# Part 2 – Answers

## 1.

Writing tests before writing the API saves you from wasting time on writing code/specs you’ll later throw away. Continuing writing tests throughout the process of creating the API prevents surprises and provides example code that uses the API – that can be used for the documentation of the API.

## 2.

1. כאשר כותבים API – צריך לתכנן אותו כך שיהיה קל לשימוש גם ללא קריאה בתיעוד בכל פעם. זו דוגמא לעיקרון בספרטניזציה שאומר שצריך למזער את התיעוד.
2. ~~עקרון חשוב בספרטניזציה הוא הקריאות של הקוד – ולפי הסרטון, API טוב צריך לאפשר לקוד שמשתמש ב-API להיות קריא.~~
3. When writing the initial spec of the API – one should keep it as short as possible (in order to allow doing modifications quikly). This follow the key rule of Spartanization – making everything as short as possible.
4. An API should be “as small as possible but no smaller”, and when in doubt whether a given functionality is needed – we should leave it out.
5. Minimize the accessibility of everything, e.g. making classes and members as private as possible.
6. Minimize mutability. Classes should be immutable, unless there’s a good reason for them not to be.
7. Functions that change the state should have a name with a verb (“clearInterruptStatus” instead of “interrupted” in the example from the lecture).

# TMP

## 3.

Using Checked Exceptions is bad because it forces the user to deal with each such exception (catch/declare it). It turns out 99% of the times, the user wouldn’t want to deal with exceptions thrown by library code, and instead would prefer to simply get an immediate indication of the failure (if failure is even possible at all).

~~In the video – we learn that having checked exceptions in an API is bad because it makes implementation “leak” into the interface (if later you want to change the implementation such that different exceptions are relevant – it’s a problem).~~

In the lecture we are given the example of the clone function in Cloneable interface, throwing a CloneNotSupportedException – which is a checked exception, and it’s a bad idea. This was a bad idea, since it forces us to catch the exception each time using the clone function, even when we know for a fact that this exception would not be thrown (because the object we’re cloning is an instance of a class implementing Cloneable).