Alisha Adam & Rohit Talreja
CS 106A – Summer 2016

Solutions for Section #1

Based on a handout by Eric Roberts

```
/*
 * File: UnitedNationsKarel.java
 * -----------------------------
 * The UnitedNationsKarel subclass builds houses at corners
 * marked by rubble.
 */

import stanford.karel.*;

public class UnitedNationsKarel extends SuperKarel {

   public void run() {
      while (frontIsClear()) {
         if(beepersPresent()) {
            pickBeeper();
            backup();
            buildHouse();
         }
         if(frontIsClear()) {
            move();
         }
      }
   }

/**
 * Builds a beeper house on stilts.
 * Precondition: Karel facing East at bottom of left stilt
 * Postcondition: Karel facing East at bottom of right stilt
 */
   private void buildHouse() {
      turnLeft();
      putThreeBeepers();
      move();
      turnRight();
      move();
      turnRight();
      putThreeBeepers();
      turnAround();
      move();
      turnRight();
      move();
      turnRight();
      putThreeBeepers();
      turnLeft();
   }
```

```
/*
 * Creates a line of three beepers.
 * Precondition: Karel is in the first square in the line
 * Postcondition: Karel is in the last square in the line
 */
   private void putThreeBeepers() {
      for (int i = 0; i < 2; i++) {
         putBeeper();
         move();
      }
      putBeeper();
   }

/**
 * Backs up one corner, leaving Karel facing in the same direction.
 * If there is no space behind Karel, it will run into a wall.
 */
   private void backup() {
      turnAround();
      move();
      turnAround();
   }
```

Try running this program on your own computer by downloading the java files from the website. What would you do if you wanted the houses to be taller? How would you make them 5 beepers high? Is there any way you could improve the style of the solutions?

Style Focus for Section 1:

Comments: Make sure to comment every method you write, and describe what the method does, and what the assumptions are before and after it is called. Write your comments so that your program could easily be understood by another person.

Good Method Names: Part of good style is good naming. You want your method name to succinctly describe what it does. Never call a method `doStuff`, give it a good specific name like `backup`. Be consistent in how you name your methods. In our solutions, we will use lower camel case naming conventions.

Short Methods: We could have written our whole program in the `run` method, but it is not good style and is difficult to follow. Try and break it down into methods that are small, understandable pieces of code, and accomplish one main task.