PUC-Minas - Ciência da Computação
AED1 – Guia 03

Tema: Introdução à programação
Atividade: Montagem de programas - JKarel

01.) Editar e salvar um esboço de programa,
o nome do arquivo deverá ser Guia0041.java,
tal como o nome da classe abaixo,
concordando maiúsculas e minúsculas, sem espaços em branco:

```
/**
  * Guia0041
  *
  * @author
  * @version 01
  */
//
// Lista de dependencias
//
  import jkarel.World;
  import jkarel.Robot;

  import IO.*;

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0041 extends Robot
{
 /**
  * construtor padrao da classe Guia00.
  * @param avenue    - uma   das coordenadas da posicao inicial
  * @param street       - outra das coordenadas da posicao inicial
  * @param direction - direcao inicial
  * @param beepers   - quantidade inicial de marcadores
  */
   public Guia0041( int avenue, int street, int direction, int beepers )
   {
    // metodo para repassar dados
    // ao construtor padrao da classe original (Robot)
      super( avenue, street, direction, beepers );
  } // end Guia0041( )
```

```
/**
   * metodo para criar configuracoes do ambiente.
   * @param nome do arquivo onde guardar a configuracao
   */
    public static void createWorld( String nome )
    {
     // o executor deste metodo (World - agente)
     // ja' foi definido na classe original (Robot)
        World.reset( );                    // limpar configuracoes
     // para nao exibir os passos de criacao do ambiente
        World.setTrace( false );       // (opcional)

     // para criar obstaculos
     // Guia0-001.txt
        World.placeNSWall( 1, 1, 1 );  // espelho do degrau 1
        World.placeEWWall( 2, 1, 1 ); // patamar do degrau 1
        World.placeNSWall( 2, 2, 1 );  // espelho do degrau 2
        World.placeEWWall( 3, 2, 1 ); // patamar do degrau 2
        World.placeNSWall( 3, 3, 1 );  // espelho do degrau 3
        World.placeEWWall( 4, 3, 1 ); // patamar do degrau 4
        World.placeNSWall( 4, 1, 3 );  // apoio da escada 'a direita

     // para colocar marcadores
        World.placeBeepers( 4, 4, 1 );// marcador no topo da escada

     // para guardar em arquivo
        World.saveWorld( nome );     // gravar configuracao
    } // end createWorld( )

  /**
   * metodo para virar 'a direita.
   */
    public void turnRight( )
    {
     // o executor deste metodo
     // deve virar tres vezes 'a esquerda
        turnLeft( );
        turnLeft( );
        turnLeft( );
    } // end turnRight( )
```

```
/**
  * metodo para contar comandos de arquivo.
  * @param filename - nome do arquivo
  */
   public void countCommands( String filename )
   {
    // definir dados
       int  length;
       FILE archive = new FILE ( FILE.INPUT, filename );
       String line;

    // repetir enquanto houver dados
       length = 0;
       line   = archive.readln ( );
       while ( ! archive.eof( ) )
       {
         // contar mais um comando
            length = length + 1;
          // tentar ler a proxima linha
            line = archive.readln( );
       } // end while
    // fechar o arquivo
       archive.close( );
    // informar a quantidade de comandos guardados
       IO.print ( "File " + filename + " has " );
       IO.println ( "" + length + " commands." );
   } // end countCommands( )

  /**
   *  Acao principal: executar a tarefa descrita acima.
   */
   public static void main( String [ ] args )
   {
    // criar o ambiente com escada
    // OBS.: executar pelo menos uma vez,
    //           antes de qualquer outra coisa
    //          (depois de criado, podera' ser comentado)
       createWorld( "Guia0041.txt" );

    // comandos para tornar o mundo visivel
       World.reset( );                        // limpar configuracoes
       World.setSpeed ( 6 );                  // escolher velocidade
       World.readWorld( "Guia0041.txt" );     // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
       Guia0041 JK = new Guia0041( 1, 1, World.EAST, 0 );

    // executar acoes
    // informar quantidade de comandos
       JK.countCommands( "Tarefa0001.txt" );
   } // end main( )
} // end class
```

```
// ------------------------------------------------ testes

/*
  Versao      Teste
  0.1         01. (     )      - teste inicial
*/
```

02.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

03.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

04.) Copiar a versão atual do programa para outra (nova) – Guia0042.java.

05.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * metodo para contar comandos de arquivo.
  *@return quantidade de comandos
  *@param filename - nome do arquivo
  */
   public int countCommands( String filename )
   {
    // definir dados
      int  length;
      FILE archive = new FILE ( FILE.INPUT, filename );
      String line;

    // repetir enquanto houver dados
      length = 0;
      line   = archive.readln ( );
      while ( ! archive.eof( ) )
      {
        // contar mais um comando
          length = length + 1;
        // tentar ler a proxima linha
          line = archive.readln( );
      } // end while
    // fechar o arquivo
      archive.close( );
    // retornar a quantidade de comandos guardados
       return length;
   } // end countCommands( )
```

