JAVA KONKURENTNO PROGRAMIRANJE

Aleksandar Kartelj
Matematički Fakultet, Beograd

kartelj@matf.bg.ac.rs
http://www.matf.bg.ac.rs/~kartelj

Teme

- Konkurentnost
  - Modeli, dizajn, Java
- Dizajn konkurentnih objekata
  - Neporomenljivost, zaključavanje, zavisnost stanja...
- Uvođenje konkurentnosti u aplikacijama
  - Autonomne petlje, jednosmerne poruke, interaktivne poruke, otkazivanje
- Konkurentne arhitekture aplikacija
  - Tok, paralelizam, slojevitost
- Biblioteke
  - Korišćenje, izgradnja i dokumentovanje reiskoristivih konkurentnih klasa

Konkurentnost

- Zašto da?
  - Dostupnost
    - Minimizacija vremena čekanja na odgovor, maksimizacija protoka
  - Modelovanje
    - Simuliranje autonomnih objekata, animacije
  - Paralelizam
    - Iskorišćavanje multiprocesorskog okruženja, više I/O istovremeno
  - Zaštita
    - Izolovanje aktivnosti u okviru niti
- Zašto ne?
  - Kompleksnost
    - Sigurnosni problemi, kompozicija aktivnosti
  - Preopterećenje
    - Visoko korišćenje resursa

Tipične primene

- I/O zavisni pristupi
  - Konkurentni pristu web stranicama, bazi podataka, soketima..
- GUI
  - Konkurentno hvatanje događaja, osvežavanje ekranskih kontrola..
- Izvršavanje stranog koda
  - Konkurentno izvršavanje apleta, JavaBeans...
- Demonski procesi (na Serverima obično)
  - Konkurentno opsluživanje više klijentskih zahteva
- Simulacije
  - Konkurentno simuliranje više realnih objekata

Konkurentno programiranje

- Konkurentnost je **konceptualno** svojstvo softvera
- Konkurentni programi mogu imati sledeća svojstva:
  - **Operabilnost nad više CPU**: klasteri, arhitekture spec. namene
  - **Paralenost**: mapiranje softvera na više CPU, radi poboljšanja performansi
  - **Deljeni pristup resursima**: objekti, memorija, fajl deskriptori, soketi..
  - **Distribuiranost:** konkurentni programi koji NE dele resurse

Objektni modeli

Modeli opisuju način gledanja na objekte

Uobičajeni entiteti:

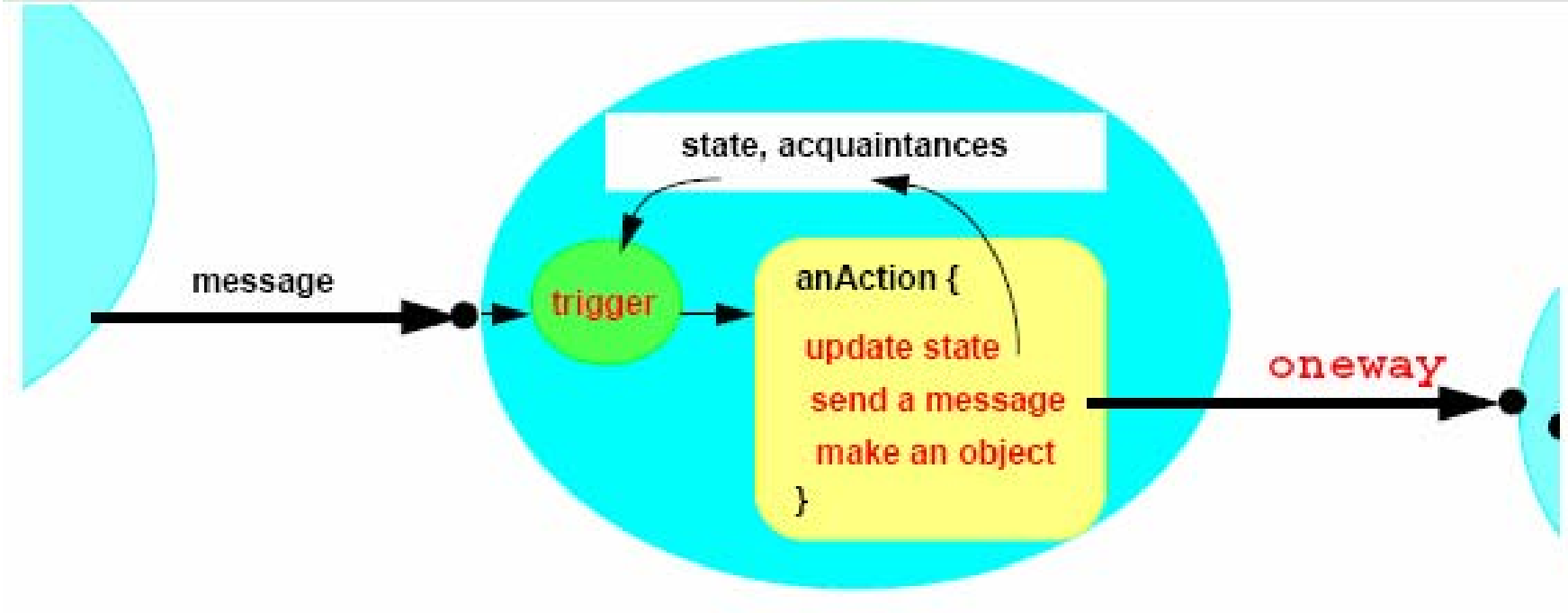
- Klase, stanja, reference, metodi, identiteti, ograničenja
- Enkapsulacija

Četiri osnovne operacije

- Prihvatanje poruke
- Ažuriranje lokalnog stanja
- Slanje poruke
- Kreiranje novog objekta

