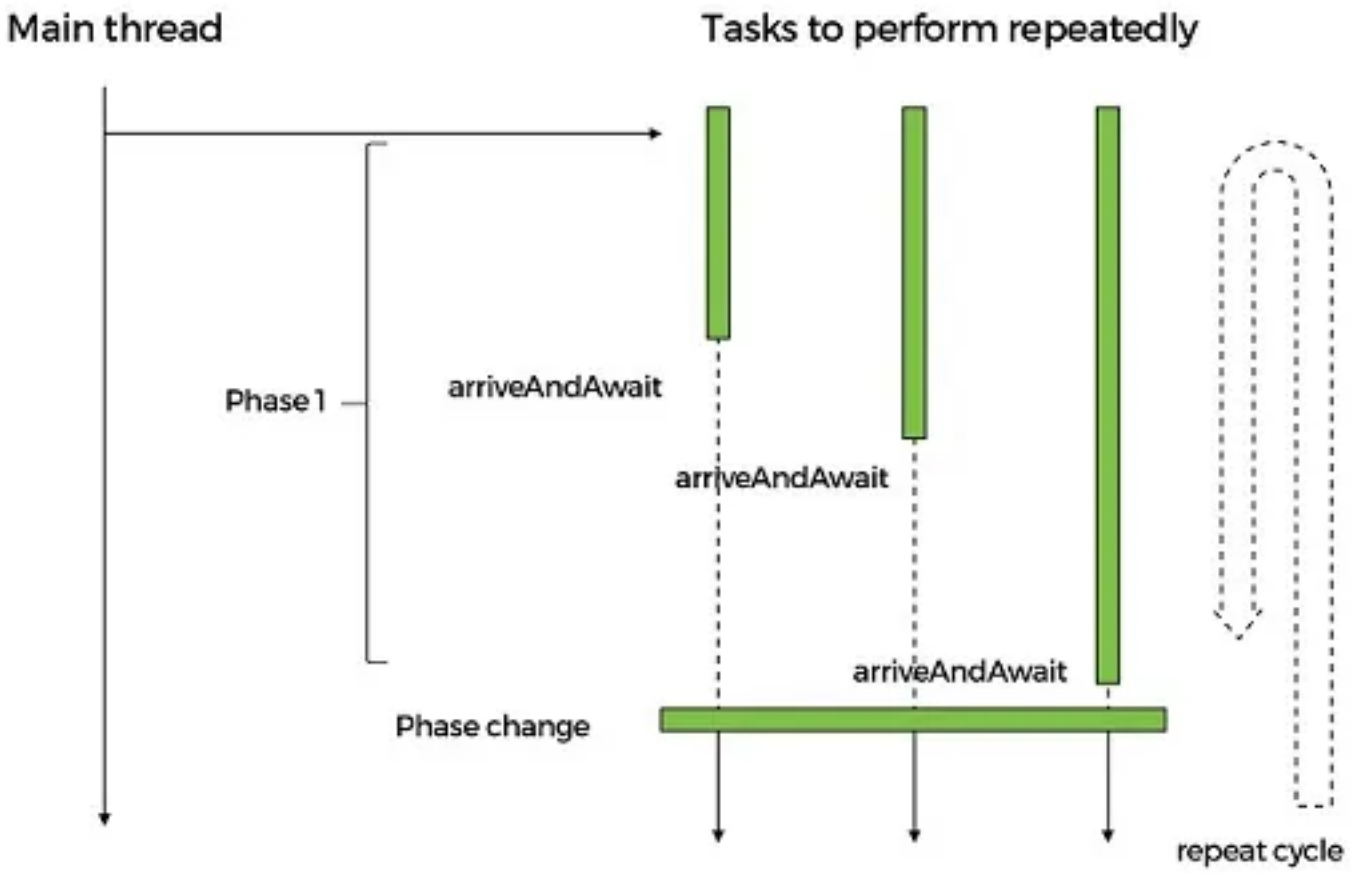
Paradigmes de programmation avancés II

Cours-01: Phaser&Exchanger

Phaser

- Appelé aussi moniteur «pas à pas» permet de délimiter les phases de synchronisation. Par exemple réaliser différentes taches, dans différentes étapes.
- Deux éléments caractérises ses Phasers:
  - Numéro de phase
  - Les parties inscrites
  - Methodes:
    - int register() :s'incrire à un phaser
    - int arrive() : Arrive vers le phaser
    - int awaitAdvance(int phase) : attend que la phase spécifiée en paramètre soit terminée
    - int arriveAndAwaitAdvance() : la partie à atteint la fin de la phase et le phaser s'incrémente de 1

