Colin Kincaid
CS 106A

Section #1
June 29, 2018

Solutions for Section #1

Based on handouts by Eric Roberts and Marty Stepp

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

/**
 * This method 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();
   }

// CONTINUED...
```

```
 /**
  * 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 downloading the Eclipse project for this section from the course website and running this program on your own computer. 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?

```
/*
 * File: HangingChadKarel.java
 * ---------------------
 * A program in which Karel cleans up hanging chads from a ballot.
 * Precondition:  Karel stands at the start of the ballot.
 * Postcondition: Karel is at the end of the ballot and all chad has
 * been cleared.
 */

import stanford.karel.*;

public class HangingChadKarel extends SuperKarel {

   public void run() {
       // To avoid the fencepost problem, we split the logic into a
       // loop to process columns, plus one final call to check the
       // last column.
       while (frontIsClear()) {
           processColumn();
           move();
       }
       processColumn();
   }

   /**
    * This method clears chad from the current column, if any.
    * Precondition: Karel is standing in the center of a column,
    * facing East.
    * Postcondition: Karel is back in same place/orientation and chad
    * has been cleared.
    */
   private void processColumn() {
       // If there is chad to clear, clear that chad from the ballot.
       if (noBeepersPresent()) {
           removeAllChad();
       }
   }

   /**
    * This method clears chad from the current column.
    * Precondition: Karel is standing in the center of a column to be
    * cleared.
    * Postcondition: Karel is standing in same place and the column
    * has been emptied.
    */
   private void removeAllChad() {
       turnLeft();     // clean the upper corner
       cleanChad();
       turnAround();   // Clean the lower corner.
       cleanChad();
       turnLeft();     // Face East
   }



// CONTINUED...
```

```
    /**
     * This method clears chad from the current corner.
     * Precondition: Karel is facing a corner to be cleared of chad.
     * Postcondition: Karel is in the same location/orientation, but
     * all chad has been cleared from the corner Karel is facing.
     */
   private void cleanChad() {
       move();
       while (beepersPresent()) {
           pickBeeper();
       }
       moveBackward();
   }

   /**
    * Karel moves one step backwards.
    * Postcondition: Karel is facing in its original direction, but
    * stepped backwards.
    */
   private void moveBackward() {
       turnAround();
       move();
       turnAround();
   }
}
```

Try downloading the Eclipse project for this section from the course website and running this program on your own computer. What would you do if you wanted Karel to repair ballots that were broken (that is, the center of the rectangle is not punched out, but some other square in the rectangle is)?

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 that accomplish one main task.

See the CS 106A Style Guide on the course website (linked to from the "Assignments" dropdown) for more style tips!