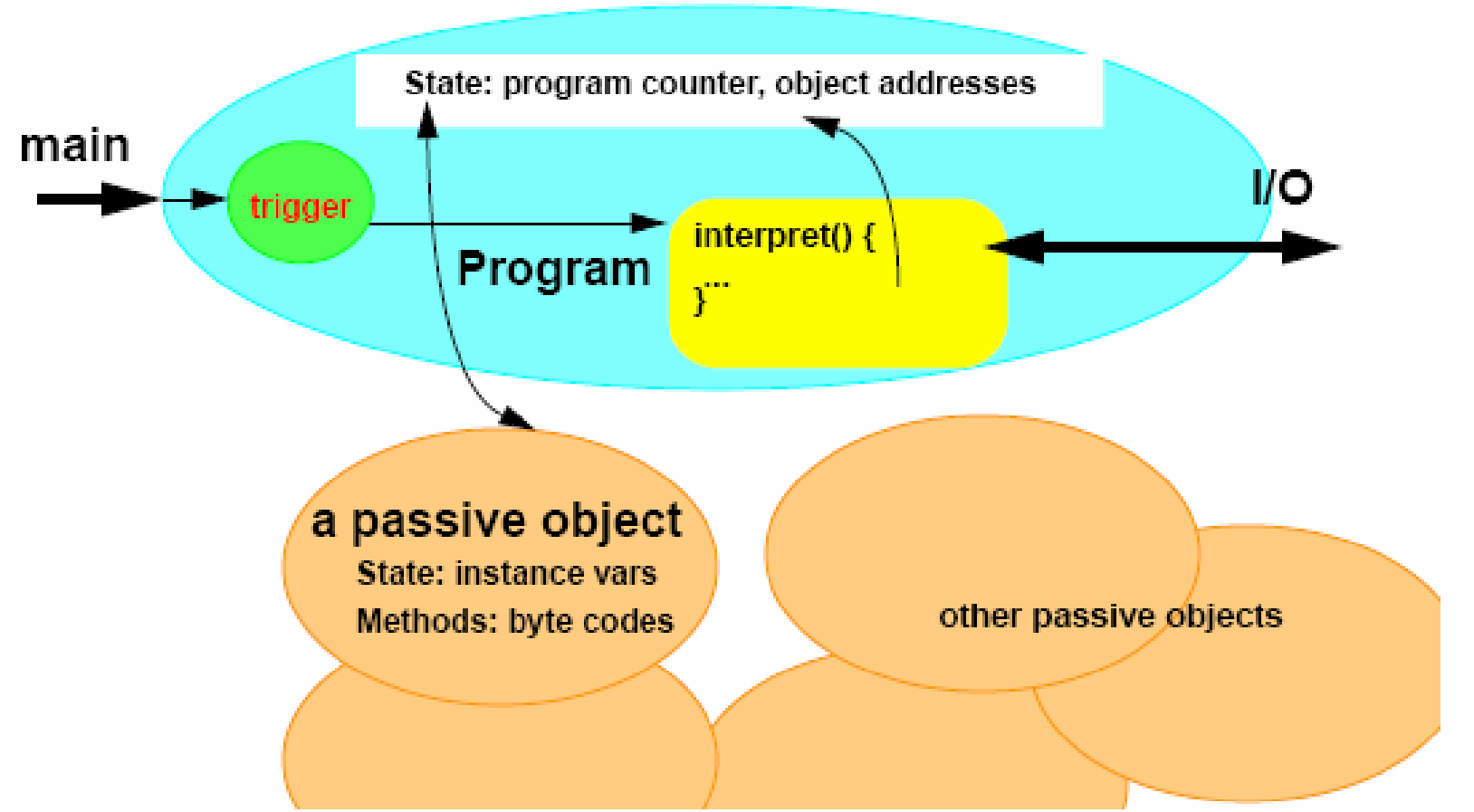
Modeli se obično razlikuju u pravilima izvršavanja ovih operacija. Dve osnovne kategorije: AKTIVNI I PASIVNI

Aktivni objektni modeli

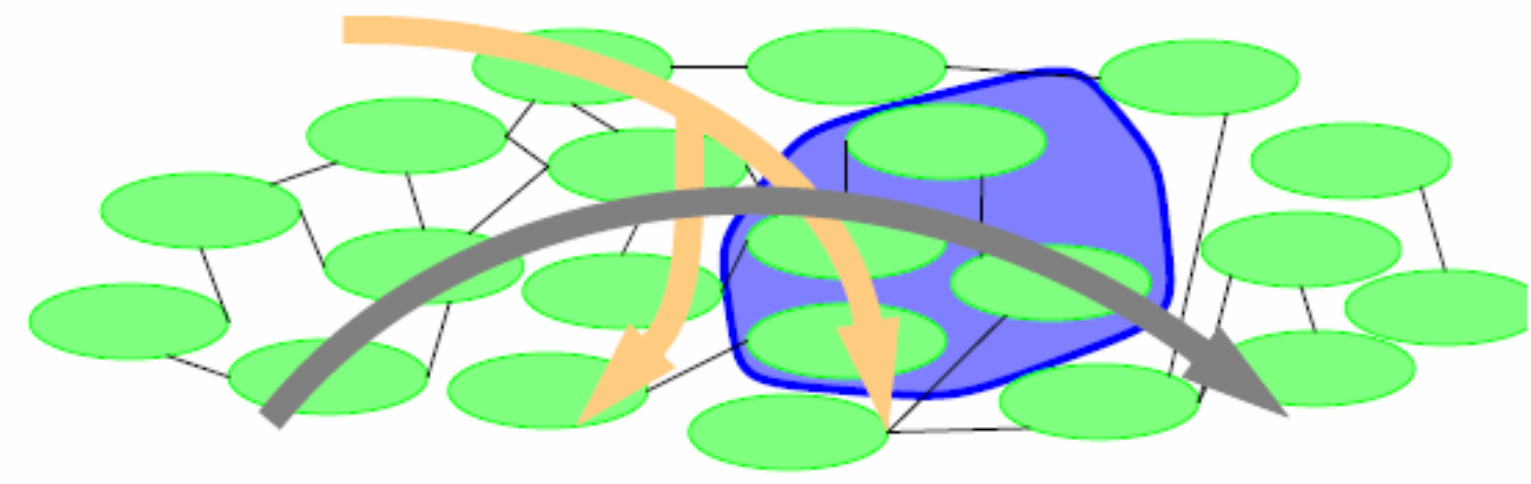


- Jedna nit = jedna radnja u momentu (kao proces)
- Akcije su reakcija na primljene poruke
- Sve poruke su jednosmerene
- Asinhrona, sinhrono razmenjivanje poruka, redovi čekanja...

