PUC-Minas - Ciência da Computação
AED1 – Guia 01

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

01.) Editar e salvar um esboço de programa,
o nome do arquivo deverá ser Guia0001.java,
tal como o nome da classe abaixo,
concordando maiúsculas e minúsculas, sem espaços em branco:

```
/**
  * Guia0001
  *
  * @author
  * @version 01
  */
//
// Lista de dependencias
//
  import jkarel.World;
  import jkarel.Robot;

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0001 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 Guia0001( 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 Guia0001( )
```

```
/**
 * 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 ); // 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( )

 /**
  *  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( "Guia0001.txt" );

   // comandos para tornar o mundo visivel
      World.reset( );                          // limpar configuracoes
      World.setSpeed ( 7 );                    // escolher velocidade
      World.readWorld( "Guia0001.txt" );    // ler configuracao do ambiente
```

```
    // definir o objeto particular para executar as acoes (agente)
      Guia0001 JK = new Guia0001( 1, 1, World.EAST, 0 );

    // executar acoes
      JK.move( );
      JK.move( );
      JK.turnLeft( );
      JK.move( );
      JK.move( );
      JK.turnLeft( );
      JK.move( );
      JK.move( );
      JK.turnLeft( );
      JK.move( );
      JK.move( );
      JK.turnLeft( );
      JK.turnLeft( );
      JK.turnOff( );
   } // end main( )
} // end class

// ---------------------------------------------- testes
//
// Versao      Teste
// 0.1         01. (     )
//
```

02.) Compilar o programa.
Se houver erros, identificar individualmente a referência para a linha onde ocorrem.
Consultar atentamente o modelo acima na linha onde ocorreu o erro (e também linhas próximas), editar as modificações necessárias.
Compilar novamente e proceder assim até que todos os erros tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
DICA: Se precisar de ajuda sobre como proceder a compilação,
consultar os vídeos com as demonstrações sobre algumas formas para fazê-lo.

SUGESTÃO: Para se acostumar ao tratamento de erros,
registrar a mensagem de erro (como comentário)
e o que foi feito para resolvê-lo.

03.) Executar o programa.
Observar as saídas.
Registrar os resultados.

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

04.) Copiar a versão atual do programa para outra (nova) – Guia0002.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.

```
/**
  * Guia0002
  *
  * @author
  * @version 02
  */

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

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0002 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 Guia0002( 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 Guia0002( )
```

```
/**
 * 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 ); // 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 especificar uma tarefa.
 */
  public void doTask( )
  {
   // especificar acoes da tarefa
      move( );
      move( );
      turnLeft( );
      move( );
      move( );
      turnLeft( );
      move( );
      move( );
      turnLeft( );
      move( );
      move( );
      turnLeft( );
      turnLeft( );
  } // end doTask( )
```

```
/**
 *  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( "Guia0002.txt" );

  // comandos para tornar o mundo visivel
    World.reset( );                      // limpar configuracoes
    World.setSpeed ( 7 );                // escolher velocidade
    World.readWorld( "Guia0002.txt" );   // ler configuracao do ambiente

  // definir o objeto particular para executar as acoes (agente)
    Guia0002 JK = new Guia0002( 1, 1, World.EAST, 0 );

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

// ---------------------------------------------- testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( ___ )         teste de tarefa
//
```

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.

```
// ---------------------------------------------- testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
//
```

08.) Copiar a versão atual do programa para outra (nova) – Guia0003.java.

09.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0003
  *
  * @author
  * @version 03
  */

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

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0003 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 Guia0003( 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 Guia0003( )

 /**
  * 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, 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 especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
      move( );
      move( );
      move( );
      turnLeft( );
  } // end doPartialTask( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doTask( )
  {
   // especificar acoes da tarefa
      doPartialTask( );
      doPartialTask( );
      doPartialTask( );
      doPartialTask( );
      turnLeft( );
  } // end doTask( )
```

```
  /**
   *  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( "Guia0003.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                       // limpar configuracoes
      World.setSpeed ( 7 );                 // escolher velocidade
      World.readWorld( "Guia0003.txt" );    // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0003 JK = new Guia0003( 1, 1, World.EAST, 0 );

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

// ---------------------------------------------- testes
//
// Versao      Teste
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
// 0.3         01. (      )        teste da tarefa parcial
//
```

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.

