In requirement (Week 5, 3c), the overall design is as follows :

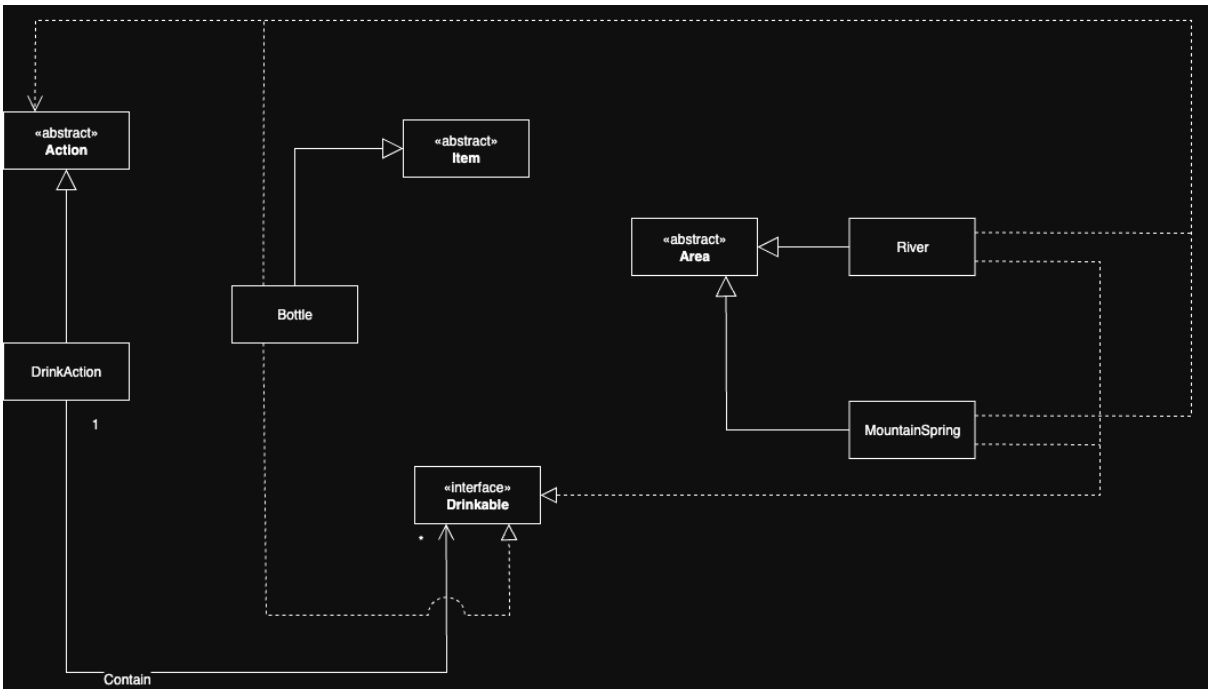

Figure 1.

In short, there is Drinkable interface, DrinkAction concrete class, and the classes that implements Drinkable interface, which are Bottle, MountainSpring, and River classes. Outside of our main focus, Bottle, MountainSpring, and River class are child class of Area abstract class, which implements the Explorable interface which in turn extends the ActionCapable interface. Apart from that, Bottle, MountainSpring, and River classes implements Drinkable interface and DrinkAction class have an association with Drinkable interface, it also extends the Action Abstract class. Further relations and design can be seen from Figure 1.

Single Responsibility Principle:

Definition : "A class should have only one reason to change, meaning that a class should have only one job." - Robert C. Martin

I design the fulfilment of this requirement by making/seperating Drinkable and DrinkAction class, even though they both are related, because I wanted to separate their concern/responsibility. I design the Drinkable interface as the blue print for the method, that is implemented in each Bottle, MountainSpring, and River classes respectively, so that it only focuses on how the drunkBy method is implemented in each class respectively. On the other hand, I design DrinkAction class so that this class will only focus on actually executing the drink action, this class will not worry about the implementation of how a camper drink.

This separation of responsibility adheres to the Single Responsibility principle and separation of concern.

Dependency Inversion Principle and Open Closed Principle:

Dependency Inversion Principle:
Definition : a concrete class should not depend on another concrete class. Instead, it should depend on abstractions.

Open Closed Principle:
Definition : Classes should be open for extension but closed for modification.

I design a Drinkable Interface to connect the concrete class drinkable object (Bottle, River, and MountainSpring) to concrete class DrinkAction, this is done to adheres to the Dependency Inversion Principle. I made sure so that DrinkAction concrete class does not directly depends on the concrete class drinkable object (Bottle, River, and MountainSpring), instead a join them through an interface (abstraction), this results in higher flexibility/changes to Drinkable object (Bottle, River, and MountainSpring) . This is also done so that it can be extended (Adding more concrete class drinkable object) without modifying DrinkAction concrete class, we can extend (adding more drinkable object) just by implementing the Drinkable interface, it doesn't have to modify any of the Drinkable interface or DrinkAction code.