GABARITO

Para responder as questões de 1 a 3, analise a classe Monster definida a seguir:

```
import java.util.*;
import java.awt.*;

class Monster
{
  int x,y;
  int largCampo, altCampo;
  Image figura;
  static Random r = null;

  // Construtor recebe dimensões da tela e a imagem a desenhar
  public Monster(int larg,int alt, Image fig)
  {
    if(r==null) r = new Random();
    largCampo = larg;
    altCampo = alt;
    x = r.nextInt(largCampo);
    y = r.nextInt(altCampo);
    figura = fig;
  }

  public int retornaVel()
  {
     return(5);
  }

  public void move()
  {
    // sorteia direção para movimento
    boolean dirx = r.nextBoolean();
    boolean diry = r.nextBoolean();
    // Avança na direção escolhida
    if (dirx) x+= retornaVel();
      else x-= retornaVel();
    if (diry) y+= retornaVel();
      else y-= retornaVel();
    // Testa se não estourou os limites
    if (x<0) x=0;
    if (y<0) y=0;
    if (x>largCampo) x=largCampo-1;
    if (y>altCampo)  y=altCampo-1;
  }

  public void draw(Graphics g)
  {
    g.drawImage(...);
  }
}
```

1. (1 pt) A partir desta classe, crie duas novas classes: FastMonster e SlowMonster. Na classe FastMonster, o passo de avanço deve ser de 20 unidades. Na classe SlowMonster, o passo de avanço deve ser de 4 unidades.

```
class FastMonster extends Monster
{
  // Construtor recebe dimensões da tela e a imagem a desenhar
  public FastMonster(int larg,int alt, Image fig)
  {
      super(larg,alt,fig);
  }

  public int retornaVel()
  {
     return(20);
  }
}

class SlowMonster extends Monster
{
  // Construtor recebe dimensões da tela e a imagem a desenhar
  public SlowMonster(int larg,int alt, Image fig)
  {
      super(larg,alt,fig);
  }

  public int retornaVel()
  {
     return(4);
  }
}
```

2. (2 pt) Crie uma classe chamada MonsterList que seja capaz de armazer instâncias das classes Monster, FastMonster e SlowMonster. A classe deve ter métodos para:
   - incluir um monstro (de qualquer tipo);
   - retornar uma referência para o monstro armazenado em determinada posição (se não houver, retornar null);
   - retornar o total de monstros armazenados

   OBS:
   - se for o caso, o número máximo de monstros que a lista armazenará pode ser informado no construtor;
   - A questão deve <u>obrigatoriamente</u> explorar o conceito de polimorfismo!

```
class MonsterList
{
  Monster lista[];  int total, max;

  public MonsterList(int max)
  {
    lista = new Monster[max];
    this.max = max;
    total = 0;
  }

  public void insere(Monster m)
  {
    if(total<max)
    {
      lista[total] = m;
```

```
      total++;
    }
  }

  public Monster get(int pos)
  {
    return lista[pos];
  }

  public int quant()
  {
    return total;
  }
}
```

3.

a) (1,5 pt) Escreva uma interface denominada SmartMonster, que deve definir um método para que os monstros se movimentem com mais "inteligência": smartMove. Para tanto, este método smartMove deve receber uma referência para uma MonsterList. A interface ainda deve definir um atributo que indica a distância mínima a ser considerada no movimento.

```
interface SmartMonster
{
  public int mindist = 10;

  public void smartMove(MonsterList ml);
}
```