```
/**
 *  Acao principal: executar a tarefa descrita acima.
 */
 public static void main( String [ ] args )
 {
  // criar o ambiente com escada
  // OBS.: executar pelo menos uma vez,
  //          antes de qualquer outra coisa
  //         (depois de criado, podera' ser comentado)
    createWorld( "Guia0042.txt" );

  // comandos para tornar o mundo visivel
    World.reset( );                          // limpar configuracoes
    World.setSpeed ( 6 );                    // escolher velocidade
    World.readWorld( "Guia0042.txt" );    // ler configuracao do ambiente

  // definir o objeto particular para executar as acoes (agente)
    Guia0042 JK = new Guia0042( 1, 1, World.EAST, 0 );

  // definir dado local
     int n;   // para a quantidade de comandos

  // executar acoes
  // obter a quantidade de comandos e guardar
     n = JK.countCommands( "Tarefa0001.txt" );
  // informar a quantidade de comandos guardada
     IO.println ( "File Tarefa0001.txt has " +
                  "" + n + " commands." );
 } // end main( )
```

06.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

07.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

08.) Copiar a versão atual do programa para outra (nova) – Guia0043.java.

09.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * metodo para receber comandos de arquivo.
  * @return grupo formado por todos os comandos
  * @param filename - nome do arquivo
  */
   public int [ ] readCommands( String filename )
   {
    // definir dados
      int [ ] commands;  // para armazenar comandos
      int  length;
      int  option;
      FILE archive = new FILE ( FILE.INPUT, filename );
      String line;
      int  x;

    // obter a quantidade de comandos
      length = countCommands ( filename );

    // criar um armazenador para os comandos
    // INDISPENSAVEL reservar o espaco para guardar
      commands = new int [ length ];

    // repetir para a quantidade de comandos
      for ( x=0; x<length; x=x+1 )
      {
       // tentar ler a proxima linha
          line = archive.readln( );
       // decodificar a linha
       // e obter o codigo do comando
          option = IO.getint( line );
       // guardar um comando
       // na posicao (x) do armazenador
          commands [ x ] = option;
      } // end for
    // fechar o arquivo
    // INDISPENSAVEL para a gravacao
      archive.close( );
    // retornar os comandos guardados
      return commands;
   } // end readCommands( )
```

```
/**
 *  Acao principal: executar a tarefa descrita acima.
 */
 public static void main( String [ ] args )
 {
  // criar o ambiente com escada
  // OBS.: executar pelo menos uma vez,
  //         antes de qualquer outra coisa
  //         (depois de criado, podera' ser comentado)
    createWorld( "Guia0043.txt" );

  // comandos para tornar o mundo visivel
    World.reset( );                        // limpar configuracoes
    World.setSpeed ( 6 );                  // escolher velocidade
    World.readWorld( "Guia0043.txt" );     // ler configuracao do ambiente

  // definir o objeto particular para executar as acoes (agente)
    Guia0043 JK = new Guia0043( 1, 1, World.EAST, 0 );

   // definir dado local
     int n;          // para a quantidade de comandos
     int [ ] cmd; // para guardar os comandos
     int k;          // indice do comando

  // executar acoes
  // obter a quantidade de comandos
    n = JK.countCommands( "Tarefa0001.txt" );
  // informar a quantidade de comandos
    IO.println ( "File Tarefa0001.txt has " +
                 "" + n + " commands." );
   // ler todos os comandos do arquivo
     cmd = JK.readCommands ( "Tarefa0001.txt" );
   // mostrar os comandos guardados, um por vez
     for ( k=0; k<n; k=k+1 )
     {
       IO.println ( "" + k + "\t" + cmd[k] );
     } // fim repetir
 } // end main( )
```

10.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

11.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

12.) Copiar a versão atual do programa para outra (nova) – Guia0044.java.

13.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
   * metodo para executar um comando.
   * @param option - comando a ser executado
   */
    public void execute( int option )
    {
     // executar esse comando
         switch ( option )
         {
           case 1: // virar para a esquerda
             if ( leftIsClear ( ) )
             {
               turnLeft( );
             } // end if
            break;
           case 2: // virar para o sul
             while ( ! facingSouth( ) )
             {
               turnLeft( );
             } // end while
            break;
           case 3: // virar para a direita
             if ( rightIsClear ( ) )
             {
               turnRight( );
             } // end if
            break;
           case 4: // virar para o oeste
             while ( ! facingWest( ) )
             {
               turnLeft( );
             } // end while
            break;
           case 5: // mover
             if ( frontIsClear ( ) )
             {
               move( );
             } // end if
            break;
           case 6: // virar para o leste
             while ( ! facingEast( ) )
             {
               turnLeft( );
             } // end while
            break;
```

```
        case 7: // pegar marcador
          if ( nextToABeeper( ) )
          {
            pickBeeper( );
          } // end if
         break;
        case 8: // virar para o norte
          while ( ! facingNorth( ) )
          {
            turnLeft( );
          } // end while
         break;
        case 9: // colocar marcador
          if ( anyBeepersInBeeperBag( ) )
          {
            putBeeper( );
          } // end if
         break;
      } // end switch
  } // end execute( )