```
// ---------------------------------------------- testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
// 0.3         01. ( OK )          teste da tarefa parcial
//
```

12.) Copiar a versão atual do programa para outra (nova) – Guia0004.java.

13.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0004
  *
  * @author
  * @version 04
  */

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

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0004 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 Guia0004( 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 Guia0004( )

 /**
  * 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 ); // 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 especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
      move( );
      move( );
      move( );
      turnLeft( );
  } // end doPartialTask( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doTask( )
  {
   // especificar acoes da tarefa
      doPartialTask( );
      doPartialTask( );
      pickBeeper( );              // apanhar marcador
      doPartialTask( );
      doPartialTask( );
      turnLeft( );
  } // end doTask( )
```

```
  /**
   *  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( "Guia0004.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                       // limpar configuracoes
      World.setSpeed ( 7 );                 // escolher velocidade
      World.readWorld( "Guia0004.txt" );    // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0004 JK = new Guia0004( 1, 1, World.EAST, 0 );

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

// ------------------------------------------------ testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
// 0.3         01. ( OK )          teste da tarefa parcial
// 0.4         01. (    )          teste do apanhar marcador
```

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.

```
// ------------------------------------------------ testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
// 0.3         01. ( OK )          teste da tarefa parcial
// 0.4         01. ( OK )          teste do apanhar marcador
//
```

16.) Copiar a versão atual do programa para outra (nova) – Guia0005.java.

17.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0005
  *
  * @author
  * @version 05
  */

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

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0005 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 Guia0005( 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 Guia0005( )

 /**
  * 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 ); // 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 especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
      move( );
      move( );
      move( );
      turnLeft( );
  } // end doPartialTask( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doTask( )
  {
   // especificar acoes da tarefa
      doPartialTask( );
      doPartialTask( );
      pickBeeper( );   // apanhar marcador
      doPartialTask( );
       putBeeper( );                // colocar marcador
      doPartialTask( );
      turnLeft( );
  } // end doTask( )
```

```
  /**
   *  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( "Guia0005.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                      // limpar configuracoes
      World.setSpeed ( 7 );                // escolher velocidade
      World.readWorld( "Guia0005.txt" );   // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0005 JK = new Guia0005( 1, 1, World.EAST, 0 );

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

// ------------------------------------------------ testes
//
// Versao      Teste
// 0.1                01. ( OK )          teste inicial
// 0.2                01. ( OK )          teste da tarefa
// 0.3                01. ( OK )          teste da tarefa parcial
// 0.4                01. ( OK )          teste do apanhar marcador
// 0.5                01. (    )          teste do colocar marcador
//
```

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.

```
// ------------------------------------------------ testes
//
// Versao      Teste
// 0.1                01. ( OK )          teste inicial
// 0.2                01. ( OK )          teste da tarefa
// 0.3                01. ( OK )          teste da tarefa parcial
// 0.4                01. ( OK )          teste do apanhar marcador
// 0.5                01. ( OK )          teste do colocar marcador
//
```

20.) Copiar a versão atual do programa para outra (nova) – Guia0006.java.

21.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0006
  *
  * @author
  * @version 06
  */
//
// Lista de dependencias
//
  import jkarel.World;
  import jkarel.Robot;

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0006 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 Guia0006( 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 Guia0006( )

 /**
  * 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, 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 mover varios passos.
 * @param steps - passos a serem dados.
 */
  public void moveN( int steps )
  {
   // repetir enquanto a quantidade de
   // passos for maior que zero
      while ( steps > 0 )
      {
        // dar um passo
           move( );
        // descontar um passo dado
           steps = steps - 1;
      } // end while
  } // end moveN( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
     moveN( 3 );
      turnLeft( );
  } // end doPartialTask( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doTask( )
  {
   // especificar acoes da tarefa
      doPartialTask( );
      doPartialTask( );
      pickBeeper( );
      doPartialTask( );
      putBeeper( );
      doPartialTask( );
      turnLeft( );
  } // end doTask( )
```

```
  /**
   *  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( "Guia0006.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                       // limpar configuracoes
      World.setSpeed ( 7 );                 // escolher velocidade
      World.readWorld( "Guia0006.txt" );    // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0006 JK = new Guia0006( 1, 1, World.EAST, 0 );

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

// ------------------------------------------------ testes
//
// Versao       Teste
// 0.1                  01. ( OK )          teste inicial
// 0.2                  01. ( OK )          teste da tarefa
// 0.3                  01. ( OK )          teste da tarefa parcial
// 0.4                  01. ( OK )          teste do apanhar marcador
// 0.5                  01. ( OK )          teste do colocar marcador
// 0.6                  01. (    )          teste da repeticao do movimento
//
```