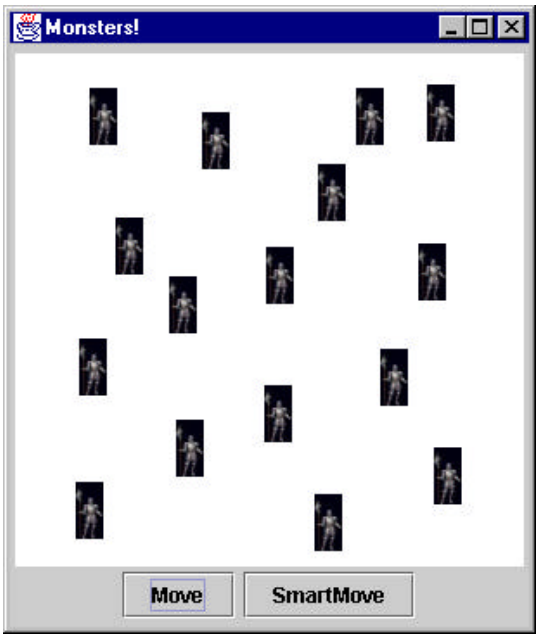
b) (2,5 pt) Reescreva a classe Monster, implementando esta interface, ou seja, o método smartMove deverá fazer o monstro andar na direção do colega mais próximo (respeitando a distância mínima – use o método *double distance(double px,double py)* da classe *java.awt.Point*). Se não houver ninguém próximo o suficiente, o movimento deverá ser aleatório, como antes.

```
class Monster implements SmartMonster
{
  // ... repete atributos e métodos como definido antes

  public void smartMove(MonsterList ml)
  {
    double proxdist = 100000;
    Point pos = new Point(x,y);    // coordenadas do monstro
    Monster proxmonst = null;      // monstro mais próximo

    for(int i=0; i<ml.quant();++i)
    {
      Monster m = ml.get(i);  // obtém referência

      if(m==this) continue;  // se for o mesmo, não pode testar
      double dist = pos.distance(m.x,m.y); // calcula distância
      if (dist<mindist) continue;  // dist. menor que a mínima, ignora
      if (dist>proxdist) continue; // dist. maior que a menor encontrada

      proxdist = dist;  // armazena menor distância
      proxmonst = m;    // armazena monstro mais próximo
    }

    if(proxmonst == null) move(); // ninguém, usa a move()
    else
```

```
    {
      int deltax = proxmonst.x – x;   // calc diferença entre os x
      int deltay = proxmonst.y – y;   // calc diferença entre os y
      if(deltax<0) x-=retornaVel();   // menor que 0 = andar p/ esquerda
      if(deltax>0) x+=retornaVel();   // maior que 0 = andar p/ direita
      if(deltay<0) y-=retornaVel();  // menor que 0 = andar p/ cima
      if(deltay>0) y+=retornaVel();  // maior que 0 = andar p/ baixo

      // Verifica se passou dos limites da janela, ajusta de acordo
      if (x<0) x=0;
      if (y<0) y=0;
      if (x>largCampo) x=largCampo-1;
      if (y>altCampo)  y=altCampo-1;
    }
  }
}
```

4. (3 pt) Crie uma aplicação Java capaz de exibir uma janela como a da figura abaixo. A aplicação deve dispor de uma instância de MonsterList e cadastrar 10 monstros de cada tipo (Monster, SlowMonster e FastMonster). Se o botão Move for pressionado, todos os monstros devem se mover aleatoriamente. Se o botão SmartMove for pressionado, todos os monstros devem se mover "inteligentemente".

   OBS: Imagine que já existe uma classe *Tela* (derivada de Jpanel), cujo construtor deve receber uma referência para a lista de monstros. Esta classe é responsável pelo desenho dos monstros. **Não é necessário implementá-la!!!**




```
class Aplic extends JFrame implements ActionListener
{
  MonsterList ml = new MonsterList(30);

  public Aplic()
  {
    super();

    JButton butmove  = new JButton("Move");
    JButton butsmove = new JButton("SmartMove");
    butmove.addActionListener(this);
    butsmove.addActionListener(this);
```

```
    JPanel bots = new JPanel();
    bots.add(butmove);
    bots.add(butsmove);

    Tela tela = new Tela(ml);

    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(tela,BorderLayout.CENTER);
    getContentPane().add(bots,BorderLayout.SOUTH);

    // cria 10 monstros de cada tipo
    for(int i=0; i<10; ++i)
    {
      ml.insere(new Monster(200,200,knight));
      ml.insere(new FastMonster(200,200,knight));
      ml.insere(new SlowMonster(200,200,knight));
    }
  }



  public void actionPerformed(ActionEvent e)
  {
      boolean intelligent;    // true = movimento inteligente
      String action = e.getActionCommand();
      if(action.equals("Move")) intelligent = false;
      else intelligent = true;
      for(int i=0;i<ml.quant();++i)
        if(intelligent) ml.get(i).smartMove(ml);
          else ml.get(i).move();
      repaint();
  }

  public static void main(String args[])
  {
     Aplic frame = new Aplic();
     frame.pack();
     frame.show();
  }
}
```