Assignment #2

Submit Assignment

Due Sunday by 11:59pm **Points** 15 **Submitting** a file upload
Available Dec 28, 2022 at 8am - Jan 29 at 11:59pm about 1 month

PigMent.java

output_example.txt

General Information

Files

skeleton + example output placeholder

Tasks and assessment

The assignment consists of two interrelated tasks. The second task is an augmentation of the first one, so you have to solve Task1 before advancing to Task2.

Each task has smaller tasks, so by skipping certain parts (or creating a simplified version of them) your program might still work. You can get points for a partially solved task. The prerequisite for a successful submission is that your program works, in other words it can be compiled and it runs without error or crashing.

By solving another task incorrectly, you can mess up your existing, partially finished, but functional program. If you already have a working program, you are advised to create the second task as a new file.

Tools

You do not need to start from scratch. Use the `PigMent.java` file as a starting point.

Task1 (9p)

Executor, Runnable, synchronized, AtomicInteger, wait-notify

Difficulty: I can do this!

Story Time

On a planet called *PigMent*, the living world hid underground from the blazing sun. An exception to this is a strange specie of pigs that has adapted to the cruel environment. As a result of a strange mutation, a purple pigment appeared in their skin, with the help of which they reflect most of the harmful ultraviolet radiation. From the absorbed light, they develop energy through photosynthesis. Pigs are gender-neutral, so once they reach a certain size, they reproduce by dividing (similarly to bacteria). Each pig has an independent will and decision-making ability, so you have to treat each pig separately and in parallel with the other pigs during the program.

---

pigSleep - 1 pont

Not everything happens immediately. Big things have to wait.. or sleep..

It is no different here, even in the society of pigs, sometimes there's not much to do. Write a `pigSleep()` method that blocks the calling thread for a certain time between `TICK_MIN` and `TICK_MAX` milliseconds.

PhotoPig - 1 pont

The pig breed is described by a class named `PhotoPig`. In order to be able to track individual pigs, each copy has a unique identifier (`id`). Furthermore, their weight can be set parametrically at birth (`mass`), which is also stored by the class.

Make sure that the initial mass can be set through a constructor argument. With the help of a globally accessible `AtomicInteger` make sure each pig receives a unique id. Start the global counter with zero and increment its value by one every time a new pig is born.

---

pigSay - 1 pont

Although the piglets tolerate the weather well, their brains are slightly cooked in the high heat. Because of this, they tend to shout nonsense on a regular basis.

Give a voice to these wonderful creatures of God by implementing the `pigSay(String msg)` method. The method should write to the console, and also display the ID and mass of the speaking pig.

Only one pig can speak at a time! Otherwise it would be just noise.

---

eatLight() és run() - 4 pont

Pigs lose weight continuously during their metabolism. Catabolic processes (losing weight) are compensated by anabolic processes (gaining weight). However, weightgain requires a caloric surplus. This is why people tend to eat. Here, however, we have photosynthesizing pigs, so it happens a little differently with them. They lay down and rest to absorb sunlight instead of eating. When they've sucked themselves up, they head home. On the way home, the anabolic processes take place within their bodies. As a result they get more muscular and more fat, so they will be bigger than before. When they reach a critical mass, they split into two. The original pig remains alive, but will carry half of its original mass. The other half becomes self-aware and separates from the mother's body.

Make the `PhotoPig` class to be a `Runnable` implementation, so individual pigs can be thrown into an `ExecutorService` object (`pigPool`). Make sure that each pig can live in parallel with the other pigs!

A pig's life begins with the `"Beware world, for I'm here now!"` cry and ends with the `"I have endured unspeakable horrors. Farewell, world!"` sigh. Between the two, they lie down to bask in the sun again and again. The sunbathing and its consequence (division) should be implemented in the `eatLight()` method. Its return value indicates whether the pig has met its fate while sunbathing. The method should return `true` if the pig is still alive. The method should be called again and again until a `false` value is returned. In the first task, the animals cannot die yet, so an endless cycle takes place here. For the first task, this is completely normal behaviour!