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.) Executar o programa.
Observar as saídas.
Registrar os resultados.

```
// ------------------------------------------------ testes
//
// Versao       Teste
// 0.1                  01. ( OK )          teste inicial
// 0.2                  01. ( OK )          teste da tarefa
// 0.3                  01. ( OK )          teste da tarefa parcial
// 0.4                  01. ( OK )          teste do apanhar marcador
// 0.5                  01. ( OK )          teste do colocar marcador
// 0.6                  01. ( OK )          teste da repeticao do movimento
//
```

24.) Copiar a versão atual do programa para outra (nova) – Guia0007.java.

25.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0007
  *
  * @author
  * @version 07
  */

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

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0007 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 Guia0007( 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 Guia0007( )

 /**
  * 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 ); // 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 mover varios passos.
 * @param steps - passos a serem dados.
 */
  public void moveN( int steps )
  {
   // repetir enquanto a quantidade de
   // passos for maior que zero
      while ( steps > 0 )
      {
        // dar um passo
           move( );
        // descontar um passo dado
           steps = steps - 1;
      } // end while
  } // end moveN( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
      moveN( 3 );
      turnLeft( );
  } // end doPartialTask( )
```

```
/**
  * metodo para especificar parte de uma tarefa.
  */
   public void doTask( )
   {
    // especificar acoes da tarefa
       doPartialTask( );
       doPartialTask( );
    // testar se ha' marcador antes ...
       if ( nextToABeeper( ) )
       {
       // ... de tentar carrega-lo
          pickBeeper( );
       } // end if
       doPartialTask( );
    // testar se carrega marcador antes ...
       if ( anyBeepersInBeeperBag( ) )
       {
       // ... de tentar descarrega-lo
          putBeeper( );
       } // end if
       doPartialTask( );
       turnLeft( );
   } // end doTask( )

  /**
   *  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( "Guia0007.txt" );

    // comandos para tornar o mundo visivel
       World.reset( );                          // limpar configuracoes
       World.setSpeed ( 7 );                    // escolher velocidade
       World.readWorld( "Guia0007.txt" );    // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
       Guia0007 JK = new Guia0007( 1, 1, World.EAST, 0 );

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

```
// ---------------------------------------------- testes
//
// Versao     Teste
// 0.1        01. ( OK )        teste inicial
// 0.2        01. ( OK )        teste da tarefa
// 0.3        01. ( OK )        teste da tarefa parcial
// 0.4        01. ( OK )        teste do apanhar marcador
// 0.5        01. ( OK )        teste do colocar marcador
// 0.6        01. ( OK )        teste da repeticao do movimento
// 0.7        01. (    )        teste com marcador na posicao (4,4)
//            02. (    )        teste sem marcador na posicao (4,4)
//
```

26.) 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.

