T1.1

In Java GC never free immediately the memory held. It's very hard to know when GC will collect, it depends on many parameters.

T1.2

Using Java8, you can create a thread with:

- Thread:

```
Runnable runnable = () -> {
    // My runnable code
};

Thread thread = new Thread(runnable);
thread.start();
```

- Executor:

```
ExecutorService executor = Executors.newSingleThreadExecutor();
executor.submit(() -> {
// My runnable code
});
```

- Scheduled Executor
- Future, FutureTask when doing asynchronous programming with Java

When I need real thread that, for example dequeue a queue (not asynchronous programming), I prefer to use Executor. Executor allows you to control easily the number of workers (thread) you want to run). It also limits thread memory impact and thread creation cost.

T1.3

No, it's not a good idea, the Java doc explication:

> **Note that this implementation is not synchronized.** If multiple threads access a hash map concurrently, and at least one of the threads modifies the map structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with a key that an instance already contains is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be "wrapped" using the `Collections.synchronizedMap` method. This is best done at creation time, to prevent accidental unsynchronized access to the map:
>
> ```
> Map m = Collections.synchronizedMap(new HashMap(...));
> ```
>
> The iterators returned by all of this class's "collection view methods" are *fail-fast*: if the map is structurally modified at any time after the iterator is created, in any way except through the iterator's own `remove` method, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an

undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

T1.4

LinkedList is best suited for adding/removing items, but it's very low to find object so if you want to get/set it will be better to use ArrayList.

T1.5

Given the way most browsers load resources, doing less request mean better network performance. So I would recommend combining multiple CSS files into one big file.

The advantages/disadvantages of either solution:

One CSS:

- Advantages:
  - Network performance
- Disadvantages:
  - Maintainability (if you don't use a tool like grunt or gulp to minify and combine them)
  - Website can be displayed without CSS if the CSS file is huge

Multiple CSS:

- Advantages:
  - Smaller cached file, if you change a small part of the CSS, you don't need to reload all CSS
  - If not a SPA, you don't have to load all CSS on each page
- Disadvantages:
  - Too many network requests
  - Loading time

T1.6

Javascript console will print:

```
Lublin
Koblenz
```

The reason is in Javascript this reference the owner of the function and not the object that a function is a method of. If you want to keep original this, you can add var self = this at the beginning of the object and use self instead of this after that.