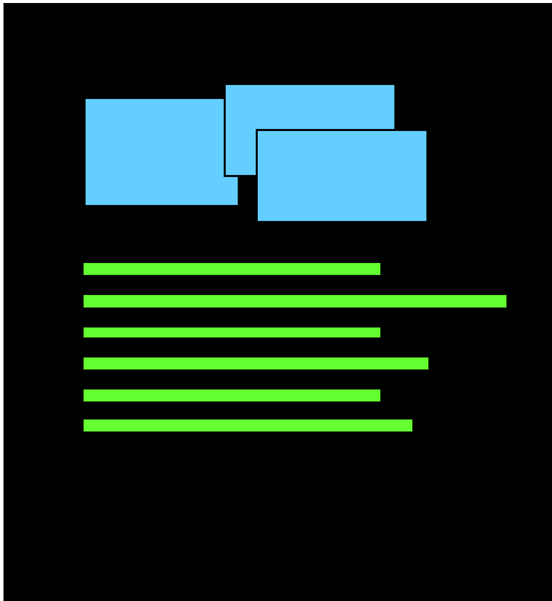
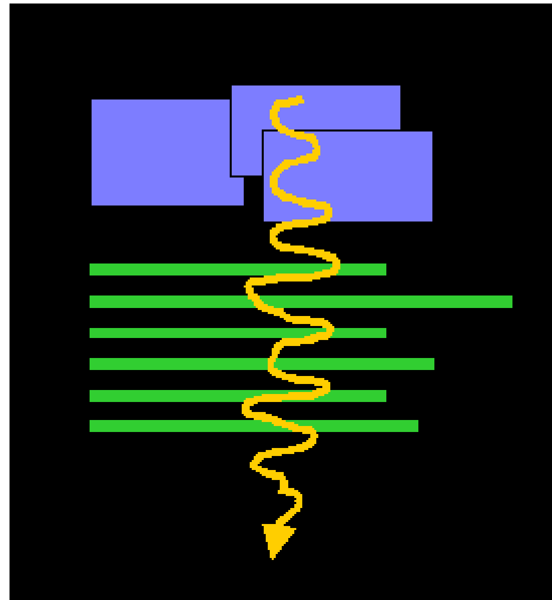
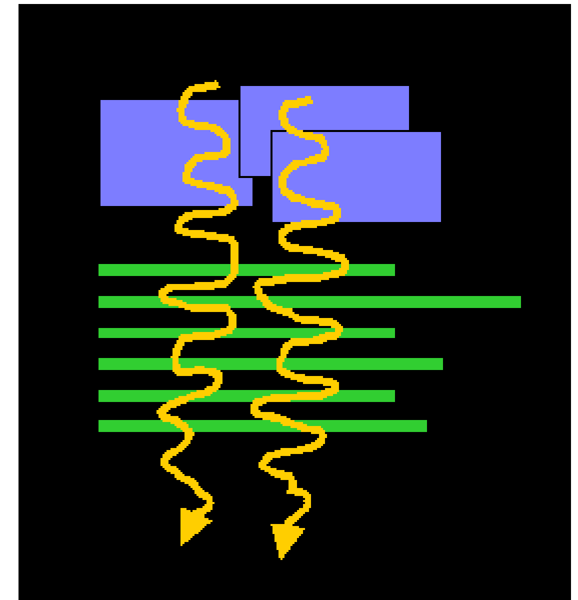
Niti