27.) Executar o programa.
Observar as saídas.
Registrar os resultados.

```
// ---------------------------------------------- testes
//
// Versao     Teste
// 0.1        01. ( OK )        teste inicial
// 0.2        01. ( OK )        teste da tarefa
// 0.3        01. ( OK )        teste da tarefa parcial
// 0.4        01. ( OK )        teste do apanhar marcador
// 0.5        01. ( OK )        teste do colocar marcador
// 0.6        01. ( OK )        teste da repeticao do movimento
// 0.7        01. ( OK )        teste com marcador na posicao (4,4)
//            02. ( OK )        teste com marcador na posicao (4,4)
//
```

28.) Copiar a versão atual do programa para outra (nova) – Guia0008.java.

29.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0008
  *
  * @author
  * @version 08
  */
//
// Lista de dependencias
//
  import jkarel.World;
  import jkarel.Robot;
```

```
/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0008 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 Guia0008( 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 Guia0008( )

 /**
  * 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 );// 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 mover varios passos.
 * @param steps - passos a serem dados.
 */
  public void moveN( int steps )
  {
   // definir dado local
      int step;   // contador de passos
   // repetir enquanto
   // contador nao atingir a quantidade
      step = 1;
      while ( step <= steps )
      {
        // dar um passo
           move( );
        // contar mais um passo dado
           step = step + 1;
      } // end while
  } // end moveN( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
      moveN( 3 );
      turnLeft( );
  } // end doPartialTask( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doTask( )
  {
   // especificar acoes da tarefa
      doPartialTask( );
      doPartialTask( );
   // testar se ha' marcador antes ...
      if ( nextToABeeper( ) )
      {
       // ... de tentar carrega-lo
          pickBeeper( );
      } // end if
      doPartialTask( );
   // testar se a quantidade do que
   // carrega e' maior que zero antes ...
       if ( beepers( ) > 0 )
       {
       // ... de tentar descarregar
          putBeeper( );
       } // end if
      doPartialTask( );
      turnLeft( );
  } // end doTask( )
```

```
  /**
   *  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( "Guia0008.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                    // limpar configuracoes
      World.setSpeed ( 7 );              // escolher velocidade
      World.readWorld( "Guia0008.txt" ); // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0008 JK = new Guia0008( 1, 1, World.EAST, 0 );

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

// ------------------------------------------------ testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
// 0.3         01. ( OK )          teste da tarefa parcial
// 0.4         01. ( OK )          teste do apanhar marcador
// 0.5         01. ( OK )          teste do colocar marcador
// 0.6         01. ( OK )          teste da repeticao do movimento
// 0.7         01. ( OK )          teste com marcador na posicao (4,4)
//             02. ( OK )          teste com marcador na posicao (4,4)
// 0.8         01. (    )          teste com a quantidade de marcadores
//
```

30.) 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.

31.) Executar o programa.
Observar as saídas.
Registrar os resultados.

```
// ---------------------------------------------- testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
// 0.3         01. ( OK )          teste da tarefa parcial
// 0.4         01. ( OK )          teste do apanhar marcador
// 0.5         01. ( OK )          teste do colocar marcador
// 0.6         01. ( OK )          teste da repeticao do movimento
// 0.7         01. ( OK )          teste com marcador na posicao (4,4)
//              02. ( OK )          teste com marcador na posicao (4,4)
// 0.8         01. ( OK )          teste com a quantidade de marcadores
//
```

32.) Copiar a versão atual do programa para outra (nova) – Guia0009.java.

33.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0009
  *
  * @author
  * @version 09
  */

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

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0009 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 Guia0009( 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 Guia0009( )
```

```
/**
 * 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 ); // 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 mover varios passos.
 * @param steps - passos a serem dados.
 */
  public void moveN( int steps )
  {
    // definir dado local
       int step;                         // contador de passos
    // repetir para contador
    // comecando em 1,
    // enquanto menor ou igual 'a quantidade,
    // variando de 1 em 1
       for ( step = 1; step <= steps; step = step + 1 )
       {
         // dar um passo
            move( );
       } // end for
  } // end moveN( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
      moveN( 3 );
      turnLeft( );
  } // end doPartialTask( )
```

```
 /**
  * metodo para especificar parte de uma tarefa.
  */
   public void doTask( )
   {
    // especificar acoes da tarefa
       doPartialTask( );
       doPartialTask( );
    // testar se ha' marcador antes ...
       if ( nextToABeeper( ) )
       {
        // ... de tentar carrega-lo
           pickBeeper( );
       } // end if
       doPartialTask( );
    // testar se a quantidade do que
    // carrega e' maior que zero antes ...
       if ( beepers( ) > 0 )
       {
        // ... de tentar descarregar
           putBeeper( );
       } // end if
       doPartialTask( );
       turnLeft( );
   } // end doTask( )

  /**
   *  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( "Guia0009.txt" );

    // comandos para tornar o mundo visivel
       World.reset( );                      // limpar configuracoes
       World.setSpeed ( 7 );                // escolher velocidade
       World.readWorld( "Guia0009.txt" );   // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
       Guia0009 JK = new Guia0009( 1, 1, World.EAST, 0 );

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

