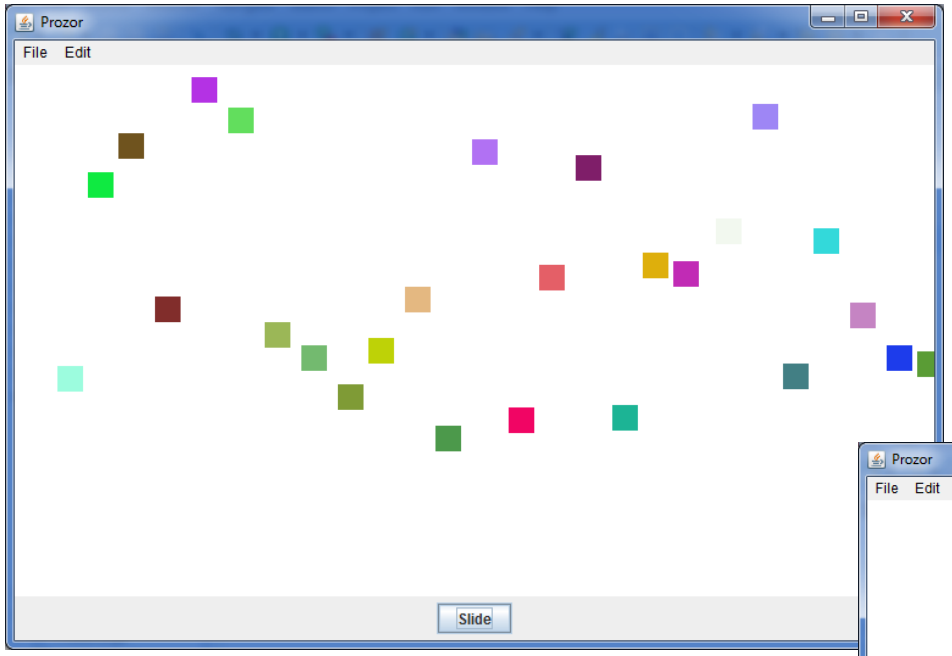
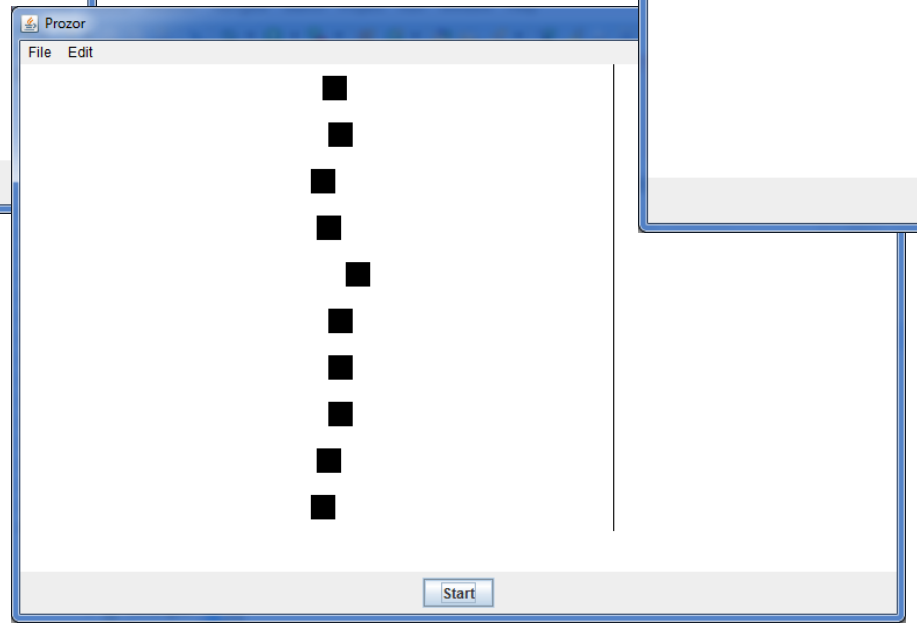
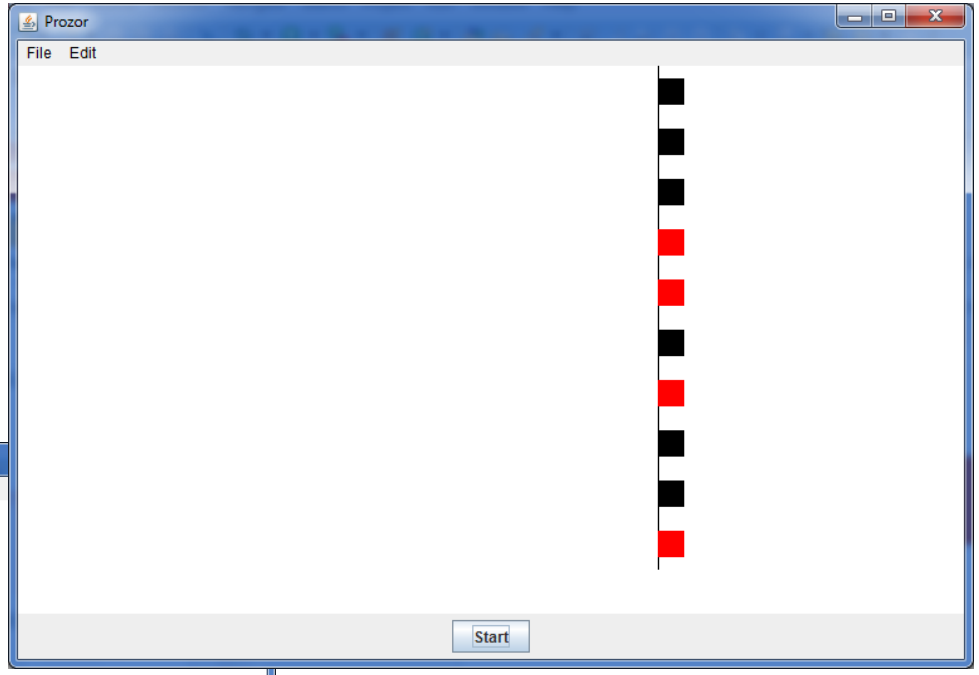
Višenitno programiranje u Javi
sinhronizacija

