Method Design & Stepwise Refinement in Alice

As we have previously discussed, design is the process where we plan our programs without using a programming language. A design is not concerned with the details of how a particular programming language looks (called the syntax of the language). Instead, good design focuses on the logical steps required to solve any particular problem.

Two common tools for design are pseudocode and flowcharts. Pseudocode somewhat resembles programming code, except it uses plain English phrases instead of specific syntax. A flowchart is a diagram that represents the logic for the solution to the problem.

Whether using pseudocode or flowcharts, a design usually starts with a very broad description of a solution. As the design develops, you will refine the design to add additional steps, correct errors, and continue to improve the detail in the design. This process is known as stepwise refinement.

For example, consider one of the classes from the People gallery in Alice. Suppose we want to have a person walk in place (i.e., they step up and down, but don't move anywhere) as part of an exercise routine. Our first attempt at a design (using pseudocode) might be very limited:

```
step up
step down
```

This is obviously inadequate, since we haven't even mentioned the left and right legs.

```
right leg up
left leg down
left leg up
right leg down
```

With Alice, we have previously used the concepts of Do together and Do in order. We don't often have to specify Do in order (since that is the default behaviour), but Do together is important, and applies to pseudocode as well.

```
Do together
    left leg up
    right leg down

Do together
    right leg up
    left leg down
```
Method Design & Stepwise Refinement in Alice

Notice that our first block of code has the right leg moving down, which assumes it is already raised. Similarly, at the end of the code, the right leg is left up.

```
right leg up
Do together
   left leg up
   right leg down
Do together
   right leg up
   left leg down
right leg down
```

From our experience with Alice, we know that animation can be quite complicated, and that there aren't primitive methods to perform "right leg up" and "left leg up". Instead, we will have to design, and then write, the code for this.

At this point, we use another element of design, which is to keep our methods from becoming overly complicated. As methods get longer, they become more difficult to understand, maintain, and modify. In order to control the growth of our methods, we add additional methods to perform smaller tasks within the larger task.

```
call the rightLegUp method
Do together
   call the leftLegUp method
   call the rightLegDown method
Do together
   call the rightLegUp method
   call the leftLegDown method
call the rightLegDown method
```

For each of the new methods, we will need to move the entire leg, but also control the bending of the knee (to move the lower part of the leg). For example, rightLegUp might look like:

```
rotate right leg upwards ¼ turn
rotate right knew downwards ¼ turn
```
1. Copy the WorkOut world from the Handout folder to your network drive.

2. Because of the way Alice is structured, we will have to write the lowest-level methods first, which are the individual leg movements. For the randomGirl2 object, select the methods tab and create a new method called rightLegUp. The whole leg will kick forward by \( \frac{1}{4} \) revolutions, while the leg below the knee (i.e., the shin) will turn backward by \( \frac{1}{4} \) revolution.

3. Complete the rightLegDown, leftLegUp, and leftLegDown methods.

4. Now incorporate all of the methods into a higher-level method called runInPlace, which is based on our earlier design.
Method Design & Stepwise Refinement in Alice

5. So far we have only created methods, so nothing will happen if we try to run the animation. It is necessary to call the method runInPlace from the main world method (in this case, called world.animation). Add a call to runInPlace to world.animation.

<table>
<thead>
<tr>
<th>world.animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>world.animation No parameters</td>
</tr>
</tbody>
</table>

6. Save your world (we will use this world again) and play the animation.