# CSCI-140 Computer Science AP
# CSCI-242 Computer Science Transfers
# Lab 7 - Secret Lives of Bees



## Introduction

### Goals

The goal of this lab is for students to gain experience with fundamental concurrency topics such as:

- Threads
- The Monitor (synchronized regions)
- Thread Control
  - `start` VS `run`
  - `join`
  - `sleep`
- Thread Synchronization
  - `wait`
  - `notify`
  - `notifyAll`

### Overview

The bees in the beehive are buzzing! If you recall from science class, a *bee* in a typical colony has one of three distinct roles:

- **Queen**: Her royal highness of the beehive. The sole queen is responsible for mating with the drones in her chamber and then depositing her eggs into sacs that are populated with nectar and pollen. Over time, each egg will develop into a fully developed adult that joins the colony.
- **Worker**: A female bee who has the specific job of gathering resources (nectar or pollen) from a field of flowers. These resources will play a critical role in the reproduction of new bees.
- **Drone**: A male bee whose only role is to mate with the queen in her chamber. The process of mating with the queen is life ending for the drone as they suffer irreversible anatomical damage.

### Simulation

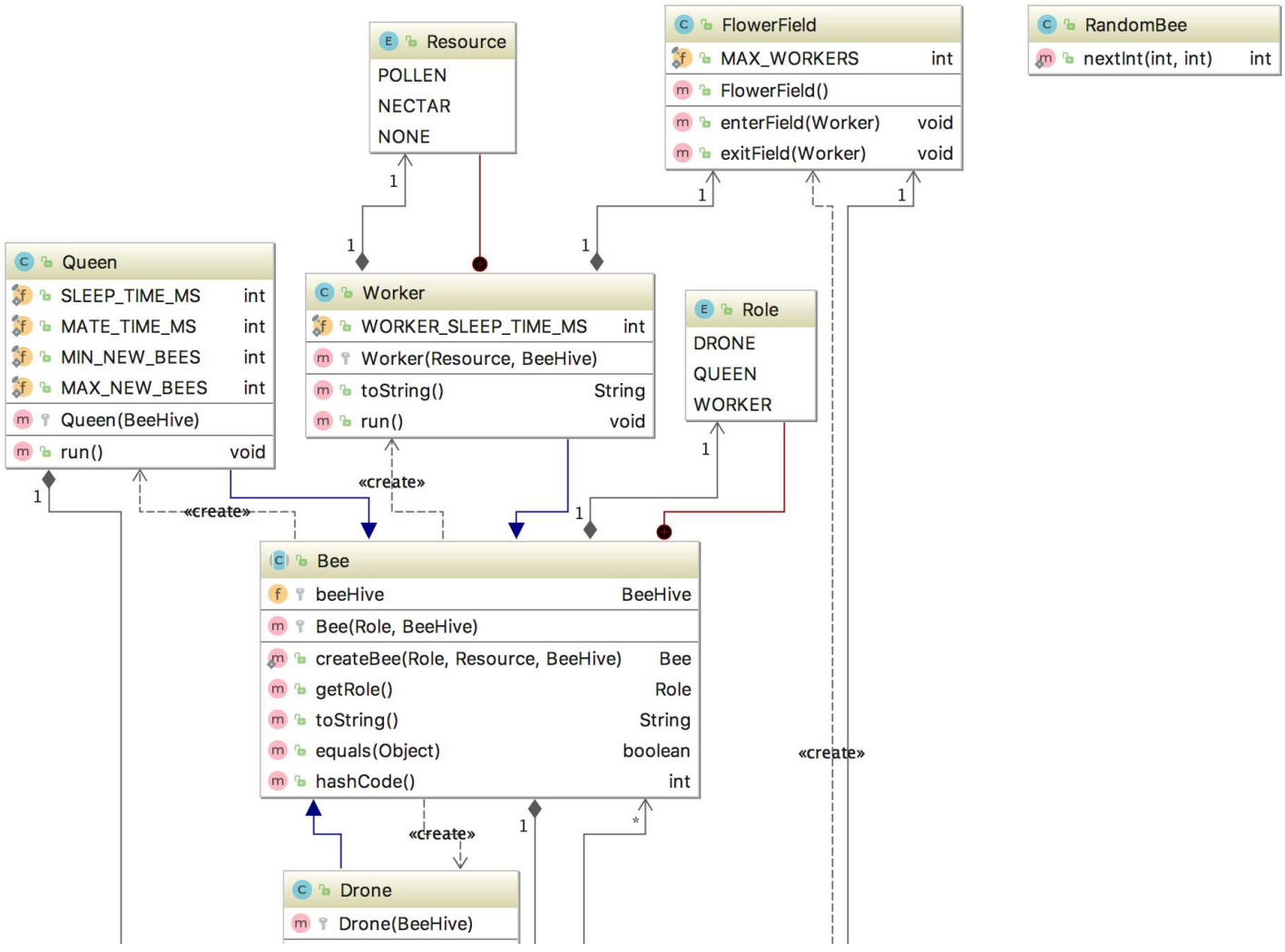
At the start of the simulation there will be one queen, any number of drones and workers, and no resources in the beehive. These are the details of the time it takes for each type of bee to perform a single task.

