The threads are designed and coordinated by GameEngine.java. In GameEngine.java, the threads are organised in an ArrayList. Threads can be added and managed as needed. The subclasses of Barbie contain the run method from the threads, allowing them to be created and managed by the Game Engine. Each Barbie follows its own adventure, while having shared adventures with other Barbies. Each subclass of Barbie has its own shared resource that is used with the collection of Barbies. This creates a potential race condition between the Barbies because they are all trying to access the same resources at the same time. My solution to this issue was to synchronise functions for the shared resources(PearlChest, MagicDustBag, MusicBox). The use of synchronisation eliminates the race condition that arises when the resources are being accessed at the same time.

My coordination mechanisms used were the synchronized functions. The functions not only eliminate race conditions, but are overall most practical for the use of multithreading. Multithreading creates many scenarios where issues with race conditions can arise. The synchronisation of functions allows for logical implementations of concepts we learned in class to be solutions to issues found in multithreading.

The subclasses of Barbie run concurrently with each other when the threads are active. This could also create a race condition. To prevent the race condition, I carefully structured the threads and had help in structuring the run methods to ensure the totality of the project.