**CodeWorldsV2 Project Spec**

## Overview

For this project, you'll fill in the missing pieces from the CodeWorldsV2 source code that is supplied with the project. Be sure to first view the lectures for the CodeWorlds module, and do the ILQs. The project has Basic, Bronze, Silver and Gold levels, each described below

## Basic Level: Non-hierarchical Implementation

For the Basic level, implement the world-reading and displaying functions of CodeWorldsV2. But importantly, do so only for one level of hierarchy.

Specifically, this means that the input will not have patterns, and that any CompositeBody object you create will have only Brick members -- no CompositeBody will include other CompositeBodys. Here are the tasks:

### Task 1 Write CompositeBody

CompositeBody was described in the lectures, but the supplied CodeWordsV2 code includes no implementation. Write a simple one, leaving methods getBounds, clone, and iterator stubbed out for now. Add a member datum to hold references to child objects. (You could use Brick references, but you'll need to change to Body references in the Silver version, so use Body references, but assume you can downcast them to Bricks.)

### Task 2 Implement getBounds and clone

Of the three Body methods needing implementation, getBounds and clone are easier than iterator. Leave stub methods for the latter in CompositeBody and Brick, and implement getBounds and clone.

Note that getBounds must be implemented by computation and not, for instance, by simply passing a boundary value in a constructor, saving it as a member and returning it. In future versions, getBounds may even return different values at different times for the same Body, as movement occurs in the simulation. So it must be computed when called.

Implementing CompositeBody.clone is optional at this level; there will be no need for it, though there will be for Brick.clone. Leave it stubbed if you like.

### Task 3 Implement InputStreamWorldFactory

Implement reading of worlds in InputStreamWorldFactory, but without patterns. Just read a list of built in items like Cow, Horse, etc.

Do this by keeping a table of model items, keyed by name. When you read a line of input specifying, say, Cow, look the model item up in the table, and clone it, per the Prototype pattern. This is a bit elaborate when all you're reading and creating are Bricks, but you'll find this design very helpful when you move to reading patterns in the Silver version.

### Task 4 Implement iterator for Brick and CompositeBody

The supplied CodeWorlds.main function must work without alteration. You'll need to implement iterator for CompositeBody in order for the loop in that main function to work. This is a slightly complex task, and will require an inner class of CompositeBody. But, per usual for this version, assume the only children of a CompositeBody are Bricks.

Once you have tasks 1 - 4 complete, check that your application works like the CodeWorldsV2 version in the lectures, and in the Jar supplied with the project, again assuming no patterns in the input file.

## Bronze Level: Add Creative Elements

The Bronze level offers a chance to do some creative work.

### Task 1

Implement the getImage methods in Horse and Sloth to provide better images. Build something at least as sophisticated as the existing Cow and Tree images, and use their code as an example.

### Task 2

Write an AutoWorldFactory class that implements WorldFactory, creating a world automatically. Add a new first commandline option of 'A', and use your AutoWorldFactory to create the world if that option is specified. You may create any world you like, though note that at this level you'll be limited to using individual Bricks and not patterns. Don't put too much work into it now, if you plan to complete the Silver level, since the patterns added in Silver will make much more complex worlds possible.

## Silver Level: Hierarchical Implementation

For the Silver level, make the implementation of InputStreamWorldFactory and CompositeBody fully hierarchical. Specifically, this means that CompositeBody may have other CompositeBodys as children.

### Task 1 Implement getBounds and clone

As was true in the Basic level, getBounds and clone are easier than iterator. Implement them first. getBounds will need to recursively compute bounding rectangles, but if you implemented it properly in the Basic level, you may find this happens automatically. You'll need to implement clone for CompositeBody, but again if you did this in the Basic level, you may already have Silver Task 1 done.

### Task 2 Implement InputStreamWorldFactory

Add processing and use of patterns in InputStreamWorldFactory. Some suggestions:

A. Note that the code needed to read a pattern like Forest or Stand is the same as that needed to read the world contents at the end of the file. Treating each such pattern as a "subworld" is a good design.

B. Extend the name-keyed table you build in the Basic level to allow entries that are CompositeBodys, representing patterns, as well as Bricks representing built in entities. Continue to clone via the Prototype design pattern, since the same CodeWorlds pattern may appear in many different spots (e.g. Forests might appear in many location in the world)

### Task 3 Implement iterator for Brick and CompositeBody

Extend iterator for CompositeBody to allow CompositeBodys as children. This is a pretty sophisticated task. You'll need to treat each child as a Body, not simply assume it's a Brick. I found it necessary to have two iterators within the CompositeBody iterator, and also to have a simple iterator for Brick itself.

## Gold Level: AutoWorldFactory Challenge

For the Gold level, improve your AutoWorldFactory class to make use of patterns, and to build as sophisticated a world as possible within a limited body of code.

Your AutoWorldFactory and any supporting code you add to go with it, may be at most one page long, following standard coding style. The challenge is not to merely create a world automatically, but to create the most interesting one you can with just one page of code. Your extra credit grade may be partly based on sophisticated use of loops, hierarchical structures, etc. and on the complexity of the world you create in one page.