Esercizio 1

```
// Il Thread1 esegue chiamate I/O a System.out che cambiano lo stato del
// thread da Running a Runnable.

// Il Thread2 invece ha un elevato consumo di CPU
// ed utilizza tutto il time-slice messo a disposizione
```

Soluzione 1: Uso di yield

```
for (int i = 1; i <= 50000; i++) {
   S9Esercizio1Yield.sum = i;

   // chiamando yield suggerisco allo scheduler che posso essere messo in pausa
   // permettendo un'esecuzione piu' frequente del Thread1.
   // Questo permette una visualizzazione a video delle somme piu' fluida.
   Thread.yield();
}
```

Soluzione 2: Uso di priority

```
// Per il thread1 definisco la piu' alta priorita'
// per permetterne una piu' frequente esecuzione
thread1.setPriority(Thread.MAX_PRIORITY);

// Per il thread1 definisco la piu' bassa priorita'
thread2.setPriority(Thread.MIN_PRIORITY);

thread1.start();
thread2.start();
```

Esercizio 2

```
while (depots.size() != 3) {
   final Depot randomDepot =
S9Factory.suppliers[random.nextInt(S9Factory.suppliers.length)];
   if (!depots.contains(randomDepot))
      depots.add(randomDepot);
}

// the current implementation is nesting synchronized blocks on different object
// but we can run on deadlock since the order of lock acquisition is random

// try to fix sorting the list by id
depots.sort(Comparator.comparingInt(Depot::getId));

// Th1: 1,3,5   1        3(w2)             5
// Th2: 3,4,5    3        4(w3)        5
// Th3: 4,5,6     4         5(w4)      6
// Th4: 5,6,8      5         6(w5)    8
// Th5: 6,7,9       6     7     9
```