Pasivni objektni modeli



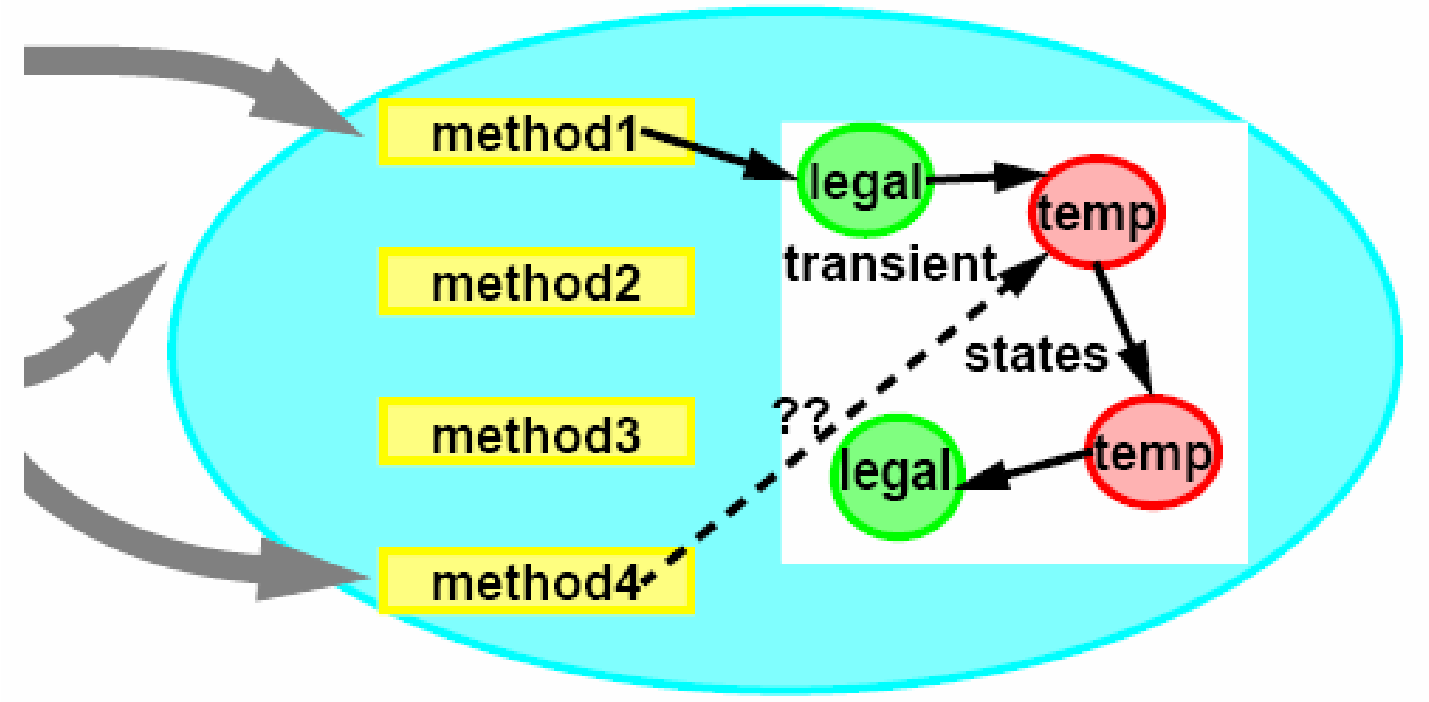
Sistem = Objekti + Aktivnosti



- **Objekti:** strukture podataka, komponente, JavaBeans, RMI objekti…
  - Mogu biti korišćeni od strane više aktivnosti – fokus je na sigurnosti
- **Aktivnosti:** poruke, niti, sesije, skripti, transakcije, tokovi podataka...
  - Fokus je na efikasnosti

Sigurni objekti

- Izvršavaju akcije **samo** kada su u konzistentnom stanju



Primeri nekonzistentnih stanja

- Istovremeno iscrtavanje i pomeranje figure na ekranu
- Povlačenje sredstava sa bankovnog računa u toku transfera novca
- Čitanje sa memorijske lokacije dok je u toku pisanje po istoj

Efikasne aktivnosti

- Svaka aktivnost bi trebalo da vrši napredak ka završenju i to što je pre moguće
  - Svaki pozvani metod bi trebalo u nekom momentu da se izvrši, a ideja je napraviti da se to desi što je pre moguće (efiksanost)

- Aktivnost može da se ne završi (nije efikasna) ukoliko:
  - Objekat ne može da primi poruku
  - Metod blokira čekanje na događaj, poruku ili uslov
  - Nedovoljni ili nepravedno raspodeljeni resursi
  - Različite greške i padovi sistema...

Problem dizajna konkurentnih programa

- Dva ekstremna pristupa:
  1. **Sigurnost na prvom mestu**
     - Pobrinuti se za sigurnost prvo, pa onda pokušati sa optimizacijom efikasnosti
     - Karakteristično za top-down OO dizajn
     - Može rezultirati sporim izvršavanjem i mrtvim stanjima (deadlocks)
  2. **Efikasnost na prvom mestu**
     - Dizajnirati održiv sistem, a onda pokušati sa poboljšanjem sigurnosti kroz koncepte zaključavanja i čuvanja resursa
     - Karakteristično za višenitno (multithreaded) sistemsko programiranje
     - Može rezultirati „bagovitim“ kodom

Garantovana sigurnost

„Nikad se ne izvrši nešto loše“

- Manifest na niskom nivou
  - Bitovi se uvek pravilno interpretiraju
  - Nema memorijskih konflikta tipa read/write ili write/write
- Manifest na visokom nivou
  - Objekti su dostupni samo kada su u konzistentnom stanju
  - Invarijantnost stanja

Garantovana efikasnost

„Uvek se bar nešto izvrši“ – uslov napretka

- Dostupnost
  - Izbegavanje bespotrebnog blokiranja
- Napredak
  - Izbegavanje zadržavanja resursa među više aktivnosti
  - Izbegavanje deadlock-a
  - Izbegavanje nepravednog organizovanja rasporeda
- Zaštita
  - Izbegavanje sukoba sa drugim programima
  - Sprečavanje „denial of service“ napada
  - Sprečavanje zaustavljanja rada pod uticajem stranih „agenata“

Pregled Jave

- Srž Jave (java core) je relativno mali i „dosadan“ objektno-orijentisani jezik
- Glavne razlike u odnosu na C++
  - Sigurnost u pogledu izvršavanja korišćenjem virtualne mašine
    - Nema nesigurnih operacija niskog nivoa (rad sa pokazivačima)
    - Automatsko skupljanje „đubreta“ (garbage collection)
  - Potpuno klas bazirana: nema globalnih promenljivih
  - Relativno jednostavnija: nema višestrukog nasleđivanja...
  - Objektne implementacije Array, String, Class...
  - Velike predefinisanje biblioteke klasa: AWT, Swing, Applets...

