HW #12

Alexey Iskhakov

Why does your implementation adhere to the S principle?

Every member in my implementation implements only its core functionality and does not do any extra work. Class **"Lamp"**, in its turn, adds an abstraction layer on top of **"Lightsource"** and **"Toggle"** functionalities, making it easier to interact with them. Pretty much as it would be easier to interact with a real lamp, than it is with a lightbulb connected to a switch with some cables.

Why does your implementation adhere to the O principle?

All of classes in my implementation are open for extension: they can become parents to other classes providing the same abstraction and no side effects.

All of classes in my implementation are closed for modification, allowing to manipulate their properties only through methods that are dictated by interfaces.

Why does your implementation adhere to the L principle?

All parent classes in my implementation, due to provided abstractions, are substitutable by their sub-types. For example, abstract class **"Lightsource"** has 2 descendants (**"Lightbulb"** and **"LED"**) that can be freely interchanged without the user even noticing the difference, since they adhere to the **L** principle.

Why does your implementation adhere to the I principle?

All classes in my implementation implement their interfaces fully and respect their requirements, providing only the requested functional. No more, no less.

Why does your implementation adhere to the D principle?

None of classes in my implementation depends on other classes – only on interface and (a once) abstract class. Therefore – they only communicate through interfaces, keeping the abstraction level as high as possible. All of classes' attributes and details are either there because of the abstractions (depend on interfaces) or do not affect the abstraction at all.