**Assignment 2**

[lgodfrey](https://paradigms.lukesterwebdesign.com/author/lgodfrey/)[June 13, 2017](https://paradigms.lukesterwebdesign.com/assignment/assignment-2/)

Top of Form

**I. Problem Statement**

For this assignment, you will implement and test a parallel program that simulates the producer/consumer problem in a themed environment.

**The Theme**

There is an epic battle between two opposing castles. Each castle builds weapons in order to conquer the other. Although the armories in each castle are empty, the residents of the castles are skilled at making weapon parts and putting them together.

Each castle has two sets of workers: producers and consumers. Producers create weapon parts and add them to queues, and consumers use the weapon parts to build weapons. Each time a weapon is built by one castle, the opposing castle loses HP at a faster rate depending on the attack power of the new weapon.

Naturally, the producers and consumers must run in parallel and not be susceptible to deadlocks.

**The Project**

You must implement the following classes.

* **WeaponPart**. A class representing a single weapon part that a producer may make. You must code up a way to calculate a random amount of time to produce each type of WeaponPart (see WeaponPartType below for a hint about how you could do that). You may use a subclass for each kind of WeaponPart, or just use a String (i.e. name) to identify the part type.

Bottom of Form

* **Weapon**. A class for representing a single weapon that a consumer may build out of weapon parts. You must code up a way to indicate which WeaponParts are required to build each type of Weapon (see WeaponType below for a hint about how you could do that). Different types of weapons should have different attack powers. You may use a subclass for each kind of Weapon, or just use a String (i.e. name) to identify each type.
* **Producer**. A class that extends thread that produces a specific type of WeaponPart and adds them to one of the Queues in Castle (see below). In the run method, continuously produce new WeaponParts and add them to the queue, sleeping for the appropriate time based on the type of WeaponPart. The run method should end if an InterruptedException is thrown.
* **Consumer**. A class that extends thread that builds a specific type of Weapon and adds them to the armory in Castle (see below). In the run method, continually build Weapons and add them to the armory, consuming the appropriate WeaponParts from Castles Queues (or waiting until they are available) based on the type of Weapon. The run method should end if an InterruptedException is thrown.
* **Castle**. A class containing a List of Producers, a List of Consumers, a List of Weapons (the armory), Queues for each weapon type, and HP. Castle should have a fixed amount of workers (i.e. 32), but how each castle uses the workers can be different (i.e. half-and-half producers and consumers, more producers, more consumers, more producers for weapon parts that take longer to build, etc).

Your main method should create two Castles using different strategies for assigning workers to producing and consuming tasks. The Castles should immediately start running the worker threads. The main thread should use a loop (possibly including Thread.sleep) that updates each Castle's HP based on the armory of the enemy Castle. The loop should end when one Castle's HP has been drained to zero. At this point, the Castle should interrupt its workers to end the program.

You may choose to implement the following classes; this is a hint about how you might make the required classes work.

* **WeaponPartType**. A helper class storing information for constructing weapon parts. This class could contain a weapon part's name, mean construction time, and standard deviation of construction time.
* **WeaponType**. A helper class storing information for constructing weapons. This class could contain a weapon's name and a List of WeaponPartTypes needed to build this type of weapon.

Your code *must* account for race conditions and *must* avoid deadlocks. That is the focus of this assignment. Hint: Check out the Java Lock.lockInterruptibly function.

**10 bonus points are available for creative extensions to the program.** One idea is to have a separate mean and standard deviation for each type of weapon for randomizing attack power, or a weapon durability system. Only substantial extensions done well will be considered for partial or full bonus.

**II. Design**

The high level design has been done for you, but you will have to determine...

* The specific methods and properties for each class.
* Appropriate values for variables like build time and HP.
* How to avoid race conditions, like two producers making the same part at the same time, or two consumers using the same part at the same time.
* How to avoid deadlocks that may have been introduced by your solution to the race condition problem.

**III. Implementation**

You may choose to implement this assignment using a top-down or a bottom-up approach. In either case, make sure you code incrementally and test as you go. You should probably come up with some small "mini-simulations" that you can use to verify pieces of your code as you write the classes.

**IV. Project Report**

Test your program to check that it operates correctly for all of the requirements listed above. Also check for the error handling capabilities of the code, where applicable. You are required to include your testing results (output of your program running) in your project report to demonstrate that your program works correctly. You should include regular cases and edge cases in your testing, where applicable. If you can’t get the whole program to run correctly, create a main function that demonstrates what your code does perform successfully.

Your report should also list some statistics about the results of running your simulation multiple times. For example, in this project you should try assigning different numbers of workers to different tasks (i.e. producer or consumer) and use different strategies for assigning workers to specific weapons or weapon parts. Which strategy tends to win?

In addition to the test results, this report should include a couple of sentences about each of the following topics:

* *Design decisions.* Why did you design it that way?
* *Implementation process.* How did you implement the design?
* *Problems.* What changes did you have to make to your original design? What were you not able to implement completely?
* *Results.* What did you learn from this project?

0.5 – 1.5 pages is an acceptable length for your reports, *not* including the test results.

**V. Rubric**

I will follow this breakdown for assigning grades:

* Design and abstraction: 30%
* Implementation of classes: 30%
* Implementation of simulation: 20%
* Report and testing (including stats from your simulation): 20%

Here is some sample code from today’s lecture about how you might set up WeaponParts for assignment 2.Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | public class Gunpowder extends WeaponPart  {  public double mean()  {  return 500;  }    public double stddev()  {  return 100;  }  }    abstract class WeaponPart  {  public WeaponPart()  {  random = new Random();  }    public int timeToCreate()  {  int time = (int) (random.nextGaussian() \* stddev() + mean());  if(time < 0)  time = 0;  return time;  }    public abstract double mean();  public abstract double stddev();    private Random random;  } |