```
// ---------------------------------------------- testes
//
// Versao     Teste
// 0.1        01. ( OK )          teste inicial
// 0.2        01. ( OK )          teste da tarefa
// 0.3        01. ( OK )          teste da tarefa parcial
// 0.4        01. ( OK )          teste do apanhar marcador
// 0.5        01. ( OK )          teste do colocar marcador
// 0.6        01. ( OK )          teste da repeticao do movimento
// 0.7        01. ( OK )          teste com marcador na posicao (4,4)
//            02. ( OK )          teste com marcador na posicao (4,4)
// 0.8        01. ( OK )          teste com a quantidade de marcadores
// 0.9        01. (    )          teste com outra forma de repeticao
//
```

34.) 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.

35.) Executar o programa.
Observar as saídas.
Registrar os resultados.

```
// ---------------------------------------------- testes
//
// Versao     Teste
// 0.1        01. ( OK )          teste inicial
// 0.2        01. ( OK )          teste da tarefa
// 0.3        01. ( OK )          teste da tarefa parcial
// 0.4        01. ( OK )          teste do apanhar marcador
// 0.5        01. ( OK )          teste do colocar marcador
// 0.6        01. ( OK )          teste da repeticao do movimento
// 0.7        01. ( OK )          teste com marcador na posicao (4,4)
//            02. ( OK )          teste com marcador na posicao (4,4)
// 0.8        01. ( OK )          teste com a quantidade de marcadores
// 0.9        01. ( OK )          teste com outra forma de repeticao
//
```

36.) Copiar a versão atual do programa para outra (nova) – Guia0010.java.

37.) Acrescentar ao programa as modificações indicadas abaixo:

```
/**
  * Guia0010
  *
  * @author
  * @version 10
  */
/**
  * Guia0010
  *
  * @author
  * @version 10
  */
```

```
//
// Lista de dependencias
//
  import jkarel.World;
  import jkarel.Robot;

/**
 * Exemplo de programa para uso com a classe JKarel.
 */
public class Guia0010 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 Guia0010( 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 Guia0010( )

 /**
  * 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 );  // 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 mover varios passos.
 * @param steps - passos a serem dados.
 */
  public void moveN( int steps )
  {
   // definir dado local
      int step;   // contador de passos
   // repetir para contador
   // comecando em 1,
   // enquanto menor ou igual 'a quantidade,
   // variando de 1 em 1
      for ( step = 1; step <= steps; step = step + 1 )
      {
        // dar um passo
           move( );
      } // end for
  } // end moveN( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doPartialTask( )
  {
   // especificar acoes dessa parte da tarefa
      moveN( 3 );
      turnLeft( );
  } // end doPartialTask( )

/**
 * metodo para especificar parte de uma tarefa.
 */
  public void doTask( )
  {
   // especificar acoes da tarefa
      doPartialTask( );
      doPartialTask( );
   // testar se ha' marcador antes ...
       if ( nextToABeeper( ) )
       {
        // ... de tentar carrega-lo
           pickBeeper( );
           doPartialTask( );
        // ... e tentar descarregar
           putBeeper( );
       }
       else
       {
           doPartialTask( );
       } // end if
      doPartialTask( );
      turnLeft( );
  } // end doTask( )
```

