**Programmazione concorrente**

Si consideri il codice dato, in cui il thread main crea tre thread produttori e tre thread consumatori. I produttori e i consumatori terminano dopo un numero casuale di iterazioni.

Si chiede di modificare il codice dato, in modo che ogni volta che un thread produttore o consumatore termina, venga rimpiazzato da un nuovo thread dello stesso tipo (cioè se ad esempio termina un consumatore, viene crerato un nuovo consumatore).

NB: si desidera che sia il thread main a creare i nuovi thread. Anche se a causa del nondetrminismo dello scheduler non è possibile garantire che il nuovo thread verrà creato immediatamente dopo la terminazione del precedente, bisogna fare in modo che non passi troppo tempo tra la terminazione e la creazione successiva. Non sono pertanto accettabili le soluzioni in cui il main va ciclicamente a controllare la situazione dei thread. Sono invece accettabili le soluzioni che, ad esempio, prevedono che i produttori e consumatori informino il main quando stanno per terminare.

Evitare i problemi tipici della programmazione concorrente (corse critiche, starvation, deadlock, ecc.).
Si ricorda che bisogna caricare il file .java (NON i .class) in un unico file zip.

**Programmazione distribuita / socket**

Si consideri il programma dato, in cui il main lancia ad intervalli di durata casuale dei thread (il cui comportamento è irrilevante). NB: il codice dato è lo stesso dell'esercizio su RMI.

Si modifichi il codice dato in modo da ottenere un sistema distribuito in cui il client lancia i thread come nel codice dato, ma lo fa solo quando riceve l'apposito comando dal server.
Il server si limita a inviare comandi al client a intervalli casuali. Pertanto il comportamento del sistema distribuito è del tutto simile al comportamento del programma dato.

Realizzare il sistema usando socket. Per semplicità si può realizzare il sistema con un solo client.

Evitare i problemi tipici della programmazione concorrente (corse critiche, starvation, deadlock, ecc.).
Si ricorda che bisogna caricare il file .java (NON i .class) in un unico file zip.

**Programmazione distribuita / RMI**

Si consideri il programma dato, in cui il main lancia ad intervalli di durata casuale dei thread (il cui comportamento è irrilevante). NB: il codice dato è lo stesso dell'esercizio su socket.

Si modifichi il codice dato in modo da ottenere un sistema distribuito in cui il client lancia i thread come nel codice dato, ma lo fa solo quando riceve l'apposito comando dal server.
Il server si limita a inviare comandi al client a intervalli casuali. Pertanto il comportamento del sistema distribuito è del tutto simile al comportamento del programma dato.

Realizzare il sistema usando RMI. Per semplicità si può realizzare il sistema con un solo client.

Evitare i problemi tipici della programmazione concorrente (corse critiche, starvation, deadlock, ecc.).
Si ricorda che bisogna caricare il file .java (NON i .class) in un unico file zip.