- Katanci
- Operacije zavisne od stanja
- Monitori u Javi

Šta ako me zanima šta se dešava u drugoj niti?







Konkurentno ponašanje niti

- Ako dve niti mogu da modifikuju i čitaju iste podatke
  - dolazi do efekta neizvesnosti (race condition), a
  - rezultat neizvesnosti trke može biti neispravno stanje nekog objekta.
- Rešenje problema neizvesnosti trke je u sinhronizaciji niti.
- Sinhronizacija niti treba da obezbedi samo jednoj niti **ekskluzivan pristup podacima** kada se nit nalazi u svom **kritičnom regionu**.

Primer 1 – nedozvoljena stanja objekta

ZADATAK

Nad jednim skladište operišu dva radnika. Jedan koji povećava stanje skladišta, drugi koji ga smanjuje. Startno stanje skladišta je 0.

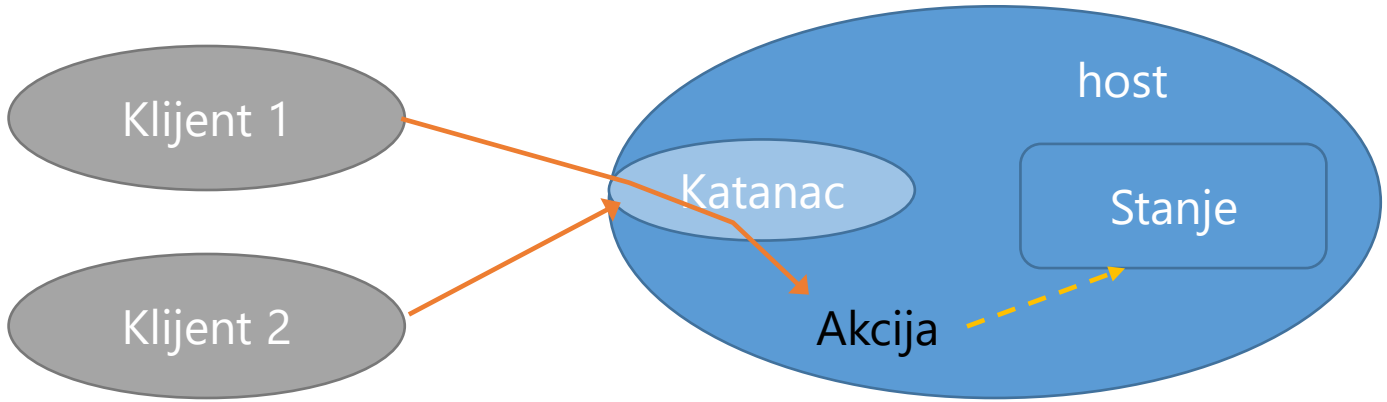
Ako jedan radnik stalno dodaje po 1 proizvod u skladište, a drugi skida i pri tome to urade isti broj puta

> Koje je dozvoljeno stanje u skladištu po obavljanju posla,
> ako smatramo da stanje može biti i negativno?

Šta se zaista dobija?

Java katanci

- Ideja – sinhronizovati rad niti obezbeđivanjem uzajamnog isključivanja rada niti nad zajedničkim objektom.
- Sinhronizacija se postiže implementacijom koncepta monitora.
  Svaki Java Object poseduje jedan **implicitni katanac (lock)**.
- Kada nit zaključa objekat, tj. dobije njegov katanac
  - samo ta nit može da pristupa objektu, što znači da
  - ako za to vreme druga nit zatraži katanac istog objekta biće blokirana i moći će da nastavi sa svojim radom tek kada prva nit katanac otpusti/oslobodi



synchronized

- Zahtev za ekskluzivni pristup objektu, tj. zahtev za dobijanjem katanca na određenim objektom je neophodno ekplicitno naglasiti!
- Zahtevanje katanca se postiže navođenjem ključne reči **synchronized**.

```
public class Location {
private double x,y;
Location(double x, double y) {this.x=x; this.y=y;}

synchronized double x() {return x;}

double y() {
        synchronized(this){
                return y;
        }
}
}
```

ono što se zaključava jeste objekat čiji je sinhronizovani metod pozvan

Primer 2 – proizvodjač/potrošač

ZADATAK

Proizvođač
generiše cele brojeve od 0 do 9, smešta ih u objekat tipa Skladiste, štampa generisani broj i miruje neko slučajno vreme
(0-100 ms)