```
  /**
   *  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( "Guia0010.txt" );

    // comandos para tornar o mundo visivel
      World.reset( );                    // limpar configuracoes
      World.setSpeed ( 7 );              // escolher velocidade
      World.readWorld( "Guia0010.txt" );  // ler configuracao do ambiente

    // definir o objeto particular para executar as acoes (agente)
      Guia0010 JK = new Guia0010( 1, 1, World.EAST, 0 );

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

// ------------------------------------------------ testes
//
// Versao      Teste
// 0.1         01. ( OK )          teste inicial
// 0.2         01. ( OK )          teste da tarefa
// 0.3         01. ( OK )          teste da tarefa parcial
// 0.4         01. ( OK )          teste do apanhar marcador
// 0.5         01. ( OK )          teste do colocar marcador
// 0.6         01. ( OK )          teste da repeticao do movimento
// 0.7         01. ( OK )          teste com marcador na posicao (4,4)
//             02. ( OK )          teste com marcador na posicao (4,4)
// 0.8         01. ( OK )          teste com a quantidade de marcadores
// 0.9         01. ( OK )          teste com outra forma de repeticao
// 1.0         01. (      )        teste com outra forma de alternativa
//
```

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.

```
// ---------------------------------------------- testes
//
// Versao     Teste
// 0.1        01. ( OK )        teste inicial
// 0.2        01. ( OK )        teste da tarefa
// 0.3        01. ( OK )        teste da tarefa parcial
// 0.4        01. ( OK )        teste do apanhar marcador
// 0.5        01. ( OK )        teste do colocar marcador
// 0.6        01. ( OK )        teste da repeticao do movimento
// 0.7        01. ( OK )        teste com marcador na posicao (4,4)
//            02. ( OK )        teste com marcador na posicao (4,4)
// 0.8        01. ( OK )        teste com a quantidade de marcadores
// 0.9        01. ( OK )        teste com outra forma de repeticao
// 1.0        01. ( OK )        teste com outra forma de alternativa
//
```

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 Guia0011 para:
- o robô partir da posição inicial (coluna=1, linha=1), voltado para leste,
  com três marcadores ("beepers");
- o robô deverá colocar um marcador nas posições indicadas:
  (6,6), (3,6) e (3,3), nessa ordem;
- retornar à posição inicial,
  voltar-se para o leste. e
  desligar-se.

OBS.: Para fazer o robô começar com marcadores, rever sua definição inicial:

Guia0010 JK = new Guia0010(1, 1, World.EAST, **0**);

02.) Definir um conjunto de ações em um programa Guia0012 para:
- configurar o mundo para conter
  inicialmente três marcadores ("beepers")
  nas posições anteriormente indicadas:
  (3,3), (6,3) e (6,6) nessa ordem;
- o robô deverá partir da posição inicial (coluna=1, linha=1),
  voltado para leste e nenhum marcador;
- buscar os marcadores nas posições indicadas,
  na ordem inversa à qual foram colocados;
- retornar à posição inicial, voltar-se para o leste e desligar-se.

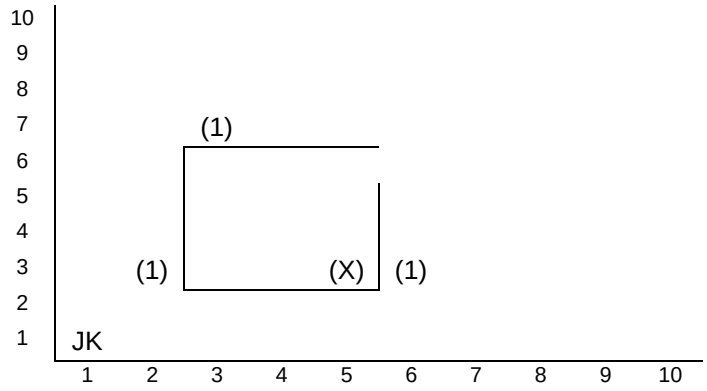
OBS.: Para colocar desde o início os marcadores nas posições indicadas,
rever o método *createworld (filename):*

World.placeBeepers (avenue, street, howMany);

03.) Definir um conjunto de ações em um programa Guia0013 para:
- o robô deverá partir da posição (coluna=1, linha=1), voltado para leste e
  sem marcadores
- buscar os marcadores ("beepers")
  nas mesmas posições iniciais do problema anterior
  (configurar o mundo com marcadores nas posições)
- descarregar todos os marcadores obtidos
  nas posições (2,1); (3,1) e (4,1), respectivamente;
- retornar à posição inicial, voltar-se para leste e desligar-se.

04.) Definir um conjunto de ações em um programa Guia0014 para:

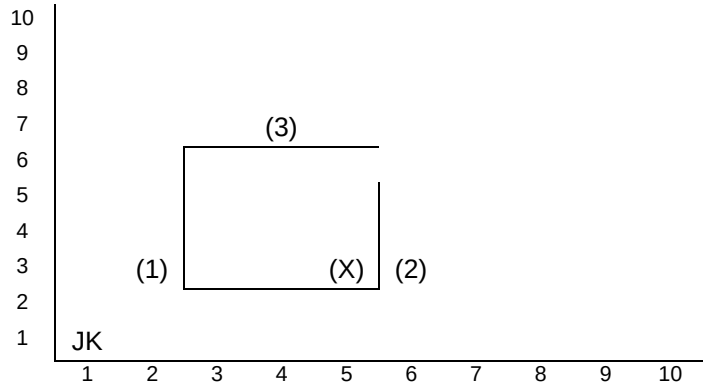
- configurar o mundo semelhante ao diagrama abaixo
  inicialmente com três marcadores nas posições indicadas (1):



- o robô deverá partir da posição inicial (coluna=1, linha=1),
  voltado para leste e com nenhum marcador;
- buscar os três marcadores ("beepers")
  e colocá-los na posição indicada por (X);
- retornar à posição inicial,
  voltar-se para o leste e desligar-se.

05.) Definir um conjunto de ações em um programa Guia0015 para:

- configurar o mundo semelhante ao diagrama abaixo
  inicialmente com seis marcadores
  na posição indicada (X):



- o robô deverá partir da posição inicial (coluna=1, linha=1),
  voltado para leste e com nenhum marcador;
- buscar os marcadores e distribuí-los
  nas quantidades indicadas e
  na ordem crescente das quantidades (1-2-3)
- retornar à posição inicial, voltar-se para o leste e desligar-se.

Tarefas extras

E1.) Definir um método em uma nova classe de robô,
que possa mover-se certo número de passos (*steps*)
de uma só vez.