Java koncepti

- **Packaging: Objects, classes, components, packages**
- **Portability: Bytecodes, unicode, transports**
- **Extensibility: Subclassing, interfaces, class loaders**
- **Safety: Virtual machine, GC, verifiers**
- **Libraries: java.* packages**
- **Ubiquity: Run almost anywhere**

Napredni java koncepti

- **Concurrency: Threads, locks, ...**
- **Distribution: RMI, CORBA, ...**
- **Persistence: Serialization, JDBC, ...**
- **Security: Security managers, Domains, ...**

Osnovni java konstrukti

- **Classes – kalupi objektnih svojstava**
  - **Instance variables – polja koja predstavljaju stanje objekta**
  - **Methods – enkapsulirane procedure**
  - **Statics – polja i metodi vezani za klasu, a ne konkretnu instancu objekta te klase**
  - **Constructors – operacije prilikom kreiranja objekta**
- **Interfaces – skupovi deklarisanih metoda**
- **Subclasses – potklase (sve nasleđuju klasu Objekat)**
- **Inner classes – Klase unutar drugih klasa**
- **Packages – prostori imena za organizovanje skupova srodnih klasa**

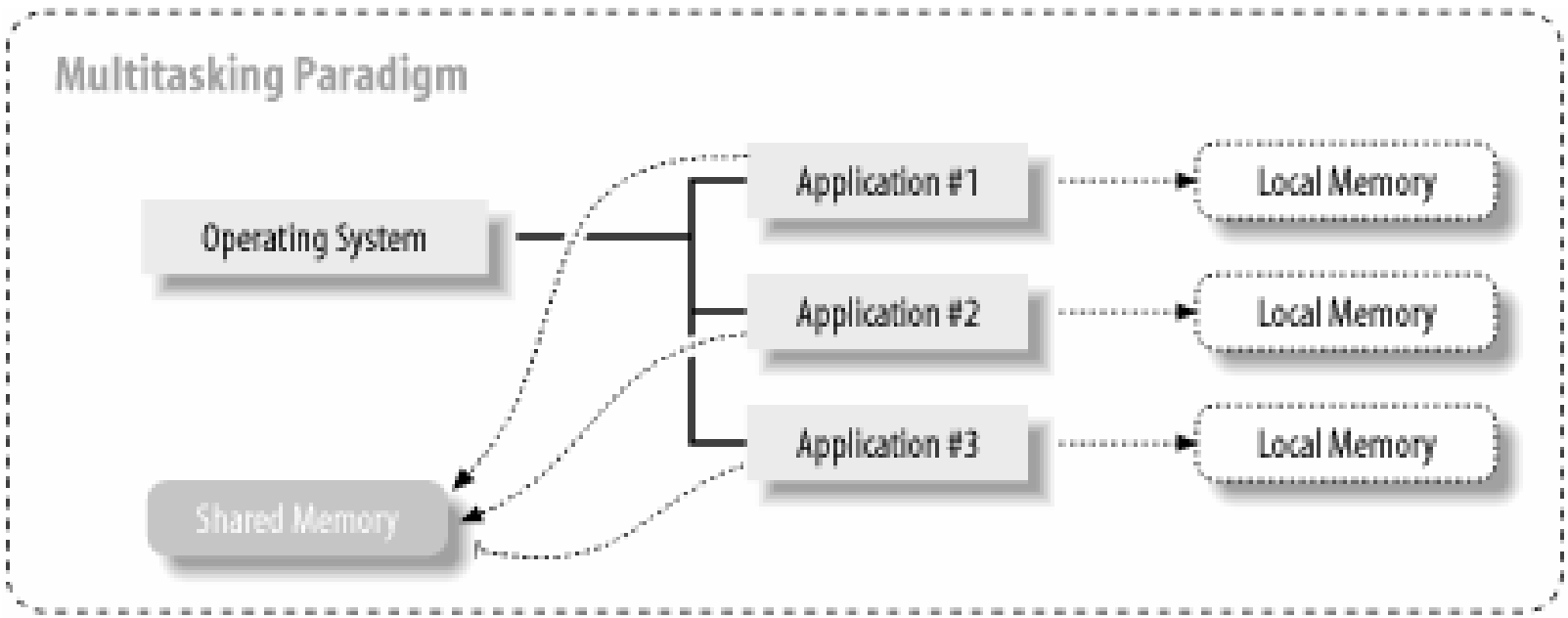
Osnovni java konstrukti

- **Visibility control - private, public, protected, per-package**
- **Qualifiers - semantička kontrola: final, abstract, etc**
- **Statements - naredbe su slične kao u C/C++**
- **Exceptions - Throw/catch kontrola u slučaju izuzetaka**
- **Primitive types - byte, short, int, long, float, char, boolean**

