CS 157 Lab 17 December 8, 2009
Building Objects

You will be building a simple game that involves a player on a square board. The player will always start in the lower left corner of the board. Somewhere on the board will be a hidden "beeper". The player's job is to find it.

1. Create a new Java project called Find the Beeper. Add classes called Player, Beeper, and Game, and in Game add a main method. In the Game class, create a public constant (using the modifiers **static** and **final)** for the size of the board, call it SIZE, and set it to 5. Do this in the same place where you would create fields. Only in this case, since SIZE will be a constant, you will make it public.
2. In the Player class, add integer instance variables for the Player's x and y position. Remember to make them private.
3. A "constructor" is like a method, in that it has parentheses in its declaration, and, possibly, uses arguments. But it does not have a return type, not even void. Its name is the same as the name of the class. Its purpose is to initialize instance variables. Constructors are declared public. You will not see an exception to this rule for at least a year. Create a constructor for the Player that has no arguments, and in the body of this constructor, set both instance variables to 1.
4. Add methods to the Player called getX and getY that return the location of the Player object. Add a move method that returns a boolean and takes two integer arguments that represent the amount that the Player is supposed to move in both the horizontal and the vertical direction. For example, if **bob** is a Player object located at (1,1), doing **bob.move(2,3)** would result in bob being located at (3, 4). You could then do **bob.move(1, -2),** and then bob would be located at (4, 2). The catch is, however, that the move method must not allow an illegal move. So if either the new x value or the new y value is less than 1 or greater than the size of the board, the position of the Player must not change, and the move method returns **false**, otherwise the move is allowed, the Player changes position, and **true** is returned.
5. Create a JUnit test class for Player, and include a test for the move method. Inside this test, create a Player object, assert that it is located at (1,1), then make a legal move and assert that the Player object is located where you expect it to be. Then make an illegal move and use an assertFalse on the move being successful, and follow it with another assertion that makes sure the Player is still located at the position before the illegal move was made. Make sure that your tests will still work if the size of the board changes.
6. The Game class will have no objects created from it. So all of its methods will be static. Add a public method to the Game class called printBoard. This method is responsible for printing "dots" where an empty spot is located, and the letter "P" where the player is located. You will need two nested for loops. The outer loop will control the y-value. In the inner loop, the x-value should change from 1 to the board size, and inside that loop, if the position of the Player is the same as the x and y values that the loop is working with, then a "P" should be printed, otherwise a "." should be printed. Also, the outer loop's control variable (the y-value) should count backwards, since the top row of the board gets printed first. When you think you have printBoard right, in the main method of the Game class create a Player object, print the board, send the Player object a legal move, and print the board again. You should see the Player object located at the right place.
7. In the Beeper class, add integer instance variables for the location as you did for the Player. But in this class you will NOT have get methods for the location. The location will be unknown, and the Beeper will not print on the board. It will be placed on a random location. To do this, you will need

to make Java choose random integer numbers from 1 to the size of the board. Create a constructor for the Beeper class that sets the x and y location of the Beeper object to random numbers between 1 and the size of the board. Also, add an **if** to the end of the constructor that resets the location of the Beeper to the opposite corner if the random location that was generated turned out to be (1, 1) (where the Player starts). To make sure this works, create a JUnit test class for the Beeper class. In it you have to make sure that the Beeper is always located within the board. Since you don't have any way to find out where the Beeper is located outside the Beeper class, you may ***temporarily*** make the location public, test it a lot of times to make sure the Beeper is on the board and not at (1, 1), and then make the location private again and comment out the tests.

8. In the Game class, we want to allow the person who is playing the game to move the Player object. So add static methods called getNewX and getNewY that use a Scanner object to ask the user for the values, and then return those integers. Test them by having the main method ask for those values, then make the Player object move by those amounts, and print out the board before and after. Try making illegal moves from time to time to see what happens.

9. Refactor the printBoard method so that it uses a String parameter that stands for the character that is to be printed (other than a "."). The effect of this is that when someone calls printBoard and passes a character in quotes, that character is printed at the Player's location. So if a "P" is to be printed, the client says `printBoard("P")`, but if the client wants an "*" to be printed there, we use `printBoard("*")`. When you have changed printBoard, test it completely in the main method of the Game class.

10. Create a method of the Beeper class called **beep** that returns a boolean and takes a Player object. It should return true if the Player is located where the Beeper is. In that case it should also print a message saying the Beeper has been found, and print the board with the "*" at the location of the Beeper and Player. If the Player is not located where the Beeper is, just return false. To test this method, we could temporarily make the Beeper's location public again. But another way is to create "cheat" methods that are not publicized. Add public methods called cheatX and cheatY that return the location of the Beeper, and use those methods ONLY in the JUnit test for **beep.** In the test method, create a Player object, call **beep** and make sure it returns **false** (the Player initially is not located where the Beeper is), then move the Player to where the Beeper is (using the cheat methods), then check to see that **beep** returns true.

11. In the Game class, in the main method, add a loop that continues as long as the Beeper has not been found. The loop should repeatedly ask the user for a new move, make the move, and print the board after each legal move. At this point you should be able to run the Game class as a Java application and play the game to find the Beeper. If you move the Player systematically around the board, you will eventually find the Beeper (make sure your board is very small; 5 is good).

12. We would like to be able to find the Beeper without having to try so many locations (imagine how hard it is when the size of the board is 80). We can make this interesting in a number of ways, all of which involve the Beeper giving out additional information. One way is for the Beeper to announce, whenever **beep** is called, whether it is located north, south, east, or west of the Player. Here you may find it helpful to add a method to the Beeper class called announceDirection that takes a Player object as a parameter and prints the basic direction from the Player that the Beeper is located. This is actually very simple to do. If the x coordinate of the Beeper is greater than the x coordinate of the Player, you should print an "E", because the Beeper is East of the Player. Similarly for west, north and south. So there should only be 8 possible things printed: N, S, E, W, NE, NW, SE, SW. Write the announceDirection method, call it from the beep method when the Player is not located where the Beeper is, and then play the game. If you make your board size 80, you should be able to find the Beeper after about 6 to 10 moves.