/**
  * metodo para executar comandos de arquivo.
  * @param commands - grupo de comandos para executar
  */
   public void doCommands( int [ ] commands )
   {
    // definir dados
      int  length = commands.length;   // obter a quantidade
      int  option;
      int  x;

    // repetir para a quantidade de comandos
      for ( x=0; x<length; x=x+1 )
      {
       // executar esse comando
          execute( commands [ x ] );
      } // end for
  } // end doCommands( )
```

```
/**
 *  Acao principal: executar a tarefa descrita acima.
 */
 public static void main( String [ ] args )
 {
  // criar o ambiente com escada
  // OBS.: executar pelo menos uma vez,
  //          antes de qualquer outra coisa
  //         (depois de criado, podera' ser comentado)
    createWorld( "Guia0044.txt" );

  // comandos para tornar o mundo visivel
    World.reset( );                          // limpar configuracoes
    World.setSpeed ( 6 );                    // escolher velocidade
    World.readWorld( "Guia0044.txt" );    // ler configuracao do ambiente

  // definir o objeto particular para executar as acoes (agente)
    Guia0044 JK = new Guia0044( 1, 1, World.EAST, 0 );

  // definir dado local
    int n;           // para a quantidade de comandos
    int [ ] cmd;   // para guardar os comandos
    int k;           // indice do comando

  // executar acoes
  // obter a quantidade de comandos
    n = JK.countCommands( "Tarefa0001.txt" );
  // informar a quantidade de comandos
    IO.println ( "File Tarefa0001.txt has " +
                 "" + n + " commands." );
  // ler todos os comandos do arquivo
    cmd = JK.readCommands ( "Tarefa0001.txt" );
  // mostrar os comandos guardados, um por vez
    for ( k=0; k<n; k=k+1 )
    {
      IO.println ( "" + k + "\t" + cmd[k]);
    } // fim repetir
  // executar comandos
     JK.doCommands ( cmd );
 } // end main( )
```

14.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

15.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

16.) Copiar a versão atual do programa para outra (nova) – Guia0045.java.

17.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * metodo para executar comandos de arquivo.
  * @param filename - nome do arquivo
  */
   public void doTask( String filename )
   {
    // definir dado local
       int n;          // para a quantidade de comandos
       int [ ] cmd;  // para guardar os comandos

    // obter a quantidade de comandos
       n = countCommands( filename );
    // ler comandos do arquivo
       cmd = readCommands ( filename );
    // executar comandos
       doCommands ( cmd );
   } // end doTask( )

 /**
  *  Acao principal: executar a tarefa descrita acima.
  */
  public static void main( String [ ] args )
  {
   // criar o ambiente com a escada
   // OBS.: executar pelo menos uma vez,
   //           antes de qualquer outra coisa
   //         (depois de criado, podera' ser comentado)
      createWorld( "Guia0045.txt" );

   // comandos para tornar o mundo visivel
      World.reset( );                          // limpar configuracoes
      World.setSpeed ( 6 );                    // escolher velocidade
      World.readWorld( "Guia0045.txt" );    // ler configuracao do ambiente

   // definir o objeto particular para executar as acoes (agente)
      Guia0045 JK = new Guia0045( 1, 1, World.EAST, 0 );

    // executar acoes descritas no arquivo
       JK.doTask( "Tarefa0001.txt" );
  } // end main( )
```

18.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

19.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

20.) Copiar a versão atual do programa para outra (nova) – Guia0046.java.

21.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * metodo para criar configuracoes do ambiente.
  * @param nome do arquivo onde guardar a configuracao
  */
   public static void createWorld( String nome )
   {
    // o executor deste metodo (World - agente)
    // ja' foi definido na classe original (Robot)
      World.reset( );                  // limpar configuracoes

    // para nao exibir os passos de criacao do ambiente
      World.setTrace( false );      // (opcional)

     // para colocar marcador(es)
       World.placeBeepers( 4, 4, 1 );

    // para guardar em arquivo
      World.saveWorld( nome );     // gravar configuracao
   } // end createWorld( )

/**
  * metodo para o robot explorar o mundo.
  */
   public void mapWorld( )
   {
    // definir dados
      int avenues,
          streets;

    // obter o tamanho do mundo
      avenues = World.numberOfAvenues( );
      streets    = World.numberOfStreets( );

    // informar o tamanho do mundo
      IO.println ( "World is "
                   + avenues + "x" + streets );
   } // end mapWorld( )
```

```
/**
 *  Acao principal: executar a tarefa descrita acima.
 */
 public static void main( String [ ] args )
 {
  // criar o ambiente
  // OBS.: executar pelo menos uma vez,
  //           antes de qualquer outra coisa
  //          (depois de criado, podera' ser comentado)
     createWorld( "Guia0046.txt" );

  // comandos para tornar o mundo visivel
     World.reset( );                          // limpar configuracoes
     World.setSpeed ( 6 );                    // escolher velocidade
     World.readWorld( "Guia0046.txt" );       // ler configuracao do ambiente

  // definir o objeto particular para executar as acoes (agente)
     Guia0046 JK = new Guia0046( 1, 1, World.EAST, 0 );