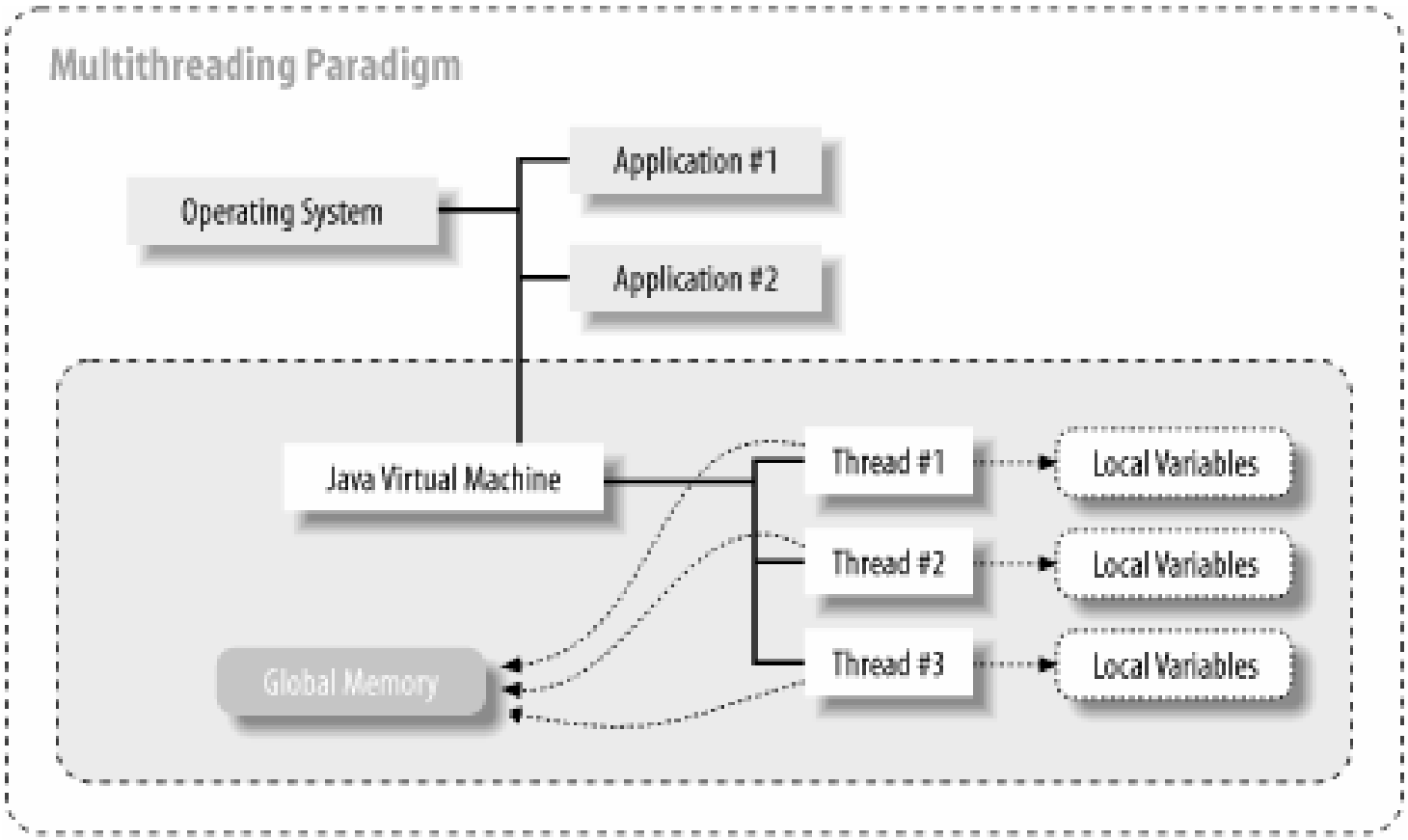
Particle applet

- t_particle\ParticleApplet.java

Multitasking

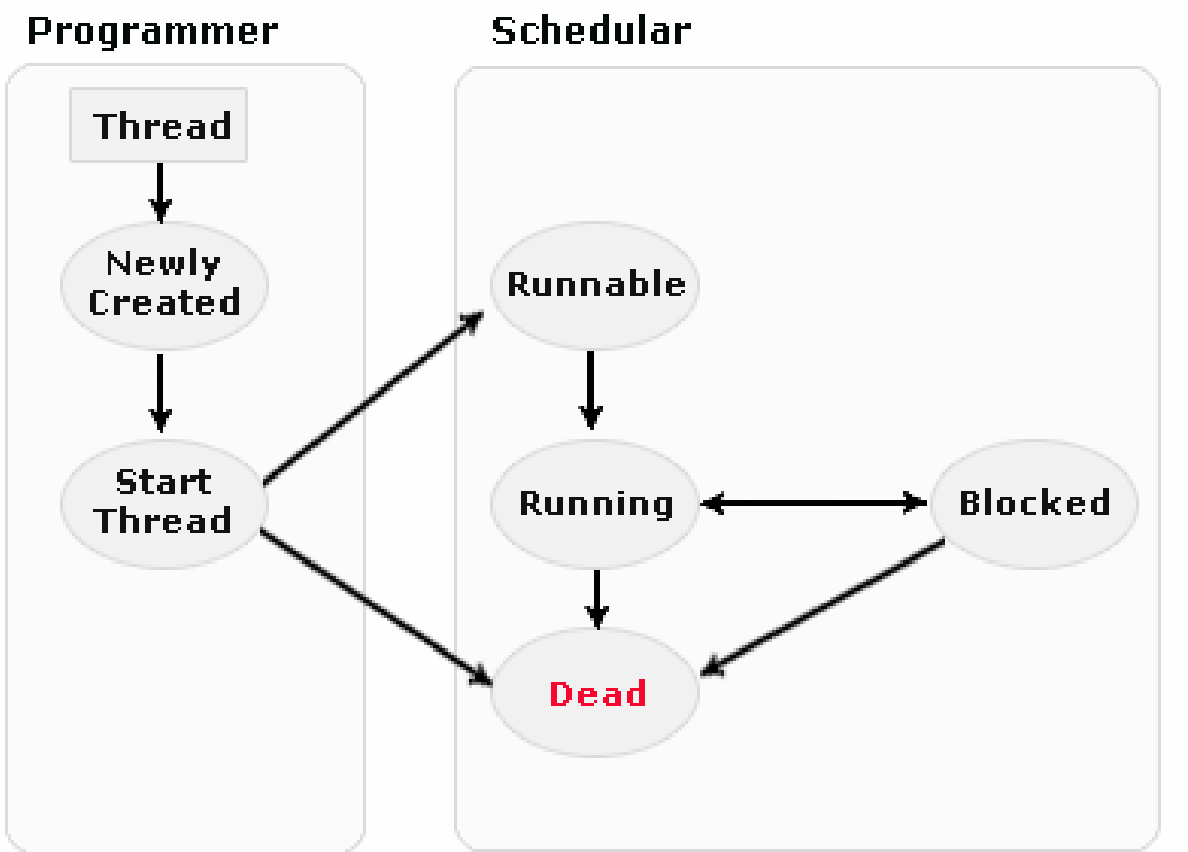


Multithreading

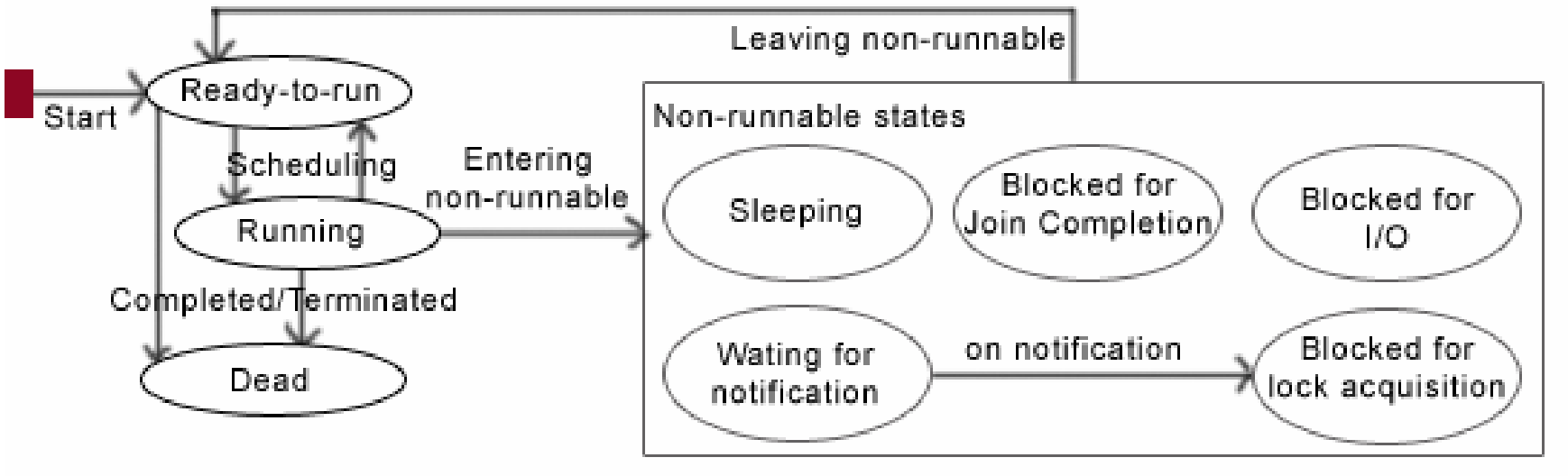


Životni ciklus niti

- http://www.roseindia.net/java/thread/life-cycle-of-threads.shtml



Životni ciklus niti - detaljnije



Java podrška za konkurentnost

- **Nit (thread) klasa predstavlja stanje jedne nezavisne aktivnosti i ima sledeće osobine:**
  - **Metode start, sleep ...**
  - **Veoma slabu garanciju kontrole i redosleda aktivnosti**
  - **Svaki Thread je član tzv. ThreadGroup-e koja se koristi za kontrolu pristupa i čuvanje podataka**
  - **Kod koji se izvršava u Thread-u je definisan metodom run()**
- **Synchronized metodi i blokovi koda kontrolišu atomičnost korišćenjem katanaca (locks)**
  - **Java postavlja automatske katance na tipove: byte, char, short, int, float i Object reference, ali ne i na double i long tip**
  - **volatile ključna reč kontroliše atomičnost jedne promenljive**
  - **Monitor metodi u klasi Object: wait(), notify(), notifyAll()**

Klasa Thread

- Konstruktor
  - Thread(Runnable r)
  - Pokretanjem niti počinje da se izvršava sadržaj metoda run()
- Ostali nasleđeni metodi
  - start() - aktivira run()
  - isAlive() - vraća true ako je nit aktivirana ali ne i zaustavljena
  - join() - čeka na završetak
  - interrupt() - izlazi iz wait, sleep ili join metoda
  - isInterrupted() - vraća informaciju da li je nit prekinuta
  - getPriority() - vraća prioritet niti
  - setPriority(int) - podešava prioritet niti

Preostali statički metodi

- currentThread() - vraća referencu na nit koja je lokalna sa pozicije poziva
