The Drinkable and Explorable interfaces allow different locations like River and MountainSpring to implement the same consistent functionality without having to initialize the same code in each respective file. However, this can introduce repetitive allowableActions() methods.

Each location measures its own impact on the camper's global stats (i.e. changing Coldness and Hydration). This ensures high cohesion and clear responsibility, improving readability and maintainability, but repeated hydration and coldness adjustments may reduce efficiency.

ExploreAction and DrinkAction use abstractions rather than concrete classes, meaning the camper is interacting with behaviours instead of specific locations. This layer of abstraction may complicate debugging.

Using the SOLID principles, we see from the code:

- Single Responsibility Principle (SRP) - River and MountainSpring handles its own respective actions; these locations' actions do not clash.
- Open/Closed Principle (OCP) - new locations can be added without altering existing locations/code;
- Interface Segregation Principle (ISP) - separate interfaces (Drinkable and Explorable) to ensure classes only implement relevant behavior.
- Dependency Inversion Principle (DIP) - ensures actions (Drinkable and Explorable) rely on abstractions instead of concrete implementations.

Pros	Cons
● New environments can be integrated with small changes. ● Each class is small, cohesive, and documented - it is easy to add more actions or location types. ● Design is very flexible and provides adaptability and space for a lot of potential features.	● River and MountainSpring repeat similar logic (allowableActions, hydration/coldness adjustments). ● Maintaining balance with hydration/coldness logic may become harder without a shared abstraction as actions increase.