konkurenčno programiranje

Koda, proces, niti



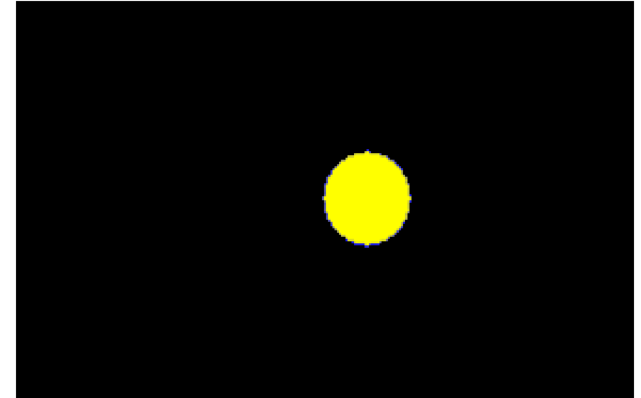




Koda programa **proces** **ve niti**

Za začetek preprosta animacija

```
import java.awt.*;
import javax.swing.*;
public class BouncingBall1 extends JFrame implements Runnable{
  Thread nit=null;
  static int w, h, x, y, dx, dy, radius = 20;
  static BouncingBall1 a;
  //************************************************
   public static void main(String[] args) {
     a = new BouncingBall1();
     a.setBounds(200,200,400,300);     a.setVisible(true);
      dx = 1; dy = 1;
      w = a.getSize().width;      h = a.getSize().height;
      x = (int)(w*Math.random());    y =(int)( h*Math.random());
      a.start();
    }
  //**************************************************
  public void paint(Graphics g){
    g.setColor(Color.black); g.fillRect(0,0,w,h);
    g.setColor(Color.yellow); g.fillOval(x-radius,y-radius, 2*radius, 2*radius);
  }
  //**************************************************
  public void start(){
       nit =new Thread(this);    nit.start();
  }
  //**************************************************
  public void run(){
    while(true){
      try {Thread.sleep(10);}catch(InterruptedException e){}
      // adjust the position of the ball
      if (x < radius || x > w - radius)  dx  =  -dx;
      if (y < radius || y > h - radius)  dy  =  -dy;
      x += dx;  y += dy;
      repaint();
 } } }
```



Animacija žogice je “hello world” ra unalniške animacije

Java

Java

Kaj je nit

- Nit (*thread*) je zaporedje izvršljivih stavkov v programu.
- Java Aplikacija: začne v metodi main() in izvaja zaporedje stavkov.
- Javanski virtualni stroj (JVM) podpira večnitnost -- teče lahko več niti sočasno.
- Nit *Garbage collector* -- nit v JVM za zbiranje pomnilnika o opuščenih objektih.

Glavna nit →

Nit z animacijo →

```
public static void main(String[ ] args) {
      a = new BouncingBall2();
      a.setBounds(200,200,400,300);
      a.setVisible(true);
       dx = 1; dy = 1;
       w = a.getSize().width;
       h = a.getSize().height;
       x = (int)(w*Math.random()); y =(int)( h*Math.random());
       a.start();  // sprozimo nit z animacijo zogice
       // pokazimo, da prva nit se vedno tece
       while (true) System.out.println(" Se vedno tecem");
    }
```

Kako tvorimo nit

Dva načina

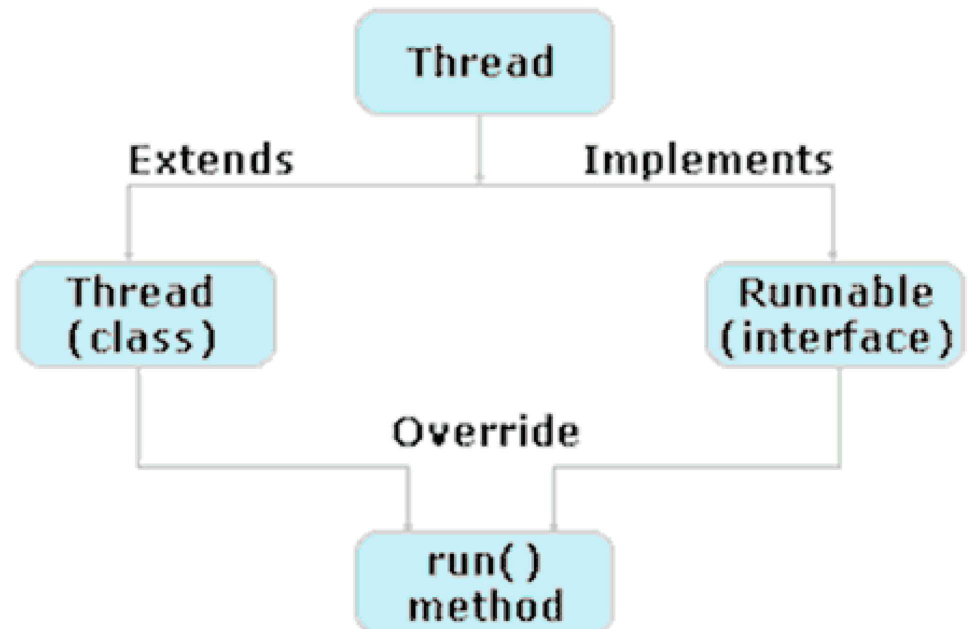
1. Implementiramo vmesnik Runnable
   Implementiramo metodo run ()