- sleep(ms) - susenduje nit na najmanje ms milisekundi (takođe se odnosi na aktuelnu nit)
- interrupted() - vraća i čisti status o prekinutosti

Dizajn konkurentnih objekata

- Obrasci za reprezentaciju i menadžment sigurnih stanja objekata:
  - Nepromenjivost (eng. Immutability)
    - Ne dopuštati promene
  - Zaključavanje (eng. Locking)
    - Garantovanje ekskluzivnog pristupa
  - Zavisnost stanja
    - Šta raditi kad ne može da se uradi ništa
  - Ostalo...

Nepromenjivost

- Nepromenjivi objekti nikad ne menjaju stanje
- Aktivnosti nad nepromenjivim objektima su uvek sigurne i efikasne

```
class ImmutableAdder {
        private final int offset_; // blank final
        ImmutableAdder(int x) { offset_ = x; }
        int add(int i) { return i + offset_; }
}
```

- Primeri nepromenjivih tipova u javi: String, Integer, Color

Primene nepromenjivosti

- **Nepromenjive reference ka promenjivim objektima**

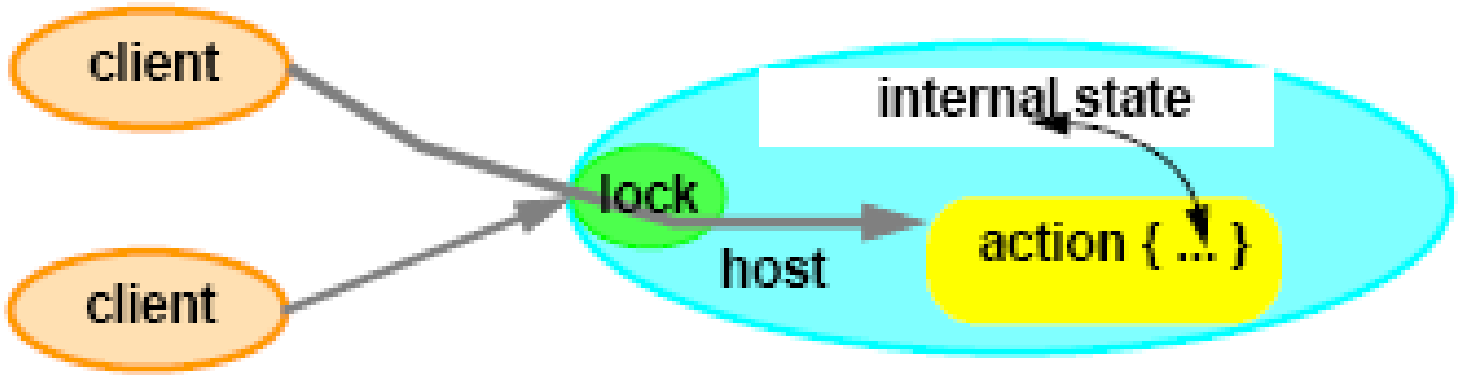
```
class Relay {
        private final Server delegate;
        Relay(Server s) { delegate = s; }
        void serve() { delegate.serve(); }
}
```

- **Parcijalna nepromenjivost**

```
class FixedList { // cells with fixed successors
        private final FixedList next; // immutable
        FixedList(FixedList nxt) { next = nxt; }
        FixedList successor() { return next; }
        private Object elem = null; // mutable

        synchronized Object get() { return elem; }
        synchronized void set(Object x) { elem = x; }
}
```

Zaključavanje (eng. Locking)



Zaključavanje

- Sprečava konflikte sa memorijskim lokacijama, tzv. Problem sihronizacije resursa
- Može se koristiti za garantovanje atomičnosti (princip sve ili ništa) metoda
- Može da izazove deadlock (mrtvo zaključavanje)

