Синхронизатор Phaser -  как и CyclicBarrier, является реализацией шаблона синхронизации [Барьер](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D1%80%D1%8C%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), но, в отличии от CyclicBarrier, предоставляет больше гибкости. Этот класс позволяет синхронизировать потоки, представляющие отдельную фазу или стадию выполнения общего действия. Как и CyclicBarrier, Phaser является точкой синхронизации, в которой встречаются потоки-участники. Когда все стороны прибыли, Phaser переходит к следующей фазе и снова ожидает ее завершения.

Проще говоря, мы используем Phaser, если у нас есть программа, которая разбита на фазы. И мы не можем начать следующую фазу, пока не закончили предыдущую.  
В чем отличие:   
**CountDownLatch:**

* Создано с фиксированным количеством потоков
* Счетчик не может быть сброшен

- Позволяет потокам ждать (await()) или продолжить выполнение (countdown()).

**CyclicBarierr:**

* Можно сбросить.
* Не предоставляет метод для продвижения потоков. Потоки должны ждать, пока не прибудут все потоки.
* Создано с фиксированным количеством потоков.

**Phaser:**

* Необязательно, что бы было известно количество потоков во время создания Phaser. Они могут быть добавлены динамически.
* Счетчик может быть сброшен и, следовательно, многоразовым.
* Позволяет потокам ждать arriveAndwaitAdwance() или продолжить выполнение arrive().
* Поддерживает несколько фаз (отсюда и название фазы).

Объект Phaser создается с помощью одного из конструкторов:

Phaser()

Phaser(int parties)

Параметр parties указывает на количество сторон-участников, которые будут выполнять фазы действия. Первый конструктор создает объект Phaser без каких-либо сторон, при этом барьер в этом случае тоже «закрыт». Второй конструктор регистрирует передаваемое в конструктор количество сторон.

Основные методы:

* **int register()** — регистрирует нового участника, который выполняет фазы. Возвращает номер текущей фазы;
* **int getPhase()** — возвращает номер текущей фазы;
* **int arriveAndAwaitAdvance()** — указывает что поток завершил выполнение фазы. Поток приостанавливается до момента, пока все остальные стороны не закончат выполнять данную фазу. Точный аналог CyclicBarrier.await(). Возвращает номер текущей фазы;
* **int arrive()** — сообщает, что сторона завершила фазу, и возвращает номер фазы. При вызове данного метода поток не приостанавливается, а продолжает выполнятся;
* **int arriveAndDeregister()** — сообщает о завершении всех фаз стороной и снимает ее с регистрации. Возвращает номер текущей фазы;
* **int awaitAdvance(int phase)** — если phase равно номеру текущей фазы, приостанавливает вызвавший его поток до её окончания. В противном случае сразу возвращает аргумент.

Пример:

У нас есть автомойка. В ней моют машины несколько автомойщиков)). Но в автомойке только одно место для машины. Т.е. пока все автомойщики не закончат мыть машину (кто-то моет салон, кто-то стекла и т.д.), мы не можем загнать следующую машину на мойку.