   Runnable zahteva metodo run(), a je sam še ne naredi

2. Podedujemo razred Thread
   Prekrijemo metodo run ()

   Thread že ima sama metodo run(), a jo "povozimo" s svojo

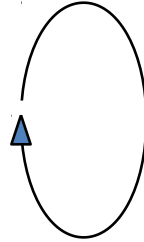
V metodo run( ) vnesemo "obnašanje", ki ga želimo



Primer več niti sočasno

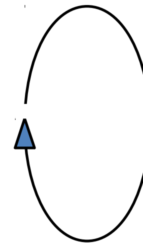
Glavna nit izpisuje: AAAAAAAA…

```
for (int j = 0;j<20;j++){
        System.out.print("A");
        try {Thread.sleep(10);}catch(Exception e){}
    }
```



Druga nit nit izpisuje: nek znak, na pr BBBBBBBBBBBBB…

```
public void run() {
      for (int k=0; k < 20; k++){
            System.out.print(ch);
            try {Thread.sleep(10);}catch(Exception e){}
      }
  }
```



Spanje niti

Metoda sleep() povzroči, da nit za delček časa zaspi.

Zaspi za interval do 1000 milisekund

```
public void run() {
    for (int k=0; k < 10; k++) {
        try {
            Thread.sleep((long)(Math.random() * 1000));
        } catch (InterruptedException e) {}
        System.out.print(num);
    } // for
} // run()
```

Niti te ejo v poljubnem zaporedju

ABBBAAABBBCCABCABCAABBBBCCAABCABBBC

Implementirajmo vmesnik *Runnable*

```
public class PisaloZnakov implements Runnable {
    char ch;

    public PisaloZnakov(char c) {
        num = n;
    }

    public void run() {
        for (int k=0; k < 10; k++)
            System.out.print(ch);
    } // run()

} // PisaloZnakov
```

Runnable zahteva implementacijo metode run().

```
Thread pisiZnak;
pisiZnak= new Thread(new PisaloZnakov('B'));
pisiZnak.start();
```

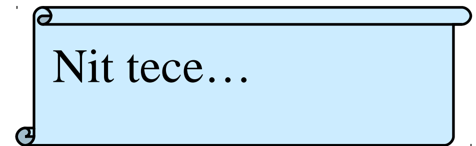
Tvorba objekta z vmesnikom Runnable

Zagon niti

DEMO

Podedujmo razred Thread: primer 1

```
public class RunThread{

  public static void main(String args[ ]) {
     NekajPisi m1=new NekajPisi("Nit tece...");
    m1.start();
   }
}

//************************************
class NekajPisi extends Thread{
  String s=null;
  NekajPisi (String s1) {
    s=s1;
  }

  public void run(){
      System.out.println(s);
  }
}
```



DEMO

Primer s tremi sočasnimi števci

```
public class StevecTest {
    public static void main(String[ ] args) {
        Stevec s1 = new Stevec(5);
        Stevec s2 = new Stevec(5);
        Stevec s3 = new Stevec(5);
        s1.start();
        s2.start();
        s3.start();
    }
}
```

