Theory: Libraries

8 minutes 0 / 5 problems solved

If you are already familiar with at least one programming language, you know that any programming language has a human-readable design. Most of them use Latin-script symbols to represent functions, keywords, and operators. However, it's not the language that the machine can understand, that's why we use interpreters, compilers, and assemblers. So, to make the process of creating a machine code easier, we need a high-level interface that allows us to operate with verbal commands which would be transformed to 'zeros and ones' by the program itself.

The same idea can be applied when you want to reuse an existing code that provides you with high-level functions and methods, rather than write it by yourself once more. We are there not to talk about tedious copy-paste but the usefulness of programming libraries.

> High-level means that each function you're calling orchestrates the low-level work for you. For example, some imaginary function `WRITE_DATA` under the hood opens the file, writes data to it, and finally closes the file.

§1. What is a library

A **programming library** is a collection of reusable and redistributable codes that has a well-defined interface to use.

Library provides you with high-level functions and methods. We can expect that a library has documentation to get familiar with the behavior of the inner implementation. You should treat a library as a black box: you have the documentation of its interface, but you don't need to know an implementation. Like programming languages isolate you from working with the machine code, libraries isolate you from working with low-level operations.

So, what do you think: can a big pile of incoherent functions be a library? Of course, but it's unlikely that someone will use it! Let's try to highlight the main features that we find important in terms of usability. A good library:

- belongs to one domain of knowledge, for example reading and writing to files, nothing more
- provides the documentation
- has a clear interface, where the name of each object reflects its function
- does not have malicious code in it
- has tests
- follows programming language's code style

Current topic:

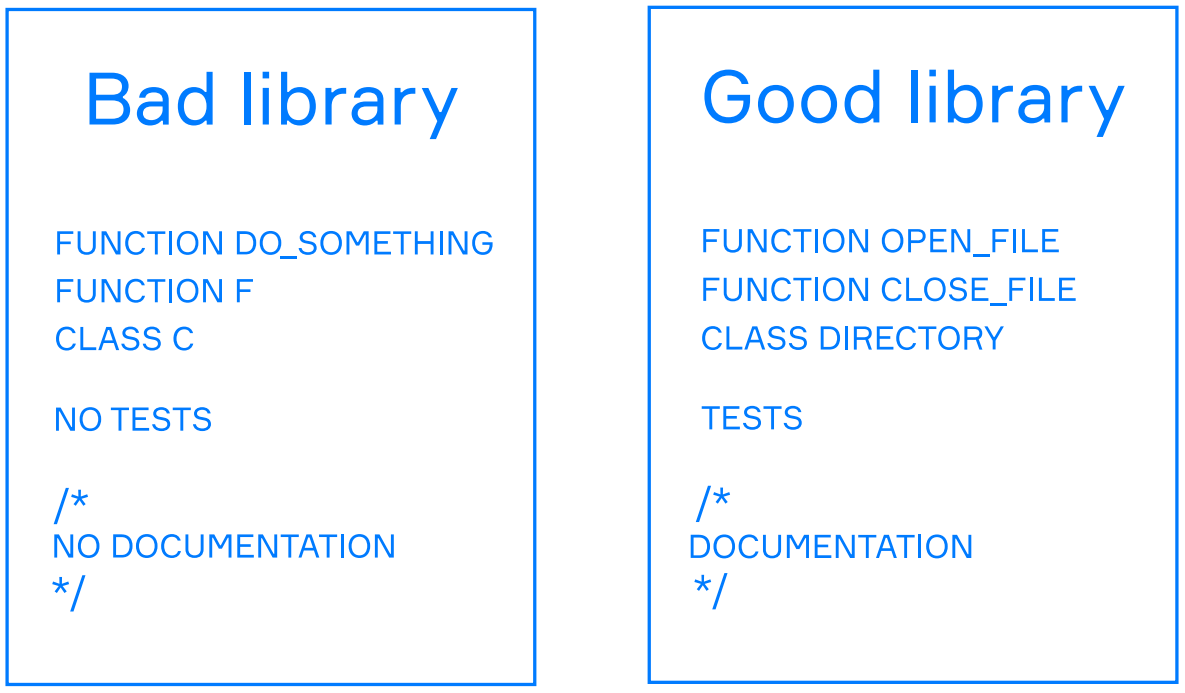
Libraries

Topic is required for:

Introduction to React

The basic window in Swing

Table of contents:



Now you know how to understand whether a library you've chosen is a good-written one. How exactly can we use it?

§2. Standard and third-party libraries

There are several types of libraries, but where you can find them?

Programming language implementations are the usual software that you can install on your computer, and most implementations come with standard libraries. The **standard library** is a stable and standardized collection of modules for the essential needs of the development process. Usually, standard libraries consist of common utilities like working with the file system, making network connections, or parsing JSON files, and are a part of the programming language specification.

The standard library can hardly cover all your needs. For example, you may want to make a desktop application or a web crawler, but the standard library doesn't give you handy tools for that. In this case you can search through the internet to find a third-party library on sites like Github. A **third-party library** is a collection of high-level modules, apart of the standard library of a programming language. Those libraries are often opensource.

To include a library in your program, you should use a keyword and its name. If you use third-party tools, you should look through the documentation and find out how you can install it on your computer first; authors of a library provide this information in the README file.

So, now you could feel like you've learned all the basics about using the libraries. Perhaps you still have a question though: is it more efficient than just writing a needed code by yourself? The answer is: in many cases, yes.

§3. Why use libraries

Libraries are not a silver bullet for all the programming problems. Not all the libraries are mature enough to be used in code production, and some of them are buggy. If the problem is too narrow, it's hard to find a library to solve it even if it exists.

Let's suppose that a library for your problem exists. We cannot cover all the cases, but we can give you several reasons to use it:

- It reduces the time of developing an end product. We can focus on implementing the logic of the application, not on making auxiliary software.
- The development of a library is community-driven. It means many people support a library, and you can join them if a library is an opensource, if you want.
- If a library is popular, many companies and programmers use it. It means that a library has been tested and exploited by different people, and a new developer in your team will likely know this library too.
- Libraries have documentation. You can just read a tutorial and start using a library without learning about inner implementation.

The main goal of libraries is to prevent people from doing the same work twice. You can follow this rule and make your software without distracting on any other issues.

If your code uses a library, you can always replace a library function with your own without breaking the program. You are in control of what you want to use.

Report a typo

266 users liked this theory. **3** didn't like it. **What about you?**

Start practicing

Comments (5) Hints (0) Useful links (0) Show discussion