Classes, Random Numbers, and Lists

Pollinating Bees Lab

# Main Objectives

* Work with Lists of Objects (for each loops, method calls on the objects inside, etcetera).
* Use several static methods from the Math class with several uses of the random method.
* Reinforce knowledge of classes with fields, constructors, and methods.
* See the basic concept of a finite state machine.
* Minor goal – practice using static final variables to reduce ‘magic numbers’.

# Prior Knowledge

* Familiarity with Math.random and generating integers.
* Familiarity with looping structures including for-each loops.
* Familiarity with class constructs of fields, constructors, and methods.
* Familiarity with Lists.

# Greenfoot Background

This lab does use the Greenfoot API relatively extensively. Students will need to know how to move, turn, setRotation, and setImages of Actors, and they will learn how to use Actor methods which return Lists of Actor types. Students will need to know how to ask the World to place new Actors on the screen. For the more detailed touches, students will need to know or learn how transparency values work on GreenfootImages.

# Getting Started

Install the Greenfoot IDE: <https://www.greenfoot.org/download>

* Greenfoot went through many changes beginning with the 3.0.0 release. Version 2.4.2, although older, tends to be one of the most stable versions but is missing a few of the newer additions.

In your Pollinating Bees folder run the project file with the Greenfoot icon to begin.

# Mini-Goal for Students

Throughout this lab there will be many numbers used to keep track of how far away Bees should be from Flowers, or Bees from other Bees, or the number of Bees given to each hive, or random locations that certain actors may be built, etcetera. Each of these numbers could be written directly into the code, but when you realize that the numbers should be modified a bit to try different values you will need to go hunt those numbers down and adjust them. Worse, if someone else were to try modifying your code in the future they would have no idea what any of the numbers represent and would be frustrated trying to update your code.

These types of numbers are often referred to as ‘magic numbers’ and should be avoided in actual code. Instead of writing the numbers directly into your code it is better practice to make static final fields to represent the number by giving the number a name. The word ‘final’ prevents the number from changing after compilation. The word ‘static’ means only one copy of the number will be stored in memory, instead of each individual object storing a separate copy of the number.

Example: private static final int BEES\_PER\_HIVE = 10;

As a mini-goal, try to avoid using as many numbers as possible. Keeping numbers like 0 or 1 for counting purposes is usually okay, or a / 2 for cutting a value in half is also okay, but having the number 450 in your code instead of using a static final variable should be avoided.

# Demonstration

View a sample demonstration of what the Pollinating Bees lab could look like here:

<https://www.greenfoot.org/scenarios/23656>

# Instructions

The instructions this time will be somewhat vague to allow the student to figure out how to approach the situations themselves. Each class will be identified with its main goals and students are asked to figure out how to make those goals come to fruition.

### The World:

The World is responsible for generating the initial Bee Hives and the initial Flowers.

Randomly generate locations for the Bee Hives to stay closer to the top of the screen but still appear like they are in the grassy areas.

Randomly generate locations for several Flowers to appear almost anywhere in the grassy areas of the screen.

The World should also specify the order of painting objects: The Hives should be on top, then Bees, then Flowers, then Seeds.

### The Hive:

The main responsibility of the Hive is to act as a home for the Bees that live in it. The Hive should keep track of its own group of Bees. The Hive is able to summon the Bees back to the Hive at any point with a particular keyboard key. Imagine the Queen Bee is calling all Bees home.

Each Hive, when constructed, should be told the number of Bees the Hive will have as well as which keyboard key it will use to summon the Bees back to the Hive. The Bees should be built and placed on the World. When the correct keyboard key is pressed for a Hive, it should tell all Bees to turn towards the Hive.

Use for-each loops whenever possible while writing the Hive class.

### The Flower:

The main responsibility of the Flower is to keep track of how long it has been since the Flower was pollinated. If enough time has elapsed since the last time a Bee pollinated the Flower, then the Flower should begin wilting (becoming transparent is an okay visual for this) until the Flower finally is removed from the World.

When constructed, the flower should choose one of the 12 different Flower images at random to create a variety of Flowers.

### The Bee:

The main responsibility of the Bee is to travel around looking for nearby Flowers. Bees get attracted to the closest Flower and once it lands on a Flower it should begin the pollination process. Bees should keep track of how much pollen (a numeric value) they have collected and eventually, once a certain level of pollen has been collected, the Bees should build and drop a new Seed at their location.

Bees should walk around in a somewhat random pattern, constantly readjusting their course to mimic frantic movements. Once a Bee is within range of some Flowers, the Bee should determine which Flower is the absolute closest then begin angling towards that Flower. Upon arrival at the Flower, the Bee should determine if there are too many Bees already on the Flower. If so, the Bee should immediately go back to roaming around somewhat randomly. However, if there are not many Bees on the Flower, the Bee should stop and ‘collect’ pollen for a short time before going back to roaming.

Consider four different ‘states’ for the Bee to be in:

* + Roaming: The Bee is randomly moving around with no care of other objects.
  + Searching: The Bee is randomly moving around looking to find Flowers within a range.
  + Targeting: The Bee has found a Flower within range and moves towards that flower.
  + Pollinating: The Bee is on a flower and is collecting pollen.

Images exist to help differentiate the different states of the Bee. Different states of an object which change from one state to another based on some conditions is the basis for a computational model called a ‘finite state machine’.

After a Bee has pollinated for a short time, it should switch back to a roaming phase where it must stay for another short time before switching to the searching phase again. If the searching phase is too fast, the Bee will just keep returning to the same Flower repeatedly.

Due to the random roaming state, the Bee may end up in a situation where it has multiple Flowers within its targeting range. If so, the Bee should determine which of those Flowers is the closest Flower to the Bee. This involves a variation of a standard APCSA algorithm of finding the smallest item in a List. You will need to use the distance formula to determine how far away each Flower is from the Bee.

### The Seed:

The main responsibility of the Seed is to be a temporary visualization of a Flower before it is fully grown. A seed should keep track of the number of acts it has been on the World. Once the number of Acts hits a certain value, the seed should switch images to the Flower it will eventually become, then it should slowly transition from being almost entirely transparent to fully visible. Once the seed is fully visible, it should add a real flower to the screen and remove itself from the screen.

A hint: the Seed can build and store the Flower it will become in its constructor and a field. This way the Seed can ask the Flower what image it will later use before eventually adding the Flower to the world.

Secondary: It is possible for the Flower and the Seed to be merged into one class.

