Design rationale for Requirement 2:

Requirement 2 comprises two main components.

Firstly, I devised the abstract Tree class as a foundational structure derived from the Ground class, extending it to create the Inheritree class and the MatureTree class. This design choice allows for potential future expansions, such as incorporating a DecayTree or other tree types, all of which can extend from the Tree class. By centralizing common attributes and behaviors within the abstract Tree class, I adhered to the principle of **DRY (Don't Repeat Yourself).** In the case of the Inheritree class, which transitions to the next tree type after five ticks, I opted not to implement a generic interface for all growable entities. Instead, I stored the reference to the next tree type within the Inheritree class itself. This decision streamlines the growth process, as it is managed by the Inheritree's tick method rather than Actor which is relying on external actions. As a result, there was no need for a Growable interface, as growth is exclusively handled by the tick method.

Moreover, I introduced the Utility class to manage randomness, in line with the **Single Responsibility Principle**. This class serves as a centralized component responsible for handling all random behaviors, ensuring that each class maintains a single, clearly defined responsibility.

Secondly, in the context of the Fruit class, I implemented a CloningFactory to guarantee that each dropped fruit instance is unique. This approach reinforces the principle of **DRY**, as it obviated the need for separate ActorFactory and ItemFactory implementations. Both of these factories shared the common responsibility of generating new instances based on input parameters. Additionally, the Fruit class was designed as an abstract entity, with LargeFruit and SmallFruit classes extending from it. By having the Tree class drop instances of Fruit, I adhered to the **Open-Closed Principle**, as the Tree class can be extended to drop different types of Fruit without requiring modifications to its core implementation.