

Solutions to the Lecture Problems

1. PutBeeperLine

```
/*
 * File: PutBeeperLine.java
 * -----
 * This program tests the putBeeperLine method, which creates a
 * line of beepers moving forward from Karels position.
 */

import stanford.karel.*;

public class PutBeeperLine extends SuperKarel {

    public void run() {
        putBeeperLine();
        turnLeft();
        putBeeperLine();
    }

    /*
     * Creates a line of beepers by putting a beeper down on every
     * intersection between Karel's current position and the next wall.
     */
    private void putBeeperLine() {
        while (frontIsClear()) {
            putBeeper();
            move();
        }
        putBeeper();
    }
}
```

2. BanishWinter

```
/*
 * File: BanishWinter.java
 * -----
 * The BanishWinter subclass gets Karel adorn a series
 * of trees with a cluster of beeper leaves.
 */

import stanford.karel.*;

public class BanishWinter extends SuperKarel {

    /* Main program */

    public void run() {
        while (beepersInBag()) {
            findTree();
            addLeavesToTree();
        }
    }

    /*
     * Moves Karel up to the next tree.
     *
     * Programming style note: Since a tree is simply a wall,
     * this method can simply call moveToWall. You could
     * therefore replace the findTree call in the main program
     * with moveToWall, but the program might then be harder to
     * read because it violates the "tree" metaphor used at the
     * level of the main program.
     */
    private void findTree() {
        moveToWall();
    }

    /*
     * Adorns a single tree with a cluster of leaves. The
     * precondition is that Karel must be immediately
     * west of the tree, facing east; the postcondition is
     * that Karel is at the bottom of the other side of the
     * tree, facing east.
     */
    private void addLeavesToTree() {
        turnLeft();
        climbTree();
        addLeaves();
        descendToGround();
        turnLeft();
    }
}
```

```
/*
 * Climbs to the top of the tree.
 */
private void climbTree() {
    while (rightIsBlocked()) {
        move();
    }
}

/*
 * Moves Karel back to the ground. Both the pre- and
 * postconditions have Karel facing south.
 */
private void descendToGround() {
    moveToWall();
}

/*
 * Creates the cluster of leaves at the top of a tree.
 * The precondition is that Karel must be facing north at
 * the top of the tree; the postcondition is that Karel
 * is still at the top, but on the other side of the
 * trunk, facing south.
 */
private void addLeaves() {
    turnRight();
    makeBeeperSquare();
    move();
    turnRight();
}

/*
 * Moves Karel forward until it is blocked by a wall.
 */
private void moveToWall() {
    while (frontIsClear()) {
        move();
    }
}

/*
 * Creates a square of four beepers, leaving Karel in its
 * original orientation. The resulting square is positioned
 * ahead and to the left of Karel's starting position.
 */
private void makeBeeperSquare() {
    for (int i = 0; i < 4; i++) {
        putBeeper();
        move();
        turnLeft();
    }
}
}
```