**question4**

One instance where using a design pattern may not be beneficial is with the Template Method Pattern, as shown in the example of preparing tea and coffee. In this case, a superclass defines the *prepareRecipe()* method, outlining the general steps for making a beverage, while subclasses like Coffee and Tea implement specific actions such as *brew()* and *addCondiments().* Although this pattern promotes reuse and consistency, it becomes unnecessary when there's only one subclass in use, like Coffee. In class, we noted that both Tea and Coffee had identical steps like *boilWater()* and *pourInCup(),* which led to the creation of a shared superclass to avoid duplication. However, if you're only working with a single beverage or don’t anticipate adding more, introducing an abstract superclass and multiple method overrides creates extra complexity without a clear benefit. It adds layers that make the code harder to follow and maintain, especially when a simpler, direct approach would be more effective. This shows how using a pattern in the wrong context can get in the way of clarity and simplicity.