   // executar acoes
      JK.mapWorld( );
 } // end main( )
```

22.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

23.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

24.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

25.) Copiar a versão atual do programa para outra (nova) – Guia0047.java.

26.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * metodo para o robot explorar o mundo.
  */
   public void mapWorld( )
   {
    // definir dados
       int avenues, treets;
       int x, y;
        int positions, steps;

    // obter o tamanho do mundo
       avenues = World.numberOfAvenues( );
       streets    = World.numberOfStreets( );
    // informar o tamanho do mundo
       IO.println ( "World is " + avenues + "x" + streets );

     // calcular o numero total de posicoes
        positions = avenues * streets;
        steps = 1;
     // repetir para todas as posicoes no mundo
        while ( steps < positions )
        {
         // mover para a proxima posicao
            move( );
         // testar se e' preciso mudar de linha
            steps = steps + 1;
            if ( steps < positions && steps % 10 == 0 )
            { // ao final de cada linha, testar ...
              if ( steps / 10 % 2 != 0 ) // … e' impar ?
              {                          // subir e virar a esquerda
                turnLeft( );
                move( );
                turnLeft( );
              }
              else                       // e' par ?
              {                          // subir e virar a direita
                turnRight( );
                move( );
                turnRight( );
              } // end if
              steps     = steps + 1;     // passer 'a proxima
            } // end if
         } // end while
    } // end mapWorld( )
```

27.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

28.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

29.) Copiar a versão atual do programa para outra (nova) – Guia0048.java.

30.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * metodo para o robot explorar o mundo.
  */
   public void mapWorld( )
   {
    // definir dados
       int avenues, streets;
       int x, y;
       int positions, steps;

    // obter o tamanho do mundo
       avenues = World.numberOfAvenues( );
       streets    = World.numberOfStreets( );
    // informar o tamanho do mundo
       IO.println ( "World is "
                    + avenues + "x" + streets );
    // calcular o numero total de posicoes
       positions = avenues * streets;
       steps = 1;
    // repetir para todas as posicoes no mundo
       while ( steps < positions )
       {
        // mover para a proxima posicao
          move( );
         // se proximo a um marcador
           if ( nextToABeeper( ) )
           {
            // informar marcador nesta posicao
              IO.println ( "Beeper found after "+steps+" steps" );
           } // end if
        // testar se e' preciso mudar de linha
          steps = steps + 1;
          if ( steps < positions && steps % 10 == 0 )
          { // ao final de cada linha, testar ...
            if ( steps / 10 % 2 != 0 ) // … e' impar ?
            {                          // subir e virar a esquerda
              turnLeft( );
              move( );
              turnLeft( );
            }
            else                       // e' par ?
            {                          // subir e virar a direita
              turnRight( );
              move( );
              turnRight( );
            } // end if
            steps     = steps + 1;      // passer 'a proxima
          } // end if
       } // end while
   } // end mapWorld( )
```

31.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

32.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

33.) Copiar a versão atual do programa para outra (nova) – Guia0049.java.

34.) Realizar as mudanças de versão e
acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * metodo para o robot explorar o mundo.
  */
   public void mapWorld( )
   {
    // definir dados
      int avenues,
          streets;
      int x, y;
      int positions, steps;
      int beepers = 0;

    // obter o tamanho do mundo
      avenues = World.numberOfAvenues( );
      streets    = World.numberOfStreets( );
    // informar o tamanho do mundo
      IO.println ( "World is "
                   + avenues + "x" + streets );
    // calcular o numero total de posicoes
      positions = avenues * streets;
      steps = 1;
    // repetir para todas as posicoes no mundo
      while ( steps < positions )
      {
       // mover para a proxima posicao
          move( );
       // se proximo a um marcador
          if ( nextToABeeper( ) )
          {
           // informar marcador nesta posicao
              IO.println ( "Beeper found after "+steps+" steps" );
            // encontrado mais um marcador
              beepers = beepers + 1;
          } // end if
```

```
      // testar se e' preciso mudar de linha
         steps = steps + 1;
         if ( steps < positions && steps % 10 == 0 )
         { // ao final de cada linha, testar ...
           if ( steps / 10 % 2 != 0 ) // … e' impar ?
           {                          // subir e virar a esquerda
             turnLeft( );
             move( );
             turnLeft( );
           }
           else                       // e' par ?
           {                          // subir e virar a direita
             turnRight( );
             move( );
             turnRight( );
           } // end if
           steps     = steps + 1;      // passer 'a proxima
         } // end if
       } // end while
      // informar quantos marcadores encontrados
        IO.println ( "Found " + beepers + " beeper(s)" );
    } // end mapWorld( )
```

