PUC-Minas - Ciência da Computação
AED1 – Guia 02

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

01.) Editar e salvar um esboço de programa,
o nome do arquivo deverá ser Guia0021.java,
tal como o nome da classe abaixo,
concordando maiúsculas e minúsculas, sem espaços em branco:

```
/**
  * Guia0021
  *
  * @author
  * @version 01
  */
//
// Lista de dependecias
//
  import jkarel.World;
  import jkarel.Robot;

  import IO.*;

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0021 extends Robot
{
 /**
  * construtor padrao da classe Guia0021.
  * @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 Guia0021( 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 Guia0021( )
```

```
/**
   * 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 marcadores
        World.placeBeepers( 4, 4, 3 ); // em (4,4), tres marcadores

     // 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 mover repetidas vezes.
   * @param vezes para executar
   */
    public void moveN( int vezes )
    {
     // repetir (com teste no inicio)
        while ( vezes > 0 )
        {
          // mover-se uma vez ...
             move ( );
          // ... e descontar uma das ainda por fazer
             vezes = vezes - 1;
        } // end while
    } // end moveN( )
```

```
  /**
   *  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( "Guia0021.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                       // limpar configuracoes
      World.setSpeed ( 7 );                 // escolher velocidade
      World.readWorld( "Guia0021.txt" );    // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0021 JK = new Guia0021( 1, 1, NORTH, 0 );

    // executar acoes repetidas vezes
       JK.moveN( 3 );
   } // 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) – Guia0022.java.

05.) Editar mudanças no nome do programa e versão,
conforme as indicações a seguir,
tomando o cuidado de modificar todas as referências,
inclusive as presentes em comentários.
Incluir na documentação complementar as alterações feitas,
acrescentar indicações de mudança de versão e
prever novos testes.

```
/**
  * Guia0022
  *
  * @author
  * @version 01
  */

//
// Lista de dependecias
//
  import jkarel.World;
  import jkarel.Robot;

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0022 extends Robot
{
 /**
  * construtor padrao da classe Guia0022.
  * @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 Guia0022( 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 Guia0022( )
```

```
/**
   * 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 marcadores
        World.placeBeepers( 4, 4, 3 ); // em (4,4), tres marcadores

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

/**
   * metodo para virar 'a direita (com repeticao).
   */
    public void turnRight( )
    {
     // definir dado local
        int vezes = 1;   // para contar quantas vezes
     // o executor deste metodo
     // devera' virar tres vezes 'a esquerda
     // repetir (com teste no inicio)
        while ( vezes <= 3 )
        {
         // virar uma vez ...
            turnLeft( );
         // ... e contar mais uma feita
            vezes = vezes + 1;
        } // end while
    } // end turnRight( )

/**
   * metodo para mover o robot repetidas vezes.
   */
    public void moveN( int vezes )
    {
     // repetir (com teste no inicio)
        while ( vezes > 0 )
        {
          // mover-se uma vez
            move ( );
          // descontar o movimento feito
            vezes = vezes - 1;
        } // end while
    } // end moveN( )
```

```
  /**
   *  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( "Guia0022.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                          // limpar configuracoes
      World.setSpeed ( 7 );                    // escolher velocidade
      World.readWorld( "Guia0022.txt" ); // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0022 JK = new Guia0022( 1, 1, NORTH, 0 );

    // executar acoes repetidas vezes
      JK.moveN( 3 );
       JK.turnRight( );
       JK.moveN( 3 );
       JK.turnRight( );
       JK.moveN( 3 );
       JK.turnRight( );
       JK.moveN( 3 );
       JK.turnRight( );

   } // end main( )
} // end class

// ---------------------------------------------- testes

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

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.

```
// ---------------------------------------------- testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da repeticao para virar 'a direita
//
```

08.) Copiar a versão atual do programa para outra (nova) – Guia0023.java.

