# 1. uzdevums

Šis uzdevums ir paredzēts, lai jūs iepazītos ar vispārīgiem principiem veidojot vairāku pavedienu programmas. Šī uzdevuma mērķis ir radīt trīs paveidienus (thread) un izpildīt tos vienlaicīgi. Kamēr tie izpildās, tie izvada savus nosaukumus standarta izvadā (System.out). Aplūkojot, kas tiek izvadīts, jūs varam novērot kā pavedieni izpildās un kādā secībā tie izpildās.

Izveidojiet klasi PrintMe, kura implementē java.lang.Runnable interfeisu. Run metodei vajadzētu veikt sekojošas darbības 10 reizes: izdrukā šī pavediena nosaukumu un gulēt divas sekundes.

Izveidojiet TestThreeThreads klasi. Klases galvenajā (main) metodē tiek izveidoti trīs paveidieni, kuri izmanto PrintMe izpildes klasi (runnable class). Piešķiriet katram pavedienam unikālu vārdu izmantojot setName metodi. Izpildiet katru pavedienu.

# 2. uzdevums

Veiciet nepieciešamās izmaiņas sekojošā kodā tā, lai tas korekti strādātu. Izmantojiet wait un notify.

public class PC {

public static void main(String[] args) {

Buffer b = new Buffer(1);

Producer p = new Producer(b);

Consumer c = new Consumer(b);

p.start();

c.start();

}

}

class Buffer {

private char[] buffer;

private int count = 0, in = 0, out = 0;

Buffer(int size) {

buffer = new char[size];

}

public synchronized void putCharacterInBuffer(char c) {

while (count == buffer.length);

System.out.println("Producing " + c + " ...");

buffer[in] = c;

in = (in + 1) % buffer.length;

count++;

}

public synchronized char getCharacterFromBuffer() {

while (count == 0);

char c = buffer[out];

out = (out + 1) % buffer.length;

count--;

System.out.println("Consuming " + c + " ...");

return c;

}

}

class Producer extends Thread {

private Buffer buffer;

Producer(Buffer b) {

buffer = b;

}

public void run() {

for (int i = 0; i < 26; i++) {

try {

buffer.putCharacterInBuffer((char) ('A' + i % 26));

} catch (Exception ex) {

}

}

}

}

class Consumer extends Thread {

private Buffer buffer;

Consumer(Buffer b) {

buffer = b;

}

public void run() {

for (int i = 0; i < 26; i++) {

try {

buffer.getCharacterFromBuffer();

} catch (Exception ex) {

}

}

}

}

# 3. uzdevums

Java piedāvā java.util.concurrent.Semaphore objektu, kurš atvieglo pavedienu sinhronizāciju. Izpētiet Java API un pārveidojiet iepriekšējā uzdevuma kodu izmantojot java.util.concurrent.Semaphore.

# 4. uzdevums

Izveidojiet klienta-servera risinājumu, kur serveris ir spējīgs apstrādāt vairākus klientus vienlaicīgi, izmantojot ServerSocket, Socket un Thread. Pieslēdzoties serverim klients nosūta teksta ziņu „Hello from <XXX>”, kur <XXX> tiek aizvietots ar studenta vārdu. Serveris pēc šīs ziņas saņemšanas, serveris nosūta klientam teksta ziņu „Hi, <XXX>”.

Lai realizētu iespēju vienlaicīgi veikt vairāku klientu pieslēgšanos, pēc klienta-servera Socket savienojuma izveidošanas, servera pusē ir jāizveido jauns pavediens (Thread), kuram tiek nodota konkrētā savienojuma apstrāde.