35.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

36.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

37.) Copiar a versão atual do programa para outra (nova) – Guia0050.java.

```
/**
  * metodo para o robot explorar o mundo
  * e fazer um mapa.
  */
   public int [ ] [ ] mapWorld( )
   {
    // definir dados
       int [ ] [ ] map;  // para guardar o mapa

      int avenues,
          streets;
      int positions,
          steps;
      int x, y;
      int beepers = 0;

    // obter o tamanho do mundo
      avenues = World.numberOfAvenues( );
      streets    = World.numberOfStreets( );
    // informar o tamanho do mundo
      IO.println ( "World is "
                   + avenues + "x" + streets );
```

```
 // reservar espaco para o mapa
    map = new int [avenues][streets];

// calcular o numero total de posicoes
   positions = avenues * streets;
   steps = 1;
// repetir para todas as posicoes no mundo
   while ( steps <= positions )
   {
    // mover para a proxima posicao
      move( );
    // se proximo a um marcador
      if ( nextToABeeper( ) )
      {
       // informar marcador nesta posicao
          x = avenue ( )-1;
          y = street    ( )-1;
          IO.println ( "Beeper found at ("+
                    (x+1)+", "+(y+1)+")" );
        // marcar posicao no mapa
          map[y][x] = 1;
       // encontrado mais um marcador
         beepers = beepers + 1;
      } // end if
      if ( steps < positions && steps % 10 == 0 )
      { // ao final de cada linha, testar ...
        if ( steps / 10 % 2 != 0 ) // … e' impar ?
        {                          // subir e virar a esquerda
          turnLeft( );
          move( );
          turnLeft( );
        }
        else                       // e' par ?
        {                          // subir e virar a direita
          turnRight( );
          move( );
          turnRight( );
        }  // end if
      }
      else
      {
       // mover para a proxima posicao
         if ( steps < positions )
         {
           move( );
         }  // end if
      } // end i
    // passar 'a proxima posicao
      steps = steps + 1;
   } // end while
 // retornar mapa
    return ( map );
} // end mapWorld( )
```

```
/**
 * metodo para mover o robot interativamente
 * e guardar a descricao da tarefa.
 */
 public void showMap( int[ ][ ] map )
 {
  // definir dados
     int avenues,
         streets;
     int x, y;

  // testar existencia de mapa
     if ( map != null )
     {
      // obter o tamanho do mundo
        avenues = map.length;
        streets   = map [ 0 ].length;

      // repetir para todas as posicoes no mundo
        for ( y=streets-1; y >= 0; y=y-1 )
        {
          for ( x=0; x<avenues; x=x+1 )
          {
            IO.print ( ""+map [ y ][ x ] );
          }
          IO.println( );
        }
     }
 } // end showMap( )

/**
 *  Acao principal: executar a tarefa descrita acima.
 */
 public static void main( String [ ] args )
 {
  // criar o ambiente
  // OBS.: executar pelo menos uma vez,
  //           antes de qualquer outra coisa
  //          (depois de criado, podera' ser comentado)
    createWorld( "Guia0050.txt" );

  // comandos para tornar o mundo visivel
    World.reset( );                          // limpar configuracoes
    World.setSpeed ( 6 );                    // escolher velocidade
    World.readWorld( "Guia0050.txt" );    // ler configuracao do ambiente

  // definir o objeto particular para executar as acoes (agente)
    Guia0050 JK = new Guia0050( 1, 1, World.EAST, 0 );

  // definir dado
     int [ ] [ ] worldNow;

  // executar acoes
     worldNow = JK.mapWorld( );
     JK.showMap ( worldNow );
 } // end main( )
```

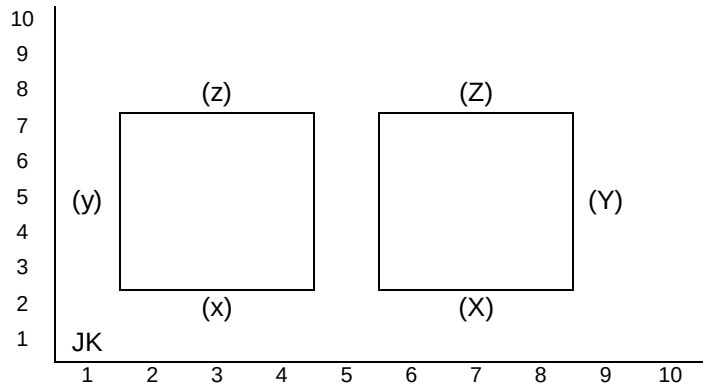
38.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

39.) Executar o programa.
Observar as saídas.
Registrar os resultados com os valores usados para testes.

Exercícios:
DICAS GERAIS: Consultar o Anexo Java para mais informações e outros exemplos.
Prever, realizar e registrar todos os testes efetuados.

Fazer um programa para atender a cada uma das situações abaixo
envolvendo definições e ações básicas.
Os programas deverão ser desenvolvidos em Java usando as classes
disponíveis no JKarel (jkarel.jar).

