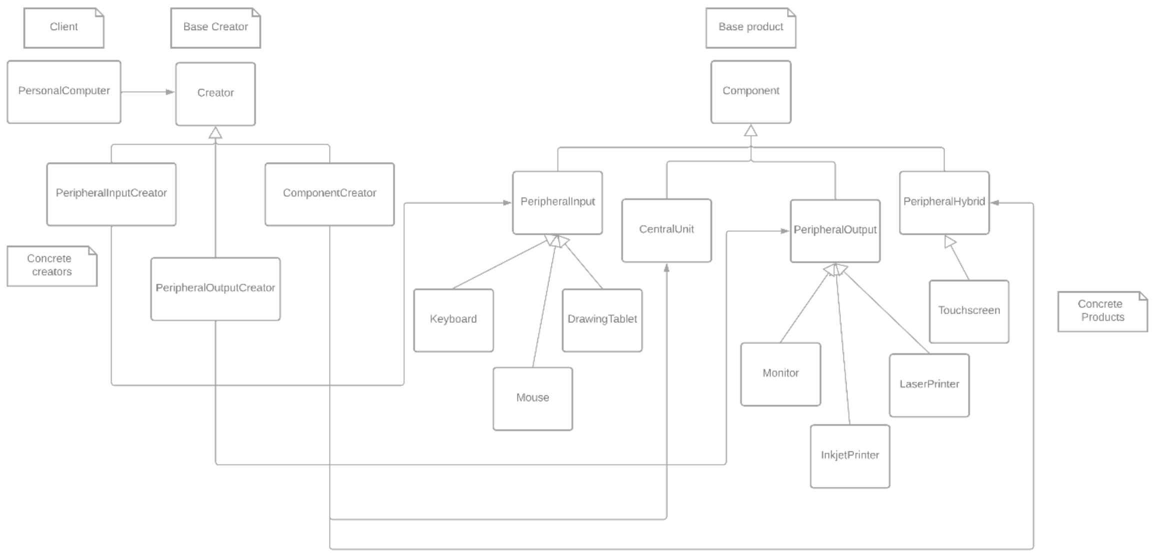
Computer Sale Documentation

Ricardo Antonio Gutiérrez Esparza

In this approach, the **Factory Pattern** was implemented to create the components that integrate a personal computer. The client code uses three concrete creators, one for peripheral inputs, one for peripheral outputs and one for additional types of components (central unit and hybrid peripherals) to instantiate concrete classes according to the type passed to the factory method and add them to the PersonalComputer class.

Below, you can see a simplified class diagram, which shows the relations between classes and the factory pattern.



I chose to use Factory Pattern because there are clearly different types of components that derive from the same class and that the client doesn't need to know its concrete type on creation. We can defer the concrete type to the concrete creators. There is no need for an Abstract Factory because each component is independent from each other.

Some improvements to this approach that I can think of are:

1. Implement a builder in top of the factories. Originally, I didn't think of using the Builder Pattern because we also need a way of removing components, however, it might be useful to have it for the first iteration of the build (when choosing the central unit and the two required peripherals).
2. Instead of having the PeripheralHybrid class, we can replace PeripheralInput and PeripheralOutput with interfaces, and make each component implement the corresponding one (the class Touchscreen must implement both). I didn't do it this way because interfaces don't allow to write code in them, so some of the functionality might have to be repeated on each concrete class, but it's an interesting option to explore.