```
public void moveN ( int steps )
{
  // incluir comandos extras
} // end moveN ( )
```

Para isso será necessário realizar um teste para verificar
se o número de passos é valor válido (maior que zero),
o que pode ser feito usando-se o teste

```
if ( steps > 0 )
{
  // mover-se uma vez
     (inserir o comando para mover)
  // e repetir o procedimento
     (inserir o comando para chamar de novo)
} // end if
```

Para repetir o método novamente,
esse deverá ser chamado com quantidade de passos
menor (*steps*-1) que a anteriormente usada (*steps*)

```
moveN ( steps - 1 );
```

Testar o método mediante substituição de vários usos consecutivos
de chamadas ao método de deslocamento unitário,
por apenas uma chamada desse novo método
com a quantidade de passos equivalentes.

E2.) Definir métodos em uma nova classe de robô,
para que se possa colocar e pegar vários marcadores (n)
de uma só vez.

```
public void putBeepers ( int n )
{
   // incluir comandos extras
} // end putBeepers ( )

public void pickBeepers ( int n )
{
   // incluir comandos extras
} // end putBeepers ( )
```

Testar o método mediante substituição de vários usos consecutivos
de chamadas aos métodos de posicionamento unitário,
por chamadas desses novos métodos.

DICA: Rever a definição do exercício anterior para usar repetições.

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.