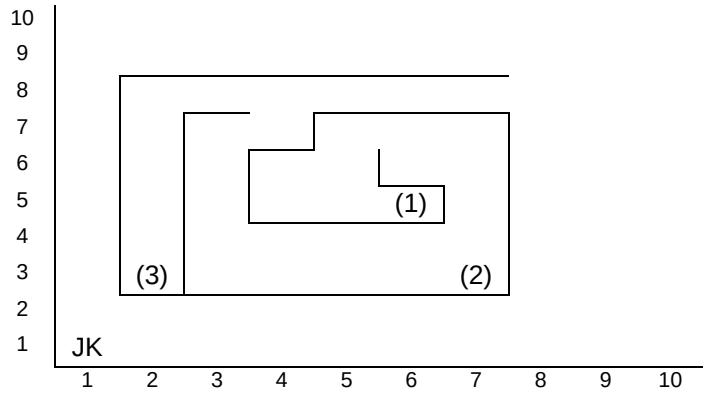
01.) Definir um conjunto de ações em um programa Guia0051 para:
- definir um robô (JK) na posição (1,1), voltado para leste,
  sem marcadores;
- dispor blocos em uma configuração semelhante a dada abaixo:



- definir dois quadrados, lado a lado,
  com uma passagem entre eles;
- definir marcadores em volta do primeiro quadrado,
  um de cada lado;
- tarefa:
  o robô deverá buscar os marcadores (minúsculas),
  e movê-los até as posições indicadas (maiúsculas),
  junto ao segundo quadrado (x-y-z-X-Y-Z);
- restrição:
  o robô deverá passar pelo "corredor"
  entre os quadrados, pelo duas vezes,
  como em um '8' deitado,
  antes de voltar a posição inicial;
  a especificação da tarefa deverá ser feita
  por um arquivo (Tarefa0051.txt);

02.) Definir um conjunto de ações em um programa Guia0051 para:

- definir um robô (JK) na posição (1,1), voltado para leste,
  sem marcadores;
- dispor blocos em uma configuração semelhante a dada abaixo:



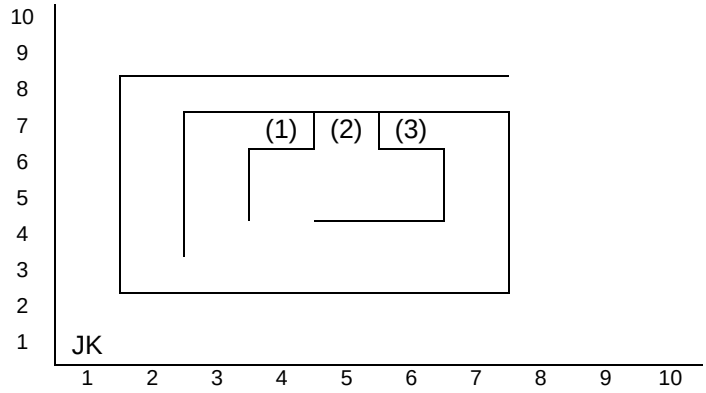
- definir um labirinto com os marcadores indicados
  segundo o modelo acima;
- tarefa:
  o robô deverá buscar os marcadores,
  na ordem indicada pela quantidade,
  e trazê-los à posição inicial;
  a especificação da tarefa deverá ser feita
  por um arquivo (Tarefa0052.txt);
- métodos deverão ser criados para
  ajudar o robô a mover-se no labirinto:

turnAround( ) - virar-se na direção contrária ao movimento
turnAroundCornerLeft( ) - fazer curva fechada à esquerda (“U”)
(acompanhar uma parede interna,
próxima ao ponto com 1 marcador)

DICAS: Inserir novos comandos no método execute( ).

03.) Definir um conjunto de ações em um programa Guia0053 para:

- definir um robô (JK) na posição (1,1), voltado para leste,
  sem marcadores;
- dispor blocos em uma configuração semelhante a dada abaixo:

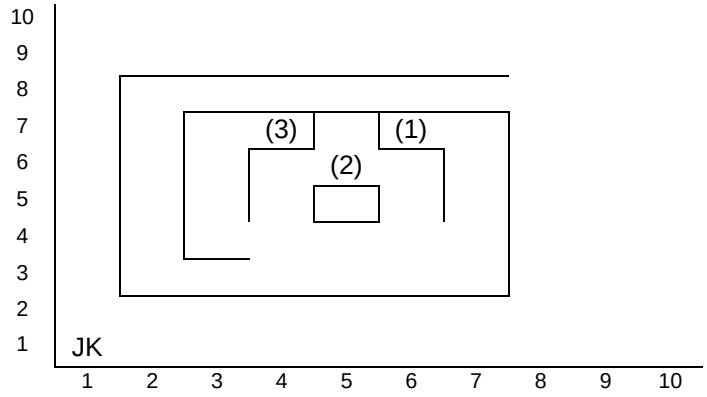


- definir um labirinto com os marcadores indicados,
  segundo o modelo acima;
- tarefa:
  o robô deverá buscar os marcadores,
  na ordem indicada,
  e trazê-los à posição inicial;
  guardar em tabelas separadas as coordenadas (x, y) e
  a quantidade de marcadores recolhidos.

DICAS: Seguir a parede pelo lado direito,
combinando testes nativos
rightIsClear( ) e frontIsClear( ).