Primer

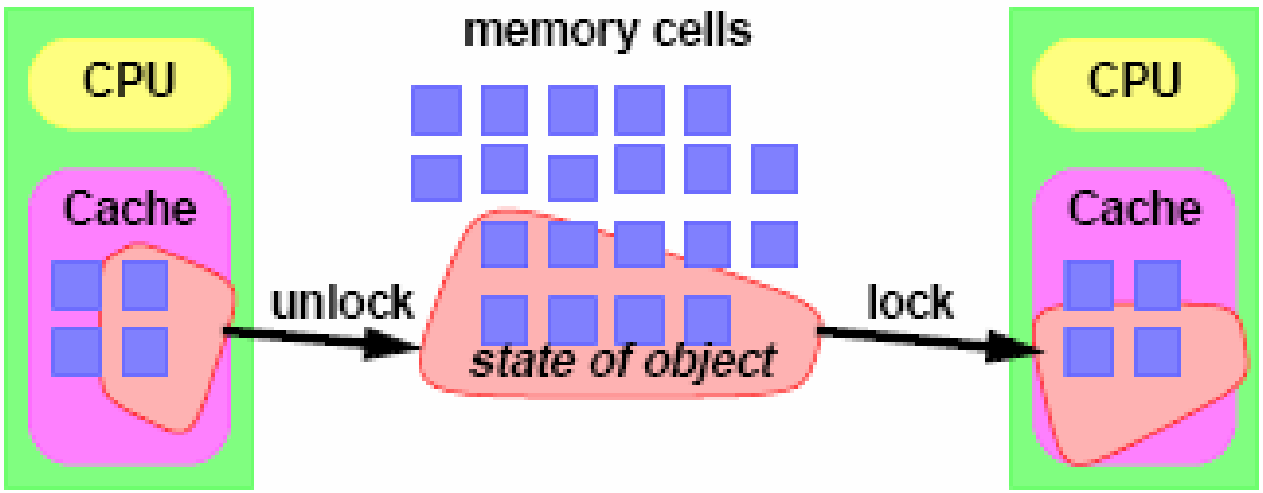
```
class Location {
  private double x_, y_;
  Location(double x, double y) { x_ = x; y_ = y; }
  synchronized double x() { return x_; }
  double y() {
        synchronized (this) {
        return y_;
        }
  }
  synchronized void moveBy(double dx, double dy) {
        x_ += dx;
        y_ += dy;
  }
}
```

Java katanci

- Svaki Java Object poseduje jedan katanac (tzv. Implicitni katanac)
  - Njime se manipuliše preko synchronized ključne reči
  - Class objekti sadrže katanac za zaštitu statičkih promenljivih i metoda
  - Skalari kao što su int, short nisu objekti, tako da njih ne možemo zaključavati
- Java katanci su više puta iskoristivi (reentrant)

Katanci i keširanje

- Zaključavanje generiše poruke između niti i memorije
- Uzimanje katanca inicira čitanje iz memorije u keš memoriju niti
- Otpuštanje katanca inicira pisanje keširanog sadržaja nazad u memoriju

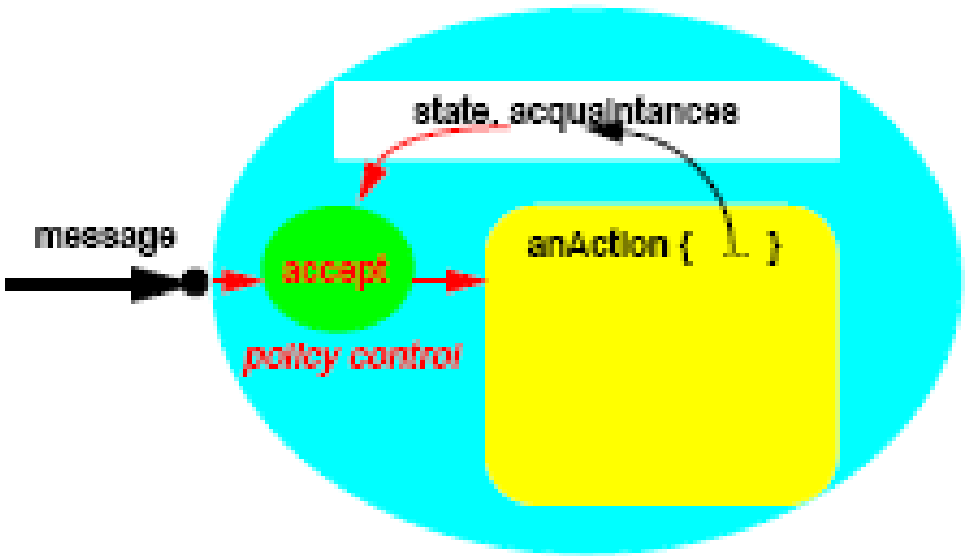


Primer sa bankom

- U priloženom materijalu....

Zavisnost stanja

- Dva aspekta kontrole
  - Poruka od strane klijenta
  - Unutrašnje stanje servera (host-a)
- Koraci u dizajnu
  - Odabrati odgovarajuće polise za rad sa akcijama
  - Definisati interfejse i protokole za te polise



Primeri operacija koje su zavisne od stanja

- Operacije na kolekcijama, tokovima i bazama podataka
  - Uklanjanje elementa sa praznog reda
- Operacije na objektima koje sadrže ograničeni skup mogućih vrednosti
  - Povlačenje novca sa praznog bankovnog računa
- Operacija koje zahtevaju deljene resurse
  - Štampanje dokumenta (zahteva štampač)
- Operacije koje zahtevaju određeni redosled izvršavanja
  - Čitanje neotvorenog fajla je npr. Loš redosled
- Operacije na spoljnim kontrolerima
  - Prebaciti u zadnju brzinu dok se auto kreće unapred

Polise za rad sa ovakvim operacijama

- Slepe akcije (blind actions) – nastavi u svakom slučaju, ne garantuje uspeh
- Neaktivnost (inaction) – ignoriše zahtev ako nije u dozvoljenom stanju
- Opiranje (balking) – izbacuje izuzetak ako je zahtev stigao dok je objekat u nedozvoljenom stanju
- Kondicionalna zaštita (guarding) – suspenduje nit dok objekat ne bude u dozvoljenom stanju
- Pokušavanje (trying) - pokušava dok ne uspe
- Intervalno čekanje (timing out) – čeka neko vreme, onda obustavlja
- Planiranje (planning) – iniciranje svih potrebnih aktivnosti koje dovode u pravo stanje