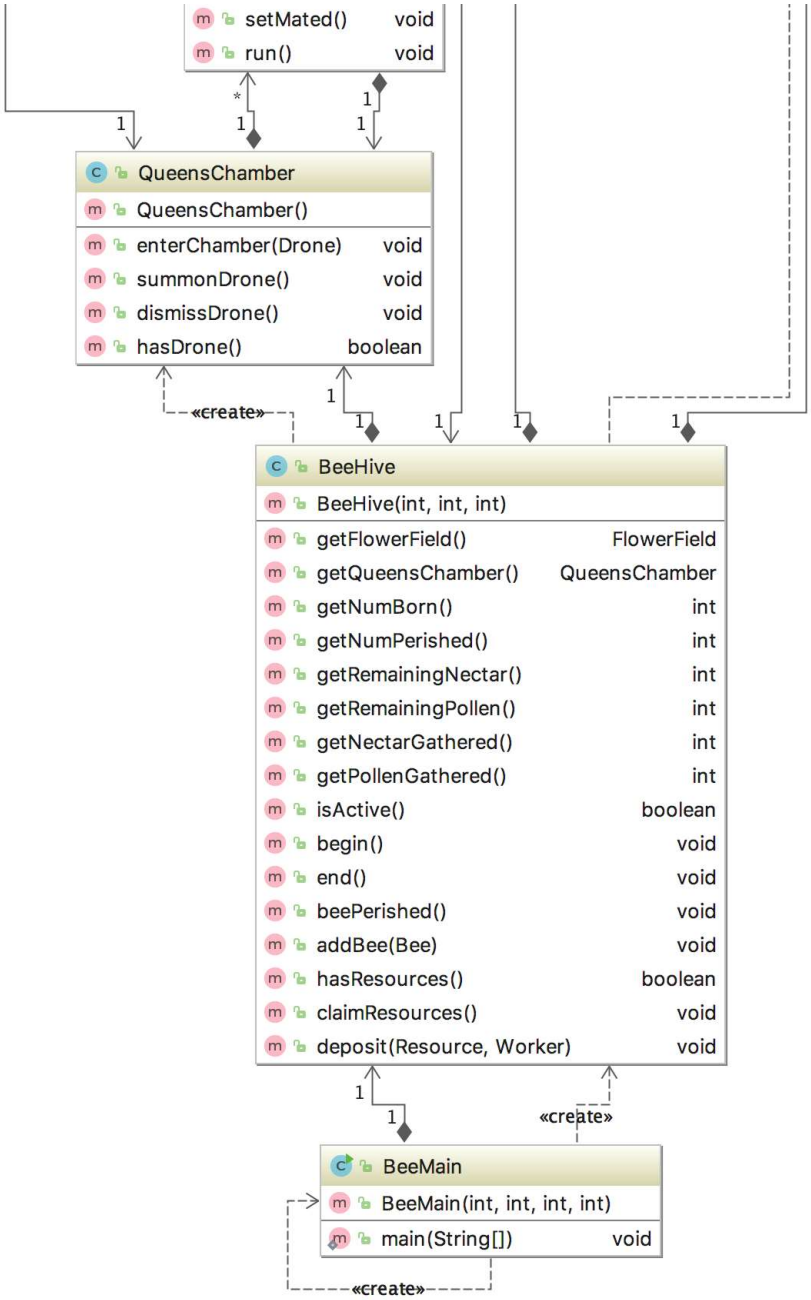
- **Worker**: A worker's task is to go to the field of flowers and gather one unit of their assigned resource. The field has a limited number of flowers. If there isn't a free flower, the worker must wait. Once a flower is free, exactly one of the waiting workers will randomly be selected to gather from the free flower. If a flower is free it takes a bee 2 units of time to gather their resource and deposit it into the beehive.
- **Drone**: The fated task of the drone is to reproduce with the queen by entering the queen's chamber and lining up first come first served. The queen will mate with the first drone when the conditions are ready (see the Queen's task). It takes one unit of time for the drone to mate with the queen. If the drone mates with the queen, they perish immediately after.
- **Queen**: The task of the queen is to reproduce with the next waiting drone. It requires the beehive have at least 1 unit of both nectar and pollen. If the conditions are met, the queen takes the required resources from the beehive and places them into an empty sac. She then mates with the next drone for 1 unit of time. This instantly creates randomly between 1 and 4 new bees (depending on enough resources) that will be added to the colony. After each attempt at mating, the queen will sleep for 1 unit of time.