04.) Definir um conjunto de ações em um programa Guia0054 para:

- definir um robô (JK) na posição (1,1), voltado para leste,
  sem marcadores;
- dispor blocos em uma configuração semelhante a dada abaixo:

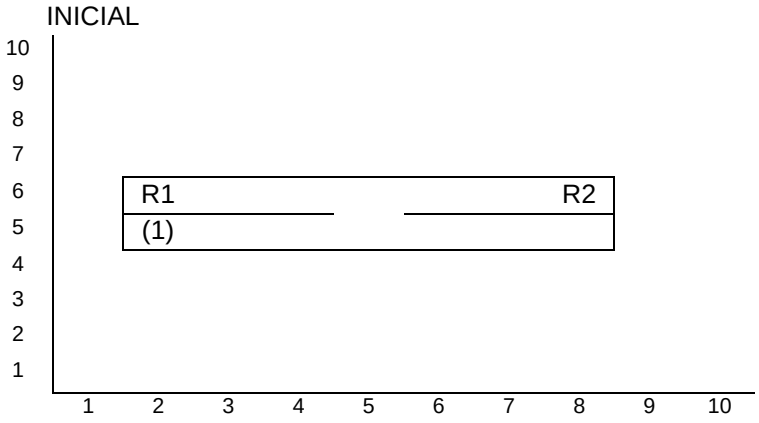


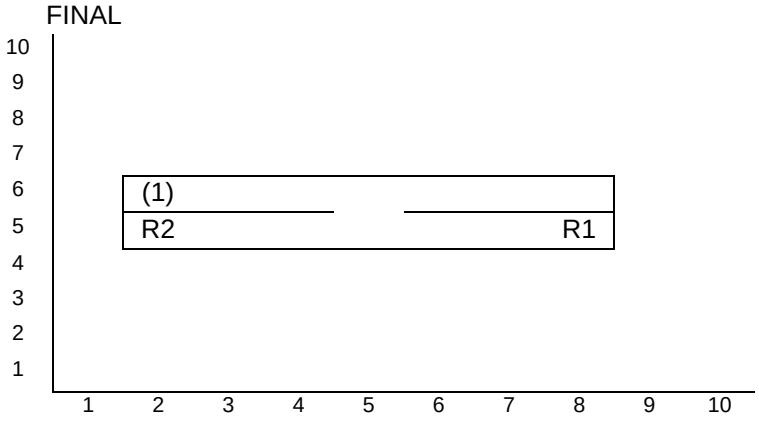
- o robô deverá buscar os marcadores indicados,
  preferencialmente da direita para a esquerda;
- retornar à posição inicial,
  voltar-se para leste e
  desligar-se;
- poderá ser marcado em um mapa o deslocamento efetuado,
  se as posições percorridas forem marcadas ou
  se forem guardadas em arquivo e
  mostradas ao final como o roteiro percorrido.

DICA: Ao mover o robô, colocar uma marca (‘x‘)
na posição correspondente no mapa,
ou gravar as coordenadas (x,y) em arquivo.

05.) Definir um conjunto de ações em um programa Guia0055 para:

- dispor blocos em uma configuração semelhante a abaixo:





- definir dois robôs, conforme indicado na configuração inicial
- definir um marcador na posição indicada, inicialmente;
- definir paredes entre os robôs,
  exceto na metade do caminho;
- tarefa:
  o robô R1 deverá buscar o marcador (1),
  mover-se até a passagem;
  ir à parte de acima,
  aguardar a aproximação de R2,
  e entregar o marcador; depois,
  o robô R2 levará o marcador
  até posição final indicada
  e ambos retornarão às suas
  respectivas posições iniciais;

- dois métodos deverão ser criados:

halfPathRight( ) - andar metade do caminho para a direita
halfPathLeft( ) - andar metade do caminho para a esquerda

- outros métodos deverão ser usados
para a percepção de um robô em relação ao outro,
antes da transferência do marcador:

a.) testar se próximo a outro robô

```
if ( nextToARobot( ) )  // robô (1)
{
  // comandos dependentes da condição
     putBeeper( );        // exemplo
}
else
{
  // comandos dependentes do contrário
} // end if
```

b.) testar se próximo a um marcador

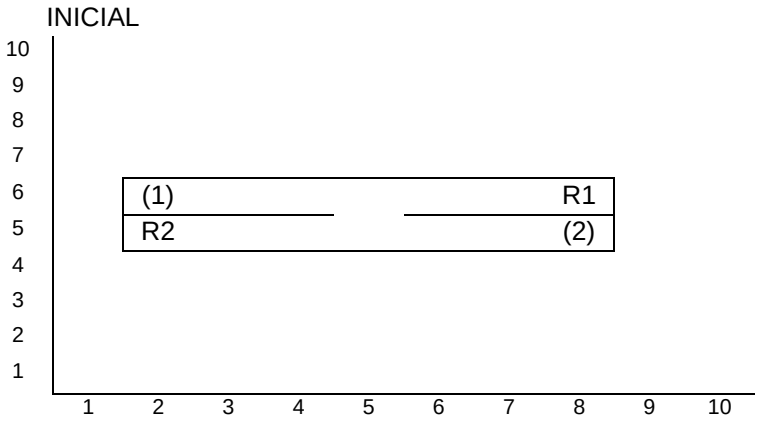
```
if ( nextToABeeper( ) ) // robô (2)
{
  // comandos dependentes da condição
     pickBeeper( );       // exemplo
}
else
{
  // comandos dependentes do contrário
} // end if
```