Phaser


Main thread
Tasks to perform repeatedly
Phase 1
arriveAndAwait
arriveAndAwait
arriveAndAwait
Phase change
repeat cycle

Phaser

```
public class Phaser1{
        public static void main(String[] args) throws InterruptedException{
            ExecutorService executor = Executors.newFixedThreadPool(3);
            Phaser phaser = new Phaser (3);
            executor.submit(new Worker(phaser));
            executor.submit(new Worker(phaser));
            executor.submit(new Worker(phaser));
            //Attend que la phase de ce phaseur avance à partir de la valeur
            de phase
            phaser.arriveAndAwaitAdvance();
            System.out.println("Tout est initialise et pas de participation
            coté main ");
            executor.shutdown();
     }

    public static class Worker implements Runnable {
             private Phaser phaser;
             Worker(Phaser phaser){ this.phaser= phaser; }
             @Override public void run(){
             System.out.println("Ready to start.");
             phaser.arriveAndAwaitAdvance();
             doWork();
    }
    public void doWork() { System.out.println("Doing work.");
    }
    }
}
```

Exchanger

- Celui-ci permet à deux threads de s'échanger des objets entre eux. sans synchronisation explicite.
- Les méthodes:
  - V exchange(V x)throws InterruptedException
  - exchange(V x, long timeout, TimeUnit unit)

- Utile pour producteurs-Consommateurs

Exchanger

```
import java.util.ArrayList;
import java.util.concurrent.Exchanger;

class ExchangerProducer extends Thread {
  private Exchanger<ArrayList<Integer>> exchanger;
  private ArrayList<Integer> buffer = new ArrayList<Integer>();

  public ExchangerProducer(Exchanger<ArrayList<Integer>> exchanger) {
    this.exchanger = exchanger;
  }

  public void run() {
    while (true) {
      try {
        System.out.println("Producer.");
        Thread.sleep(1000);
        fillBuffer();
        System.out.println("Producer has produced and waiting:" + buffer);
        buffer = exchanger.exchange(buffer);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }

  public void fillBuffer() {
    for (int i = 0; i <= 3; i++) {
      buffer.add(i);
    }
  }
}
```

Exchanger

```
class ExchangerConsumer extends Thread {
  private Exchanger<ArrayList<Integer>> exchanger;
  private ArrayList<Integer> buffer = new ArrayList<Integer>();
  public ExchangerConsumer(Exchanger<ArrayList<Integer>> exchanger) {
    this.exchanger = exchanger;
  }

  public void run() {
    while (true) {
      try {
        System.out.println("Consumer.");
        buffer = exchanger.exchange(buffer);
        System.out.println("Consumer  has received:" + buffer);
        Thread.sleep(1000);
        System.out.println("eating:"+buffer);
        buffer.clear();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}
public class Main {
  public static void main(String[] args) {
    Exchanger<ArrayList<Integer>> exchanger = new Exchanger<>();
    ExchangerProducer producer = new ExchangerProducer(exchanger);
    ExchangerConsumer consumer = new ExchangerConsumer(exchanger);
    producer.start();
    consumer.start();
  }
}
```

Synchronizers

- CountDownLatch – waits until latch reaches terminal state
- Semaphore – waits until permit is available
- CyclicBarrier – waits until N threads rendezvous
- Phaser – extension of CyclicBarrier with dynamic parallelism
- Exchanger – waits until 2 threads rendezvous
- FutureTask – waits until a computation has completed