## Design

We strongly suggest that you follow the supplied design for this lab. This class diagram details all the classes and their relationships.

Please note that this diagram, and the documentation below, both exclude the private state. It is expected you will define your own private state using good design principles, e.g. a data member that is shared by all subclasses should be defined in the shared superclass.






## Documentation

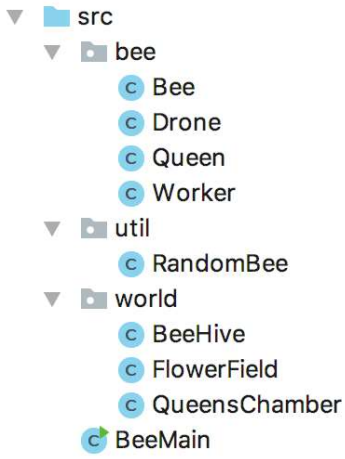
Here is the documentation for all the classes in the classes in the simulation.

## Starter Code

In the event you are having problems with GitHub and cannot get the starter code, you can access it here.

### Project Structure

Your project structure should look like the following once you have the starter code. Note that there are three packages, `bee`, `util` and `world`. Make sure you put the Java source files in the right location.




# Implementation

## Command Line

The program can be run on the command line as:

```
$ java BeeMain seconds drones nectar_workers pollen_workers
```

If the number of arguments are correct, they are guaranteed to be valid integers that will be used as separate seeds to the random number generated (used to initially shuffle the order of the heroes).

If the number of arguments is incorrect, display a usage message and exit the program, i.e.:

```
Usage: $ java BeeMain seconds drones nectar_workers pollen_workers
```

## Input

This program has no user input outside of the simulation numbers provided on the command line. Your program should not prompt for any input while running and should run the simulation to completion.

## Output

Look here for a bunch of sample runs. These are all good things to test your code with to make sure you are running correctly. Of course there is no guarantee that the output for any of these will be the same as yours, based on the random number generator and the very nature of threads in general.

## Implementation Details

### Custom Random Number Generator

You are being provided with a class `util.RandomBee` that has a method nextInt that the queen can use for randomly generating the number of new offspring after successfully mating with a drone.

### Factory Method Pattern

From your main and world package perspectives, the Bee constructor is not accessible. You again will use the factory method that you saw from lab 4 to create the bees using Bee::createBee.

Whenever a client wants to make a new bee, for example a pollen worker, assuming that `beeHive` points to the BeeHive instance, they will do:

```
Bee.createBee(Bee.Role.WORKER, Resource.POLLEN, beeHive)
```

### Constant Field Values

This page lists all of the constant field values that you should define and use.

### Enums

When implementing the Java Enum, Worker.Resource you do not need to implement the `valueOf` and `values` methods. Those are provided automatically when you implement an enum. If you are confused, take a look the source code for the provided enum Bee.Role, in `src/bee/Bee.java`. You only need to define the enum and the two types.

### Bee Queue

The bees are stored in the beehive as a ConcurrentLinkedQueue. This is a thread safe collection from the `java.util.concurrent` package. The reason for this is because when shutting down the simulation in the main thread by looping over the collection and joining on all the bee threads, the Queen could be creating a new bee to add to that same collection (she hasn't gotten back to checking that the beehive is now inactive). Using this collection prevents a ConcurrentModificationException.

### Flower Field

As you saw in the in-lab activity, the flower field is a monitor that the worker bees use in order to keep track of how many workers are currently "in the field". There is a limit of 10 workers at once in the field, and they must obey this. If the limit is not reached, they can gather, otherwise they must wait and get notified by another worker bee who has finished gathering. The order the bees gather from the field is random and unpredictable.

### Queen's Chamber

The Queen's chamber is a more complicated monitor than the flower field. It is used by both the Queen and the drone bees in order to facilitate reproduction of new bees. Unlike the flower field, the drones must line up in order when they enter the queen's chamber. If the queen is not ready to mate, the drones must wait. When the queen is ready to mate, she will notify all of the drones that she is ready, and the drone that is at the front of the line will unblock and come out of the line to mate with the queen. After mating the drone will perish. The queen will create between 1 and 4 new bees randomly, as long as there are enough resources.