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

```
/**
  * Guia0023
  *
  * @author
  * @version 01
  */

// . . .

/**
  * metodo para mover o robot em um retangulo.
  */
   public void doRectangle( )
   {
    // definir dado local
       int vezes;   // para contar

    // executar acoes repetidas vezes
    // repetir (com teste no inicio e variacao)
       for ( vezes=1; vezes<=4; vezes=vezes+1 )
       {
        // mover-se tres vezes ...
           moveN( 3 );
        // ... e virar 'a direira
           turnRight( );
       } // end for

   } // end doRectangle( )
```

```
 /**
  *  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( "Guia0023.txt" );

   // comandos para tornar o mundo visivel
     World.reset( );                      // limpar configuracoes
     World.setSpeed ( 7 );                // escolher velocidade
     World.readWorld( "Guia0023.txt" ); // ler configuracao do ambiente
     World.showWorld( );                  // mostrar configuracao atual

   // definir o objeto particular para executar as acoes (agente)
     Guia0023 JK = new Guia0023( 1, 1, NORTH, 0 );

   // executar acoes repetidas vezes
     JK.doRectangle( );

   } // end main( )
} // end class

// ------------------------------------------------ testes

/*
// Versao      Teste
// 0.1         01. (      )        teste inicial
// 0.2         01. ( OK )          teste da repeticao para virar 'a direita
*/
```

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.

```
// ------------------------------------------------ testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da repeticao para virar 'a direita
// 0.3         01. ( OK )          teste da repeticao para percorrer um quadrado
//
```

12.) Copiar a versão atual do programa para outra (nova) – Guia0024.java.

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

```
/**
  * Guia0024
  *
  * @author
  * @version 01
  */

// . . .

/**
  * metodo para coletar marcadores.
  */
   public void pickBeepers( )
   {
    // repetir (com teste no inicio)
    // enquanto houver marcador proximo
      while ( nextToABeeper( ) )
      {
        // coletar um marcador
          pickBeeper ( );
      } // end while
   } // end pickBeepers( )

/**
  * metodo para mover o robot em um retangulo.
  */
   public void doRectangle( )
   {
    // definir dado local
      int vezes;   // para contar

    // executar acoes repetidas vezes
    // repetir (com teste no inicio e variacao)
      for ( vezes=1; vezes<=4; vezes=vezes+1 )
      {
       // mover-se tres vezes ...
          moveN( 3 );
        // coletar marcadores, se houver
          pickBeepers( );
       // ... e virar 'a direira
          turnRight( );
      } // end for

   } // end doRectangle( )
```

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) – Guia0025.java.

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

```
/**
  * Guia0025
  *
  * @author
  * @version 01
  */

//
// Lista de dependecias
//
  import jkarel.World;
  import jkarel.Robot;

  import IO.*;          // para entradas e saidas por console

// . . .

/**
  * metodo para coletar marcadores.
  */
   public void pickBeepers( )
   {
    // definir dado local
      int contador = 0;

   // repetir (com teste no inicio)
   // enquanto houver marcador proximo
     while ( nextToABeeper( ) )
     {
       // coletar um marcador
         pickBeeper ( );

        // e contar mais um coletado
          contador = contador + 1;

     } // end while

    // informar quantos foram coletados
      if ( contador > 0 )
      {
       // mostrar quantidade
         IO.println ( "Beepers = "+ contador );
       // pausa
         IO.pause ( "Apertar ENTER para continuar." );
      } // fim se

  } // end pickBeepers( )
```

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) – Guia0026.java.

