

Cognome e nome:

Matricola:

## Prova 1

Si supponga di avere cinque processi che arrivano nel sistema al tempo di arrivo specificato nella seguente tabella, dove sono indicate anche le durate dei CPU burst:

<i>Processo</i>	<i>Tempo di Arrivo</i>	<i>CPU Burst</i>
<i>P1</i>	0	10
<i>P2</i>	6	7
<i>P3</i>	8	5
<i>P4</i>	11	8
<i>P5</i>	18	10

Si mostri la **sequenza di esecuzione** e si calcolino il **tempo di risposta** e il **tempo di completamento** di ciascun processo, considerando i seguenti algoritmi di scheduling:

1. **FCFS**
2. **SJF senza prelazione**

## Prova 2

Si descriva, anche mediante figure opportunamente commentate, come funziona l'allocazione contigua della memoria e in che cosa consiste la frammentazione esterna.

## Prova 3

Si vuole realizzare un sistema per gestire la pesca presso un laghetto artificiale. Al fine di evitare lo spopolamento completo e il sovraffollamento del laghetto il numero di pesci al suo interno deve sempre mantenersi all'interno di un intervallo **minPesci** e **maxPesci**. Il laghetto è frequentato da  $P$  pescatori e vi lavorano  $N$  addetti al ripopolamento. Presso il laghetto può trovarsi un numero qualsiasi di pescatori, o un numero qualsiasi di addetti. **ma mai pescatori e addetti contemporaneamente.**

Ogni pescatore pesca esattamente un pesce alla volta, mentre ogni addetto immette nel laghetto 10 pesci alla volta. L'operazione di pesca richiede a ciascun pescatore un tempo casuale compreso tra 200ms e 800ms, mentre quello di ripopolamento richiede a ciascun addetto un tempo casuale tra 300ms e 600ms. Dopo aver pescato un pesce il pescatore si allontana dal laghetto per 1 secondo dopodiché si riavvicina al laghetto per pescare un altro pesce. Ogni addetto, dopo aver liberato 10 pesci nel laghetto, si allontana dal laghetto per 3 secondi dopodiché si riavvicina al laghetto per ripopolarlo.

Si modelli il sistema descritto in Java, dove i pescatori e gli addetti sono dei thread che interagiscono tramite un oggetto *laghetto* che espone solo i seguenti metodi:

- **void inizia(int t):** permette alla persona di cominciare l'operazione di pesca o di ripopolamento a seconda del valore di  $t$ , dove  $t = 0$  indica "pesca" e  $t = 1$  indica "ripopolamento".
- **void finisci(int t):** permette alla persona di terminare l'operazione di pesca o di ripopolamento a seconda del valore di  $t$ , come per il metodo precedente.

Si implementino due soluzioni che riproducano il funzionamento del problema sopra descritto utilizzando:

- la classe **Semaphore** del package **java.util.concurrent**
- gli strumenti di mutua esclusione e sincronizzazione del package **java.util.concurrent.locks**

Si scriva infine un **main** d'esempio che, facendo uso di una delle due soluzioni precedenti, inizializzi un oggetto laghetto con **minPesci** = 50 e **maxPesci** = 200, inizializzi 40 pescatori, 5 addetti e ne avvii l'esecuzione.