```java
public class RB {
  /**
  * Helper Class for separating the potential elements of a command more easily
  */
  private class Command {
    // possible parameters of a command
    // the command itself
    String command = "";
    // for & endfor : register or jump to line
    int param1 = -1;
    // for : state of iterator
    int param2 = -1;
    public Command(String[] command){
      this.command = command[0];
      // if one parameter given, else if two parameters given
      if(command.length == 2){
          this.param1 = Integer.parseInt(command[1]);
      } else if(command.length == 3){
          this.param1 = Integer.parseInt(command[1]);
          this.param2 = Integer.parseInt(command[2]);
      }
    }
    /**
    * Validator for true RB commands
    * @return: boolean : true for RB commands, false for everything else, like loop commands
    */
    public boolean isTrueRBCommand(){
      switch(command){
          case "move" : case "turnLeft" : case "turnRight" : case "pickUp" : return true;
          default: return false;
      }
    }
    /**
    * Getter for parameter 1
    */
    public int p1(){
      return this.param1;
    }
    /**
    * Getter for parameter 2
    */
    public int p2(){
      return this.param2;
    }
    /**
    * Getter for command
    */
    public String c(){
      return this.command;
    }
  }
  // program cache
  private String[] programmspeicher;
  // limit of commands
  private int anzahlKommandos;
  // current state
  private int programmposition;
  // loop register
  private int[] register;

  /**
  * most rudimental constructor for RB. All initial variables are set without any parameters given
  */
  public RB(){
    programmspeicher = null;
    anzahlKommandos = 0;
    programmposition = 0;
    register = null;
  }

  /**
  * most complex constructor for RB. Program and maximum of commands can be set by calling this constructor
  */
  public RB(String[] program, int kommandos){
    anzahlKommandos = kommandos;
    this.programmHochladen(program);
  }

  /**
  * more complex constructor for RB. Program can be set by calling this constructor
  */
  public RB(int kommandos){
    programmspeicher = null;
    anzahlKommandos = kommandos;
    programmposition = 0;
    register = null;
  }

  /**
  * prgrammHochladen
  * @param String[] : a String array of program lines
  * @return int : the number of true RB commands found in the given program, necessary greater or equal to 0
  * if return is smaller than 0, the function found an error in the program given.
  */
  public int programmHochladen(String[] programm){
    // Initialise register
    this.register = new int[2];
    // Initialise program state index
    this.programmposition = 0;
    // Initialise counter for RB commands
    int bewegungsbefehle = 0;
    // for each loop through program array
    for(String zeile : programm){
        // split each line into its words, e.g. "for 0 2" -> {"for", "0", "2"};
        Command command = new Command(zeile.split(" "));
        // decide on first word / command
        switch(command.c()){
          // true RB command:
            case "move" : case "turnLeft" : case "turnRight" : case "pickUp" :
            bewegungsbefehle++;
          // break and return error code when counter reaches max amount of commands
          if(bewegungsbefehle > this.anzahlKommandos){
            return -2;
          }
          break;
        // loop head detected
        case "for" :
          // Check for parameters to be 2 individual numbers and int values, if not, break and throw a syntax error
          if(!Integer.class.isInstance(command.p1()) || !Integer.class.isInstance(command.p2())){
            return -1;
          }
          break;
        // end of loop keyword detected
        case "endfor" :
          // check for parameters to be one int value
          if(!Integer.class.isInstance(command.p1())){
            return -1;
          }
          break;
        // everything else is also not definded and throws a syntax error
        default : return -1;
      }
    }
    // set program cache to the given program
    programmspeicher = programm;
    // return count of true RB commands
    return bewegungsbefehle;
  }
  /**
  * Worker service for RB to complete a level
  * @return String : current true RB command or 'end' command
  */
  public String schritt(){
    // initial command
    Command command = this.currentCommand();
    // loop while command is not true RB command : not really necessary, due to loop break on command found but a primitive loop condition
    while(!command.isTrueRBCommand()){
      // if end of program reached, return "end" directly from here
      if(programmposition >= programmspeicher.length){
        return "end";
      } else {
        // get current command
        command = this.currentCommand();
        // loop breal on true RB command - as defined
        if(command.isTrueRBCommand()){
            break;
        }
        // endfor found : jump back to loop head given by parameter 1
        if(command.c().equals("endfor")){
          programmposition = command.p1();
        } else if (command.c().equals("for")){
          // for found :
          // save position of loop in program (later used)
          int loopPosition = programmposition;
          // current state of loop saved in register
          int registereintrag = register[command.p1()];
          // if loop not finished by condition
          if(command.p2() > registereintrag){
            // step into loop
            programmposition++;
            // increase the loop's iterator
            register[command.p1()] = registereintrag + 1;
          } else {
            // loop finished!
            // find endfor fitting to finished loop and step to it. use loopPosition to determine the origin of endfor
            while(!(currentCommand().c().equals("endfor")) || !(command.p1() == loopPosition)){
              programmposition++;
            }
            // step to element after endfor
            programmposition++;
            // reset register entry for loop
            register[command.p1()] = 0;
          }
        }
      }
    }
    // Increase position by 1 to the next element to be executed
    programmposition++;
    // return command found
    return command.c();
  }
  /**
  * Getter for maximum potential commands RB can take
  */
  public void anzahlKommandos(int kommandos){
    this.anzahlKommandos = kommandos;
  }
  /**
  * Getter for the Command at the current state of the program
  */
  private Command currentCommand(){
    return new Command(programmspeicher[programmposition].split(" "));
  }
}
```