Tvorba treh niti

Niti poženemo

```
Stevec 0: 0
Stevec 1: 0
Stevec 2: 0
Stevec 1: 1
Stevec 2: 1
Stevec 1: 2
Stevec 0: 1
Stevec 0: 2
Stevec 1: 3
Stevec 2: 2
Stevec 0: 3
Stevec 1: 4
Stevec 0: 4
Stevec 2: 3
Stevec 2: 4
```

Izpis programa

Razred števec (dedujemo Thread)

```
class Stevec extends Thread {
   private static int vsehStevcev = 0;
   private int idStevca, maxCount;

 public Stevec(int maxCount) {
       this.maxCount = maxCount;
       idStevca = vsehStevcev++;
   }

/** Nit naredi “maxCount” izpisov, nato se izteče. */
   public void run() {
       for(int stevilo=0; stevilo<maxCount; stevilo++) {
           System.out.printf (“Stevec %s: %s%n", idStevca, stevilo);
            try {
                 Thread.sleep ( (int) Math.random()* 1000); }
           catch(InterruptedException ie) { }
       }
   }
}
```

DEMO

Razred Števec (implementira Runnable)

```
class Stevec2 implements Runnable {
   private int maxCount;
   private int idStevca;
   static private int vsehStevcev = 0;

   public Stevec2(int max){
      idStevca = ++vsehStevcev;
      maxCount=max;
   }
   public void run(){
     int stevilo =0;
     while(stevilo++< maxCount) System.out.printf("Stevec %s: %s%n", idStevca, stevilo);
   }

   public static void main(String[ ] args){
      Runnable r1= new Stevec2(5);
      Thread t1 = new Thread(r1);   t1.start();
      Runnable r2= new Stevec2(5);
      Thread t2 = new Thread(r2);   t2.start();
   }
}
```

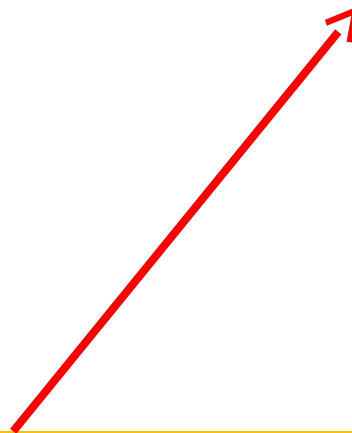
Instanca Thread ovije instanco implementacije Runnable

DEMO

Izpis pri preskusu števca

Stevec 0: 1
Stevec 0: 2
Stevec 0: 3
Stevec 0: 4
Stevec 0: 5
Stevec 1: 1
Stevec 1: 2
Stevec 1: 3
Stevec 1: 4
Stevec 1: 5

Stevec 0: 0
Stevec 1: 0
Stevec 2: 0
Stevec 2: 1
Stevec 1: 1
Stevec 2: 2
Stevec 0: 1
Stevec 2: 3
Stevec 1: 2
Stevec 2: 4
Stevec 0: 2
Stevec 1: 3
Stevec 0: 3
Stevec 1: 4
Stevec 0: 4



Pri dovolj velikem številu iteracij in ve jem številu niti bi bil izpis pomešan

Primer proizvajalec - porabnik

Proizvajalec (producer) generira cela števila med 0 in 9 ter jih shranjuje v "škatlo" .

Porabnik (consumer) jemlje števila iz iste "škatle".

Da bi bila sinhronizacija bolj zanimiva, proizvajalec med vnosi zaspi v naključnih intervalih.

Problem: tekmovalne razmere (race conditions)

Tako proizvajalec kot porabnik (sočasno?) uporabljata isti objekt

Toda kaj, če je proizvajalec prehiter?

In kaj, če je porabnik prehiter?

