Exercise 3.1

1. See BoundedBuffer.java for implementation and BounderBufferTest for testing.

2. Our class is thread safe because no concurrent execution of the methods take() and insert() can result in race conditions. That is because we ensured that there is an happe-before relationship between the method take() and insert(). In particular:

    - Class state: shared variables are LinkedList<T> list, Semaphore slots, Semaphore item, Semaphore mutex

    - Escaping is not possible since every shared state var is defined as private, so can be accessed only by public methods of the class.

    - Publication is safe since every shared variable is private there is no way that other external threads access those

    - Immutability: our class is not immutable because we have methods to modify final objects.

    - Mutual exclusion: is guaranteed by the use of the mutex semaphore

3. We think no because barriers are used to synchronize a group of threads so that each one waits until all the others have reached a certain point of synchronization. They are useful when it is necessary to coordinate a group of threads to perform a certain task at the same time.

4. The fair parameter manages the fairness of the policy used in the queue, if set to true the queuing policy is fair, so the queue is implemented as FIFO. In this example we do not think that this parameter matter since the 3 semaphores decide the order of threads. An example where the fair parameter can be useful is the one of readers and writers with infinite readers, this could be a way to prevent writers from being blocked indefinitely

Exercise 3.2

1. If our class has static attributes, to get and set them we need to use class, calling *synchronized (Person.class).*

2. Our implementation is thread safe: no race condition can occur; in particular:
    a. Class state: one (or could be more) static field which is lastId which is shared among all instances of our class; the access to this variable is always synchronized on Person.class, and if we add more static parameters the methods that access them should do the same thing.
    b. escaping: is not possible since every var is defined as private (so can be accessed only by public methods of the class or by the constructor) and even with getter we never return pointers but always the values of the variables so escaping is not possible.

    c. publication is safe since every var is defined as private, and we use synchronization to access the fields.
    d. the class is not immutable, since you can modify the values of name, address, zip,...
    e. mutual exclusion is given by the use of synchronization on all access to those fields. For static field is guaranteed by the synchronization over Person.class.

3. See Person.java for implementation

4. No, because testing does not prove the absence of error, we can say that we are more confident that it actually works.

Exercise 3.3

1. See OurSemaphore.java for implementation