The `eatLight()` method treats both anabolic (mass-increasing) and catabolic (mass-reducing) processes. After the pig had busked in the sun, we can evaluate the metabolic outcome. The mass obtained from sunbathing is determined by the capacity of the internal organs, so it is independent of the mass of the pig. After photosyntesis `FEED` amount of mass is always added. The weight lost depends on the weight of the pig, make the pig lose `mass / BMR` amount of weight. Note that heavier pigs have a worse ratio of gained and lost weight. If this ratio is less than 50%, then the pig should procriate! You can use the following formula: `mass / BMR > FEED / 2`, if this is evaluated to true, then a new pig is supposed to be born at the end of `eatLight()`. The original pig remains alive at half the mass. With the other half create a new pig and bring it to life by throwing it into the `ExecutorService pigPool` object. If we have reached `MAX_POP` number of births, then do not place the newborn into the executor object!

After eating light, the pigs squeal in joy `"Holy crap, I just ate light!"`, and when splitting, they squeal `"This vessel can no longer hold the both of us!"`.

---

main() - 2 pont

At the beginning of the simulation, create `INIT_POP` number of pigs. Set the initial mass of each pig to `INIT_MASS`.

If the global pig counter has reached the `MAX_POP` value, the newly born pig should send a signal, causing the simulation to end. In theory, the simulation can run indefinitely, but in practice we want to interrupt it after a certain number of pig births. Since we are interrupting the simulation, rather than waiting for it to finish, there is no need to fiddle with a "nice" shutdown. It's perfectly fine if you use `pigPool.shutdownNow()` to immediately terminate all running threads.

Task2 (6p)

BlockingQueue, ReentrantLock, Condition

Difficulty: I mean.. I like a challange, right? Right?!

Story Time

The pigs have become so crazy from the gamma radiation caused by the many solar flares that they no longer despise cannibalism. According to them, this is just a more efficient and faster form of getting their daily nutrition. Pigs now come out of their hiding places not only to sunbathe, but also to hunt.

First, they walk around busy places looking for prey. If they find a pig no bigger than them, they look at it as food. While watching, a hunter may fell prey to another hunter. The hunter hides after findind a suitable target. After hiding the other hungry pigs can no longer see the hunter.

If he did not find prey, or while he was hiding, someone else got to his prey, or the prey got away (e.g. the prey is also a hunter who went hiding or just simply went home), then the hunt is unsuccessful. In other cases, the hunt is successful and the pig will be enriched with half the weight of the prey. The prey is killed in this case.

Regardless of success, the hunt is exhausting, so the pigs lie down to rest for the day before heading home. They will then become visible to other hunters again. In return, they get the extra mass for photosynthesis. If they are still alive by the time they wake up, then they survived the the current trip. On the way home, the weight loss and division resulting from metabolism take place in a similar way as in the first task.

---

openArea és aTerribleThingToDO() - 4 pont

Use a `PriorityBlockingQueue` (`openArea`) to enable the pigs to register when they are visible and when they are not visible to the hunters. Help with instantiation: `new PriorityBlockingQueue<>(MAX_POP, (a,b) -> a.mass - b.mass);`

During hunting the hunter spots a pig with the help of the queue. After the prey is identified, the hunter tries to hide. This takes `pigSleep()` amount of time and the hunter can be seen by other predators during this action. To make the pig visible to others it has to register (insert) itself with the `openArea` object and deregister (remove) itself afterwards. If the `openArea.remove(this)` command failes, it means the hunter was hunted down by another hunter and is dead now. If the hunter is still alive, but the prey could not be removed from the `openArea` object

, then the hunt was a failure. E.g. the hunt failes if the prey hides (the prey removes itself from the queue) or was killed by another hunter in the meantime.

Implement the hunting mechanism inside the `aTerribleThingToDO()` method. The `eatLight()` method should register and deregister similary to the hunt method (pigs are visibles during sunbathing). If the pig cannot remove itself from the queue, then the pig is dead and the method returns false. The `run()` method now calls both the hunt and sunbathe method in a loop, until one of them returns false.

After a successful hunt the pig writes `"Bless me, Father, for I have sinned."` to the console.

Improved parallelisation - 2 pont

Use the registration / deregistration mechanism as a form of mutual exclusion. If used properly, multiple synchronizations can be avoided.

Use explicit locks (`ReentrantLock`)! Use `Condition` instead of wait-notify

Use the interrupt from `pigPool.shutDownNow()` to initiate the end of the program. Surviving pigs get overconfident and yell `"Look on my works, ye Mighty, and despair!"`.