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

```
/**
  * metodo para mover o robot interativamente.
  */
   public void movel( )
   {
    // definir dados
      int option;

    // apresentar comandos
      IO.println ( );
      IO.println ( "JKarel commands:" );
      IO.println ( );
      IO.println ( "0 - turnOff" );
      IO.println ( "1 - turnLeft          2 - to South" );
      IO.println ( "3 - turnRight         4 - to West " );
      IO.println ( "5 - move              6 - to East "  );
      IO.println ( "7 - pickBeeper        8 - to North" );
      IO.println ( "9 - putBeeper" );
      IO.println ( );

    // repetir (com testes no fim)
    // enquanto opcao diferente de zero
      do
      {

        // ler opcao
          option = IO.readint ( "Command? " );

        // escolher acao dependente da opcao
          switch ( option )
          {
            case 0: // terminar
              // nao fazer nada
             break;
```

```
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;
```

```
            default:// nenhuma das alternativas anteriores
              // comando invalido
                IO.println ( "ERROR: Invalid command." );
            } // end switch
        }
        while ( option != 0 );
    } // end movel( )

   /**
    *  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( "Guia0026.txt" );

     // comandos para tornar o mundo visivel
       World.reset( );                          // limpar configuracoes
       World.setSpeed ( 7 );                    // escolher velocidade
       World.readWorld( "Guia0026.txt" ); // ler configuracao do ambiente

     // definir o objeto particular para executar as acoes (agente)
       Guia0026 JK = new Guia0026( 1, 1, World.EAST, 0 );

     // executar acoes interativamente
       JK.movel( );
    } // end main( )
} // end class

// ---------------------------------------------- testes

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

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) – Guia0027.java.

26.) 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 a opcao de comando
       switch ( option )
       {
         case 0: // terminar
           // nao fazer nada
          break;
         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;
            default:// nenhuma das alternativas anteriores
              // comando invalido
                IO.println ( "ERROR: Invalid command." );
          } // end switch
        } // end execute( )

    /**
      * metodo para mover o robot interativamente.
      */
       public void movel( )
       {
        // definir dados
           int option;
        // apresentar comandos
           IO.println ( );
           IO.println ( "JKarel commands:" );
           IO.println ( );
           IO.println ( "0 - turnOff" );
           IO.println ( "1 - turnLeft          2 - to South" );
           IO.println ( "3 - turnRight         4 - to West " );
           IO.println ( "5 - move              6 - to East "  );
           IO.println ( "7 - pickBeeper        8 - to North" );
           IO.println ( "9 - putBeeper" );
           IO.println ( );
        // repetir (com testes no fim)
        // enquanto opcao diferente de zero
           do
           {
             // ler opcao
                option = IO.readint ( "Command? " );
            // executar commando
                execute ( option );
           }
           while ( option != 0 );
        } // end movel( )
```

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) – Guia0028.java.

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

```
/**
  * metodo para mover o robot interativamente
  * e guardar a descricao da tarefa em arquivo.
  * @param filename - nome do arquivo
  */
  public void learn( String filename )
  {
   // definir dados
     int option;
   // definir arquivo onde gravar comandos
     FILE archive = new FILE ( FILE.OUTPUT, filename );

   // apresentar comandos
     IO.println ( );
     IO.println ( "JKarel commands:" );
     IO.println ( );
     IO.println ( "0 - turnOff" );
     IO.println ( "1 - turnLeft        2 - to South" );
     IO.println ( "3 - turnRight       4 - to West " );
     IO.println ( "5 - move            6 - to East "  );
     IO.println ( "7 - pickBeeper    8 - to North" );
     IO.println ( "9 - putBeeper" );
     IO.println ( );
   // repetir enquanto a quantidade de
   // passos for maior que zero
     do
     {
       // ler opcao
          option = IO.readint ( "Command? " );
       // testar se opcao valida
          if ( 0 <= option && option <= 9 )
          {
            // executar comando
               execute ( option );
            // guardar o comando em arquivo
               archive.println( ""+option );
          } // end if
     }
     while ( option != 0 );
   // fechar o arquivo
   // INDISPENSAVEL para a gravacao
     archive.close( );
  } // end learn( )
```

```
/**
 *  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( "Guia0028.txt" );

  // comandos para tornar o mundo visivel
    World.reset( );                          // limpar configuracoes
    World.setSpeed ( 7 );                    // escolher velocidade
    World.readWorld( "Guia0028.txt" );    // ler configuracao do ambiente

  // definir o objeto particular para executar as acoes (agente)
    Guia0028 JK = new Guia0028( 1, 1, World.EAST, 0 );

  // ensinar sequencia de acoes
  // e grava-las em arquivo
     JK.learn( "Tarefa0001.txt" );
 } // end main( )//
```

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) – Guia0029.java.

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

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

    // repetir enquanto houver dados
       line   = archive.readln ( );   // tentar ler a primeira linha
       while ( ! archive.eof( ) )     // testar se nao encontrado o fim
       {
         // decodificar a linha
            option = IO.getint( line );
         // guardar mais um comando
            execute ( option );
         // tentar ler a proxima linha
            line = archive.readln( ); // tentar ler a proxima linha
       } // end for
    // fechar o arquivo
    // RECOMENDAVEL para a leitura
       archive.close( );
   } // end read( )

 /**
  *  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( "Guia0029.txt" );

   // comandos para tornar o mundo visivel
      World.reset( );                        // limpar configuracoes
      World.setSpeed ( 7 );                  // escolher velocidade
      World.readWorld( "Guia0028.txt" );     // ler configuracao do ambiente

   // definir o objeto particular para executar as acoes (agente)
      Guia0029 JK = new Guia0029( 1, 1, World.EAST, 0 );

   // executar acoes gravadas em arquivo
       JK.read( "Tarefa0001.txt" );
  } // end main( )
```

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) – Guia0030.java.