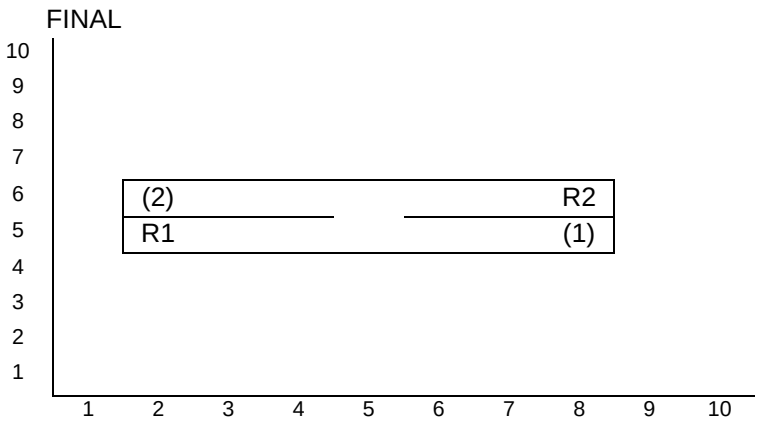
- o robô R1 deverá testar se está próximo
ao outro (ambos na mesma posição);
se estiver, deverá deixar o marcador
para o outro robô (R2) pegar;
então, o segundo deverá testar
se há um marcador disponível
e recolher esse marcador,
antes de completar a tarefa, e
voltar à posição inicial.
Se o primeiro robô chegar à posição combinada
para a entrega, e o outro não estiver lá, deverá
retornar à posição inicial com o marcador.
DICA: Dividir a tarefa em subtarefas.

Tarefa extra

E1.) Dividir as tarefas do último exercício e
distribuí-las em arquivos diferentes,
chamando-os para execução na ordem esperada.

E2.) Definir um conjunto de ações para resolver o seguinte problema:
- dispor blocos em uma configuração semelhante a abaixo:





- onde cada robô deverá buscar os marcadores
de seu respectivo "andar",
irem até o ponto de encontro (metade superior),
trocarem os marcadores, voltar aos seus "andares",
guardar os marcadores recebidos e
retornarem às posições iniciais.

DICA: Um robô só poderá receber marcadores de outro,
se não estiver carregando algum.

Atividade suplementar

Associar os conceitos de representações de dados e a metodologia sugerida para o desenvolvimento de programa (passo a passo), para modificar o modelo proposto (exemplos associados ao JKarel) e introduzir, pouco a pouco, as modificações necessárias, cuidando de realizar a documentação das definições, procedimentos e operações executadas.

Para pensar a respeito

Qual a estratégia de solução ?
Como definir uma classe com um método principal que execute essa estratégia ?
Serão necessárias definições prévias (extras) para se obter o resultado ?
Como dividir os passos a serem feitos e organizá-los em que ordem ?
Que informações deverão ser colocadas na documentação ?
Como lidar com os erros de compilação ?
Como lidar com os erros de execução ?

Fontes de informação

apostila de Java (anexo)
exemplos (0-9) na pasta de arquivos relacionada
bibliografia recomendada
lista de discussão da disciplina
websites

Processo

1 relacionar claramente seus objetivos e
registrar isso na documentação necessária para o desenvolvimento;

2 organizar as informações de cada proposição de problema:
2.1 escolher os armazenadores de acordo com o tipo apropriado;
2.2 realizar as entradas de dados ou definições iniciais;
2.3 realizar as operações;
2.4 realizar as saídas dos resultados;
2.5 projetar testes para cada operação, considerar casos especiais

3 especificar a classe:
3.1 definir a identificação do programa na documentação;
3.2 definir a identificação do programador na documentação;
3.3 definir armazenadores necessários (se houver)
3.4 definir a entrada de dados para cada valor
3.5 testar se os dados foram armazenados corretamente
3.6 definir a saída de cada resultado ou (execução de cada ação)
3.7 testar a saída de cada resultado com valores (situações) conhecidas
3.8 definir cada operação
3.9 testar isoladamente cada operação, conferindo os resultados

4 especificar as ações da parte principal:
4.1 definir o cabeçalho para identificação;
4.2 definir as constantes, armazenadores e dados auxiliares (se houver);
4.3 definir a estrutura básica de programa que possa permitir a
execução de vários dos testes programados;

5. realizar os testes isolados de cada operação e
depois os testes de integração;
5.1 registrar todos os testes realizados.

Dicas

- Digitar os exemplos fornecidos e testá-los.
- Identificar exemplos que possam servir de modelos para os exercícios, e usá-los como sugestões para o desenvolvimento.
- Fazer rascunhos, diagramas e esquemas para orientar o desenvolvimento da solução, previamente, antes de começar a digitar o novo programa.
- Consultar os modelos de programas e documentação disponíveis.
- Anotar os testes realizados e seus resultados no final do texto do programa, como comentários.
- Anotar erros, dúvidas e observações no final do programa, também como comentários.

Conclusão

Analisar cada resultado obtido e avaliar-se ao fim do processo.