução.

10) Dada a classe abaixo:

```
class Father {
  private int x, y, lado;

  public int getLado()                 { return lado; }
  public void setLado(int l)           { lado = l; }
  public int area()                    { return lado * lado; }
  public void translada(int a, int b) { x = a; y = b; }

  public String debug()                {
      return getLado() + " (" + x + "," + y + ") = " + area();
  }
}
```

E o seguinte código:

```
class Main {
      public static void main(String[] args) {
            Father f = new Father();
            Son s = new Son();

            f.setLado(2);
            f.translada(5, 6);
            System.out.println(f.debug());

            s.setLado(10);
            s.translada(10, 11);
            s.escala(0.5);
            System.out.println("Lado encolhido: " + s.getLado());
            System.out.println(s.debug());
      }
}
```

Implemente a classe `Son`, **sem criar** propriedades e herdando da classe `Father`, para que o código acima gere a seguinte saída:

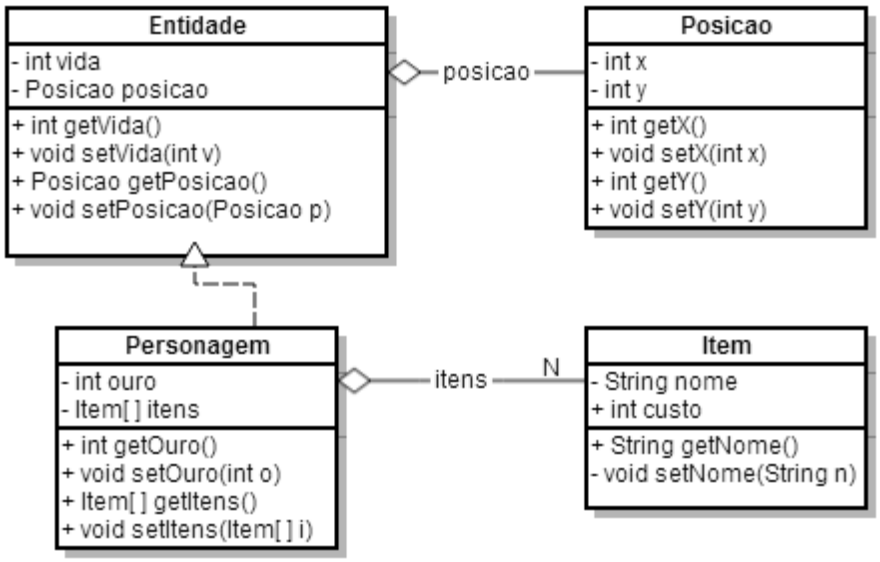
```
2 (5, 6) = 4
Lado encolhido: 5
5 (10, 11) = -25
```

Universidade Federal da Fronteira Sul

Ciência da Computação

Programação I

11) Utilize o diagrama UML abaixo:



Para a classe `Personagem` assuma que a propriedade `itens` (e seus getters/setter) **não existe mais**. Escreva os métodos **construtores** necessários para as classes para que o trecho de código abaixo funcione e inicialize **todas** as propriedades dos objetos instanciados. Você **não** pode criar métodos que não sejam construtores (como getters e setters), apenas usar os já existentes.

```
int x = 5, y = 6, ouro = 10, vida = 500;
Entidade e = new Entidade(x, y, vida);
Entidade e2 = new Entidade(); // inicializa todas os atributos com zero
Personagem p = new Personagem(ouro, vida, x, y);
Personagem p2 = new Personagem(ouro, vida); // inicializa a posição em (0,0)
```