Primer opiranja (balking)

```
class Failure extends Exception { }
  class BalkingCounter {
  protected long count_ = MIN;
  synchronized long value() { return count_;}
  synchronized void inc() throws Failure {
          if (count_ >= MAX) throw new Failure();
          ++count_;
  }
  synchronized void dec() throws Failure {
          if (count_ <= MIN) throw new Failure();
          --count_;
  }
}
```

Kondicionalna zaštita (guarding)

- Generalizacija zaključavanja za aktivnosti koje su zavisne od stanja objekta
  - Zaključavanje (locking) je značilo čekaj dok nije spreman (tj. Dok neka druga nit radi nad objektom)
  - Kondicionalna zaštita znači da se čeka ne samo dok ne bude spreman, već i dok ne bude ispunjen proizvoljni predikat

- Za razliku od zaključavanja ovde se pazi i na efikasnost, a ne samo na sigurnost

Implementacija 1 - zauzeto čekanje (busy wait)

```
class UnsafeSpinningBoundedCounter { // don't use
        protected volatile long count_ = MIN;
        long value() { return count_; }
        void inc() {
                while (count_ >= MAX); // spin
                ++count_;
        }
        void dec() {
                while (count_ <= MIN); // spin
                --count_;
        }
}
```

- Problemi:
  - Nesigurno – nema zaštite od read/write konflikata
  - Neefikasno – troši procesorsko vreme na praznu petlju

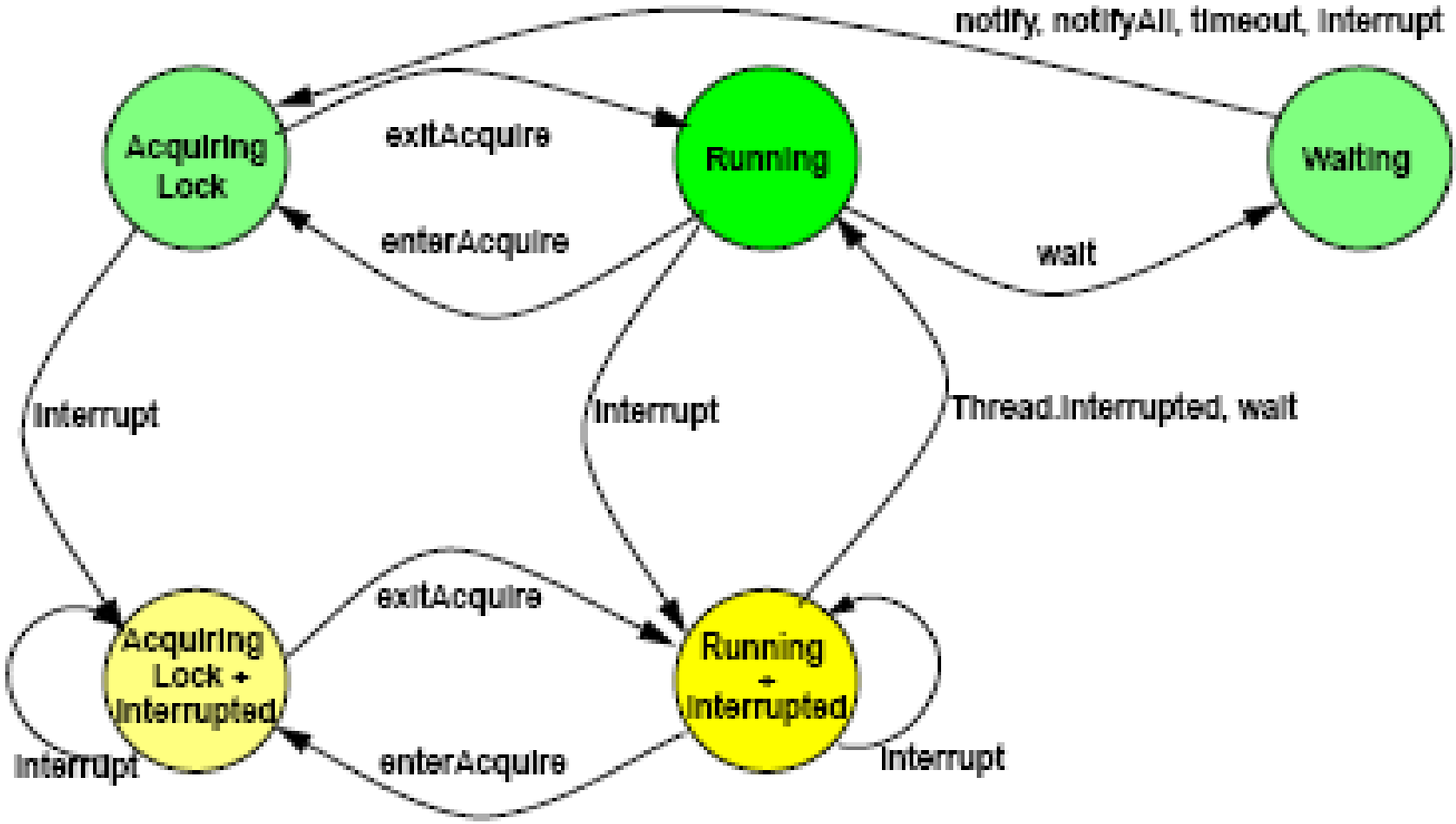
Implementacija 2 – suspendovano čekanje (suspension)

```
class GuardedBoundedCounter {
        protected long count_ = MIN;
        synchronized long value() { return count_; }
        synchronized void inc() throws InterruptedException {
                while (count_ >= MAX) wait();
                ++count_;
                notifyAll();
        }
        synchronized void dec() throws InterruptedException {
                while (count_ <= MIN) wait();
                --count_;
                notifyAll();
        }
}
```

Java monitori

- Svaki Java objekat ima svoj red (skup) čekanja (wait set)
- wait()
  - Suspenduje nit
  - Nit se smešta u red čekanja za ciljani objekat
  - Otpušta se katanac implicitno i tako daje šansa drugima
- notify()
  - Ako postoji, nit T se bira iz reda čekanja
  - T zaključava katanac na ciljanom objektu
  - T nastavlja izvršavanje naredbe koje je posle wait()
- Ostalo: notifyAll(), wait(ms)

Interakcije sa komandom prekida



SLEDI: problem proizvođač potrošač, pravičnost u Javi, izgladnjivanje...