```
/**
 * metodo para traduzir um comando.
 * @param option - comando a ser traduzido
 */
 public String dictionary( int option )
 {
  // definir dado
     String word = ""; // palavra vazia
  // identificar comando
      switch ( option )
      {
        case 1: // virar para a esquerda
          word = "turnLeft( );";
         break;
        case 2: // virar para o sul
          word = "faceSouth( );";
         break;
        case 3: // virar para a direita
          word = "turnRight( );";
         break;
        case 4: // virar para o oeste
          word = "faceWest( );";
         break;
        case 5: // mover
          word = "move( );";
         break;
        case 6: // virar para o leste
          word = "faceEast( );";
         break;
        case 7: // pegar marcador
          word = "pickBeeper( );";
         break;
        case 8: // virar para o norte
          word = "faceNorth( );";
         break;
        case 9: // colocar marcador
          word = "putBeeper( );";
         break;
      } // end switch
   // retornar palavra equivalente
      return ( word );
 } // end dictionary( )
```

```
/**
  * metodo para receber comandos de arquivo e traduzi-los.
  * @param filename - nome do arquivo
  */
   public void translate( String filename )
   {
    // definir dados
       int  option;
       FILE archive = new FILE ( FILE.INPUT, filename );
       String line;
    // repetir enquanto houver dados
       line   = archive.readln ( );   // tentar ler a primeira linha
       while ( ! archive.eof( ) )      // testar se nao encontrado o fim
       {
         // decodificar a linha
            option = IO.getint( line );
          // tentar traduzir um comando
             IO.println ( dictionary ( option ) );
         // guardar mais um comando
            execute ( option );
         // tentar ler a proxima linha
            line = archive.readln( ); // tentar ler a proxima linha
       } // end for
    // fechar o arquivo
    // RECOMENDAVEL para a leitura
       archive.close( );
   } // end translate( )

 /**
  *  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( "Guia0030.txt" );

   // comandos para tornar o mundo visivel
      World.reset( );                        // limpar configuracoes
      World.setSpeed ( 7 );                   // escolher velocidade
      World.readWorld( "Guia0030.txt" ); // ler configuracao do ambiente

   // definir o objeto particular para executar as acoes (agente)
      Guia0030 JK = new Guia0030( 1, 1, World.EAST, 0 );

   // executar ações
      JK.translate( "Tarefa0001.txt" );
  } // end main( )
```

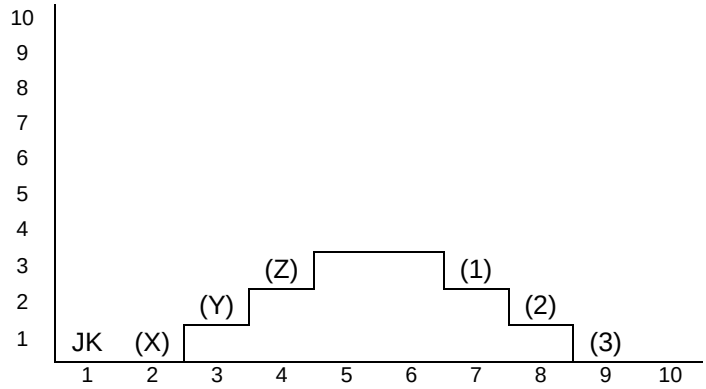
38.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

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 Guia0031 para:
- configurar o mundo semelhante ao diagrama abaixo:




- definir uma "escada" com seis marcadores,
em cada degrau do lado oposto, conforme a figura acima;
- tarefa:
o robô (JK) deverá começar o trajeto ao pé da "escada",
buscar os marcadores, e deixá-los do outro lado
nas posições correspondentes (X,Y,Z); e voltar à posição inicial;
- métodos deverão ser criados e usados
para deslocar um robô na "escada":

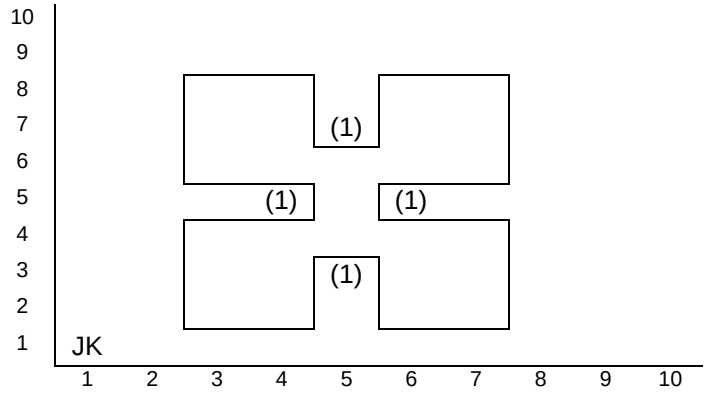
stepUpRight( ) - um degrau para cima e à direita
stepDownRight( ) - um degrau para baixo e à direita
stepUpLeft( ) - um degrau para cima e à esquerda
stepDownLeft( ) - um degrau para baixo e à esquerda.

Exemplo para se descrever um método:

```
/*
   Descricao:
 */
   public void stepUpRight( )
   {
     // acoes para subir um degrau
   } // fim stepUpRight( )
```

02.) Definir um conjunto de ações em um programa Guia0032 para:

- configurar o mundo semelhante ao descrito abaixo:




- definir uma estrutura com quatro marcadores,
  um em cada "nicho";
- tarefa:
  o robô (JK) deverá começar o trajeto abaixo da estrutura,
  buscar os marcadores, no sentido anti-horário,
  e trazê-los à posição inicial;
- métodos deverão ser criados e usar
  as seguintes condições nativas em testes ou repetições:

  rightIsClear( ) - se caminho livre à direita
  leftIsClear( ) - se caminho livre à esquerda
  nextToABeeper( ) - se próximo a um marcador.

  Exemplos de como usar uma condição nativa:

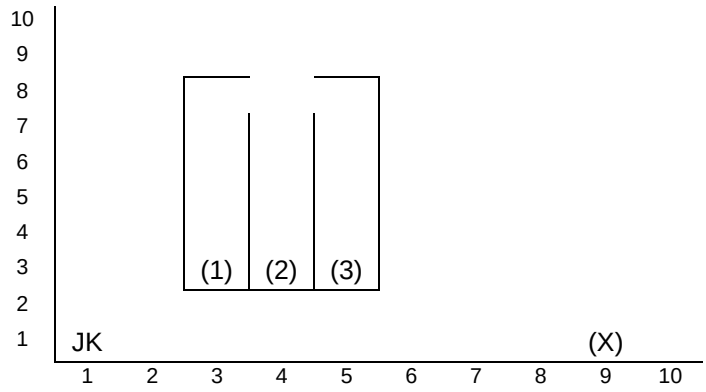
```
// testar se está próximo a um marcador,
// antes de tentar pegá-lo
   if ( nextToABeeper( ) )
   {
      pickBeeper( );
    } // fim se

// testar se poderá virar e mover-se
// para a direita
  if ( rightIsClear( ) )
  {
     turnRight( );
     move( );
  } // fim se
```

DICA: Verificar a possibilidade de pegar mais de um marcador por vez.

03.) Definir um conjunto de ações em um programa Guia0033 para:

- definir um robô (JK) na posição (1,1), voltado para leste,
  sem marcadores
- dispor blocos em uma configuração
  semelhante a mostrada abaixo
- buscar os três marcadores nas posições indicadas




- descarregar todos os marcadores obtidos
  na posição (9,1) mediante um novo método (a ser criado)

  putBeepers( )

  que poderá usar em testes as condições nativas

  anyBeepersInBeeperBag( ) - está portando marcadores?
  areYouHere(x, y) - está na posição (x,y)?

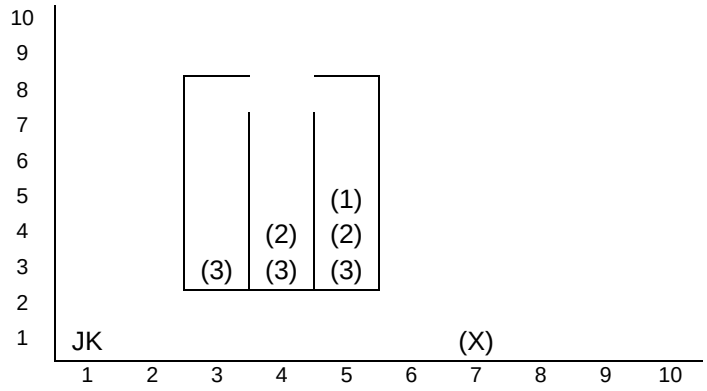
  Exemplos de como usar uma condição nativa:

```
// testar se está em determinada posição
  if ( areYouHere(1,1) )
  {
    move( );
  } // fim se
```

- retornar à posição inicial,
  voltar-se para leste e
  desligar-se.

04.) Definir um conjunto de ações em um programa Guia0034 para:

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




- buscar os marcadores nas posições indicadas, na ordem crescente das quantidades;
- descarregar os marcadores na posição indicada (X);
- retornar à posição inicial, voltar-se para leste e desligar-se;
- todas as posições visitadas pelo robô que tiverem marcadores deverão ser guardadas em arquivo, cujo nome deverá ser Tarefa0034b.txt.

DICA:
As posições poderão ser guardadas quando o robô "**pegar o marcador**".
Para abrir o arquivo e acrescentar conteúdo ao final:

```
FILE arquivo = new FILE ( FILE.APPEND, “Tarefa0034b.txt” );
```

Para obter as coordenadas do robô, definir e usar valores inteiros para guardar as coordenadas:

```
int x, y;

x = avenue( );  // obter posicao atual (avenue)
y = street( );    // obter posicao atual ( street )
```

Ao gravar, colocar cada valor em uma linha:

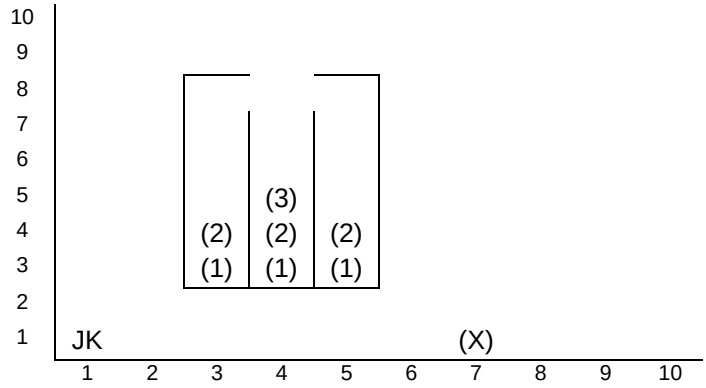
```
arquivo.println ( ""+x );
arquivo.println ( ""+y );
```

Para garantir a gravação, fechar o arquivo:

```
arquivo.close ( );
```

05.) Definir um conjunto de ações em um programa Guia0035 para:

- reutilizar a configuração do problema anterior;
- o robô deverá partir da posição inicial (coluna=1, linha=1),
  voltado para leste e com nenhum marcador;
- buscar os marcadores
  nas quantidades indicadas e
  na ordem crescente das quantidades,
  e descarregar os marcadores na posição indicada (X),
  seguindo os comandos guardados em arquivo;
- retornar à posição inicial,
  voltar-se para o leste e,
  antes de desligar-se,
  reposicionar os marcadores na configuração abaixo:




- todas os códigos das ações necessárias para a execução
  deverão ser primeiro guardados em arquivo
  (por treinamento ou por edição direta),
  cujo nome deverá ser Tarefa0035.txt,
  e depois aplicados mediante leitura.

Tarefa extra

E1.) Definir um método em uma nova classe de robô,
que se possa contar e informar o número de comandos (linhas)
em um arquivo contendo a descrição de uma tarefa.

DICA: Definir um contador
e contar mais uma linha lida,
ao tentar e conseguir ler uma linha (de cada vez).

E2.) Definir um método em uma nova classe de robô,
que se possa ler o número de comandos (linhas)
em um arquivo que descreva uma tarefa e, em seguida,
tentar ler e executar cada um desses comandos.

DICA: Copiar um arquivo contendo a descrição de uma tarefa,
editar esse arquivo para conter, na primeira linha,
a quantidade de comandos nele existentes.

Exemplo:

Arquivo original (TAREFA000.TXT)

5
5
5

Arquivo novo I (TAREFA001.TXT)

**3**
5
5
5

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. Usar /* ... */ para isso.

Conclusão

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