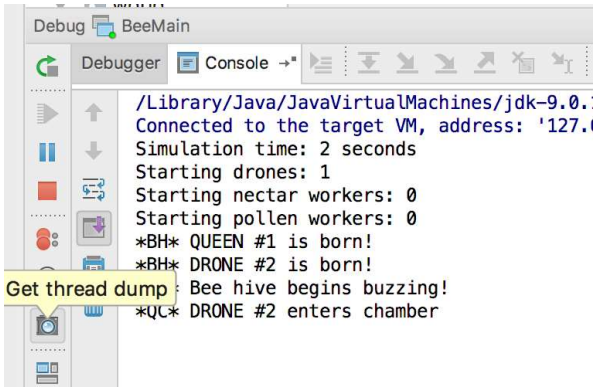
At the end of the simulation, there may be drones still waiting in the Queen's chamber. It is the queen's job to dismiss each drone individually, until there are no more drones waiting in line inside her chamber.

**Please note!** The javadocs will not generate the `synchronize` keyword for the methods that you have to write in QueensChamber. It is absolutely required that all of these methods are declared to be synchronized and do not allow multiple threads to run in them at the same time. You will get an IllegalMonitorStateException if you try to `wait()` in an unsynchronized method.

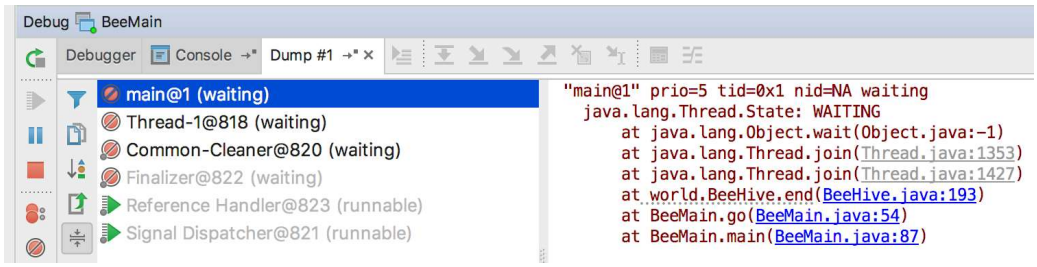
### Thread Debugging

Let's say you have a scenario where your program appears to deadlock and not finish. If you run the program in debug mode (green bug icon next to the normal run one), you can select the camera icon to get a *thread dump*.

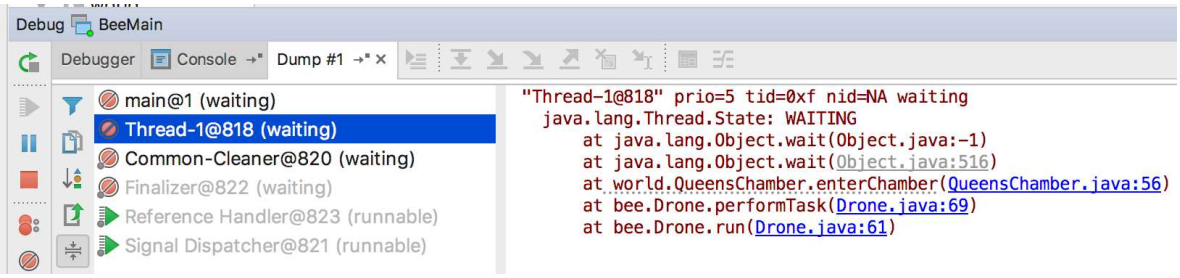
In this example with just one queen and one drone (no workers), there appears to be a hang. In debug mode we see:




By clicking the camera icon we can see the threads that are running (in the left hand window). Let's look at the main thread first.




It looks like the main thread is in `world.BeeHive.end` (look at the stack trace in red to the right) and is trying to join on a separate thread that has not completed its run cycle. Let's look at the separate thread directly below the main thread.




The problem is there is a drone thread that is "stuck" in the queen's chamber waiting to be notified (take a look at the right stack trace and you will see the drone is waiting inside `world.QueensChamber.enterChamber`). To fix this deadlock problem you need to make sure the queen dismisses every drone she doesn't mate with before the queen thread finishes running.

## Submission

Try is setup to handle code that you submit with packages. However, all files need to be sent "flat" without sub-directories. The try command to run from your src directory (and assuming your log.txt file is also in the src/ directory):

```
try csapx-grd lab7-1 *.java */*.java log.txt
```

Please note that you should not modify these files and they will be replaced with the ones provided to you from the starter code when try compiles your submission:

- `src/BeeMain.java`
- `src/bee/Bee.java`
- `src/util/RandomBee.java`

## Grading

The grade breakdown for this lab is as follows:

- Problem Solving: 15%
- In-Lab Activity: 10%
- Design: 10%
- Functionality: 55%
- Code Style: 10%