Korisnik
uzima sve cele brojeve iz Skladista brzinom kojom oni postaju dostupni

Testna klasa

```
… main(String[] args){
 Skladiste m1=new Skladiste();
 Proizvodjac p=new Proizvodjac(m1); Korisnik c=new Korisnik(m1);
 p.start();
 c.start();
}
```

Proizvodjac extends Thread

```
Skladiste m;

run() { …
        m.stavi(3);
        … }
```

Korisnik extends Thread

```
Skladiste m;

public Nabavka(Skladiste m){
 this.m=m;
}
run() {…
        k = m.uzmi();
        …}
```

Skladiste

```
int stanje;

public syncronized int uzmi() { return sadrzaj; }
public syncronized void stavi(int vrednost) { sadrzaj = vrednost; }
```

Uslovna zaštita

```
class Skladiste {
private int sadrzaj;
private boolean otkljucano = false;
public synchronized int uzmi() {
   while (otkljucano == false) {
     try {
          wait();
     } catch (InterruptedException e) { }
   }
       otkljucano = false;
       return sadrzaj;
  }
public synchronized void stavi(int vrednost) {
  sadrzaj = vrednost;
  otkljucano = true;
  notify();
  }
}
```

Komentar

- Zauzimanje i oslobađanje monitora se radi automatski, čime se obezbeđuje da se ne pojavi race condition.
- Na početku uzmi() metode, ukoliko Proizvodjac nije generisao novi broj, onda Korisnik čeka da Proizvodjac postavi novu vrednost u Skladiste.
- Postavlja se pitanje kako može Proizvodjac da postavi novu vrednost ako Korisnik zauzima monitor (Korisnik zauzima monitor zato što je unutar metode uzmi())?
- Ovaj problem je rešen na taj način što kada nit izvršava **wait**() metodu čime se monitor, tj. objekat automatski oslobađa, a sama nit ostaje blokirana. Ovo daje mogućnost da Proizvodjac zauzme monitor i postavi novu vrednost.
- Kada Proivođač završi sa radom i uz put pozove **notify**() metod, blokirana nit korisnika će biti odblokirana i vraćena u runnable stanje.

wait() i notify()

- `wait()`: u atomskoj (neprekidivoj) operaciji suspenduje nit i oslobađa bravu objekta
- `notify()`: obaveštava jednu od niti koje čekaju sa `wait()` da nastavi sa izvršavanjem. Kada se nit ponovo pokrene nakon što je stigao `notify` brava se ponovo zaključava. Metod notify budi samo jednu nit i to onu koja je najduže čekala.
- `notifyAll()`: za buđenje svih niti koje čekaju sa wait().
- Sva tri metoda mogu biti pozvana samo iz sinhronizovanog konteksta.
- Test uslova treba uvek da bude u petlji (ne treba zameniti while sa if)

Sinhronizacija statičkih metoda klase

- Zajedničke metode klase rade nad bravom klase (ne bravom objekta)
- Dve niti ne mogu da izvršavaju sinhronizovane statičke metode nad istom klasom u isto vreme
- Brava klase nema nikakav efekat na objekte te klase
  - jedna nit može da izvršava sinhronizovani statički dok druga može da izvršava sinhronizovani nestatički metod

Java katanci

- Obrada ugnježdenih poziva:
  - pozvan metod se izvršava, ali brava se ne otključava po povratku
- Konstruktor ne treba da bude sinhronizovan
  - Izvršava se u jednoj niti dok objekat još ne postoji pa mu druga nit ne može pristupiti
- Dodatni razlog protiv javnih podataka
  - kroz metode se može upravljati sinhronizacijom

Sinhronizacija i nasleđivanje

- Kada se redefiniše metod u izvedenoj klasi:
  - osobina synchronized neće biti nasleđena
  - redefinisani metod može biti sinhronizovan ili ne bez obzira na odgovarajući metod roditeljske klase
- Novi nesinhronizovani metod neće ukinuti sinhronizovano ponašanje metoda roditeljske klase
  - ako novi nesinhronizovani metod koristi super.m() objekat će biti zaključan za vreme izvršenja metoda m() roditeljske klase

Korišćenje postojeće klase

- Ako želimo da u okruženju sa više niti koristimo klasu koja je već projektovana za sekvencijalno izvršenje u jednoj niti (ima nesinhronizovane metode)
- Dva rešenja:
  - kreirati izvedenu klasu sa sinhronizovanim redefinisanim metodama i prosleđivati pozive koristeći super
  - koristiti sinhronizovanu naredbu za pristup objektu
- Prvo rešenje je bolje jer ne dozvoljava da se zaboravi sinhronizacija naredbe za pristup