20. Iterator:

a) Tutki kuinka Javan iteraattori käyttäytyy, jos yritetään iteroida kokoelmaa kahdella säikeellä yhtä aikaa, kun molemmilla on oma iterattori.

= Molemmat iteraattorit tulostavat ykkösestä kahteenkymmeneen, mutta sysout tulosteiden perusteella järjestys näyttäisi olevan täysin satunnainen. Esim. thread2 saattaa tulostaa 6 lukua ennen kuin thread1 tulostaa edes yhtä.

b) entä, jos säikeet käyttävät samaa iteraattoria vuorotellen?

= Tulostaa kerran luvut yhdestä kymmeneen, mutta kuten kuvasta näkee järjestys heittelee aika paljon.

```
Thread 2 : 1
Thread 1 : 2
Thread 2 : 3
Thread 1 : 4
Thread 1 : 6
Thread 1 : 7
Thread 1 : 8
Thread 1 : 9
Thread 1 : 10
Thread 2 : 5
```

c) Kuinka käy, jos kokoelmaan tehdään muutoksia iteroinnin läpikäynnin aikana.

= Jos kokoelmaa muokataan säikeen kulkiessa sen läpi iteraattorin avulla, Iterator.next() heittää `ConcurrentModificationException.`

Yllä oleva poikkeus voidaan ratkaista esim. käyttämällä perinteistä for-looppia. Koska se ei käytä iteraattoria kokoelman elementtien läpi kulkemiseen, se ei aiheuta ConcurrentModificationException-poikkeusta.

d) Keksi jotain muuta testattavaa (esim. iteraattorin remove, forEachRemaining).

= Testasin iteraattorin forEachRemaining metodia, joka tulosti listan sisältämät luvut.

```
it.forEachRemaining((number) -> System.out.println(number));
```

metodia käytetään suorittamaan annettu toiminto kullekin jäljellä olevalle elementille peräkkäin nykyisessä säikeessä, kunnes kaikki elementit on käsitelty tai toiminto aiheuttaa poikkeuksen.

”ForEachRemaining-menetelmä ei tarjoa mitään muuta hyötyä kuin se, että se eliminoi tarpeen kirjoittaa while-silmukan.”