Koda “škatle” (to bo skupni objekt)

```
public class Skatla {
    private int vsebina;                      // kaj je v skatli?
    private boolean polna = false;  // ali je skatla polna?

    public  int vzemi (String kdo) {
        while (polna == false) ; // cakamo, da bo kaj v skatli
        polna = false;
        System.out.format("Porabnik %s vzel %d%n", kdo, vsebina);
        return vsebina;
    }

    public void vstavi(String kdo, int vrednost) {
        while (polna == true);  // cakamo, da bo skatla spet prazna
        vsebina = vrednost;  // damo vrednost v skatlo
        polna = true;
        System.out.format("Proizvajalec %s vstavil  %d%n", kdo, vsebina);
    }
}
```

Koda proizvajalca

```
public class Proizvajalec extends Thread {
    private Skatla skatla;
    private String ime;

    public Proizvajalec (Skatla c, String ime1) {
        skatla = c;
        ime = ime1;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            skatla.vstavi(ime, i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Koda porabnika

```
public class Porabnik extends Thread {
    private Skatla skatla;
    private String ime;

    public Porabnik (Skatla c, String ime1) {
        skatla = c;
        ime = ime1;
    }

    public void run() {
        int vrednost = 0;
        for (int i = 0; i < 10; i++) {
            vrednost = skatla.vzemi(ime);
        }
    }
}
```

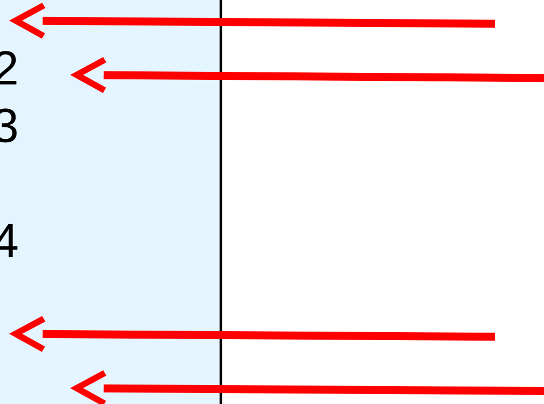
Aplikacija proizvajalec - porabnik

```
class ProizvajalecPorabnik  {
    public static void main(String[] args) {
        Skatla s = new Skatla();  // naredimo skupno skatlo

        System.out.println("Tvorba in zagon proizvajalca!");
        Proizvajalec janez = new Proizvajalec (s, "janez");
        janez.start();

        System.out.println("Tvorba in zagon porabnika!");
        Porabnik polde = new Porabnik (s, "polde");
        polde.start();

    }
}
```

DEMO

Izpis je “zmešan”

Tvorba in zagon proizvajalca!
Tvorba in zagon porabnika!
Proizvajalec janez vstavil 0
Porabnik polde vzel 0
Proizvajalec janez vstavil 1
Porabnik polde vzel 1
Porabnik polde vzel 2
Proizvajalec janez vstavil 2
Proizvajalec janez vstavil 3
Porabnik polde vzel 3
Proizvajalec janez vstavil 4
Porabnik polde vzel 4
Porabnik polde vzel 5



Obe niti tekmujeta za isti objekt in nista sinhronizirani

Splošna rešitev tekmovanja za vire

Sinhroniziramo **del** kode

```
synchronized(nekObjekt) {
    koda
}
```

Poenostavljena razlaga:
Ko nit vstopi v sinhronizirani del kode, ne more vanjo vstopiti nobena druga nit, dokler prva te kode ne zapusti.

Sinhronizacija cele metode

```
public synchronized void nekaMetoda() {
    telo
}
```

To je ekvivalentno temu:

```
public void nekaMetoda() {
    synchronized (this) {
        telo
    }
}
```

Ko je neka nit v sinhronizirani metodi, morajo vse ostale niti, ki poskušajo klicati to metodo, počakati

Popravljen primer proizvajalec porabnik

```
public class Skatla {
    private int vsebina;
    private boolean polna = false;
    public synchronized int vzemi(String kdo) {
        while (polna == false) {
            try {
                wait();  // cakamo, da bo kaj v skatli
            } catch (InterruptedException e) { }
        }
        polna = false;
        System.out.format("Porabnik %s vzel %d%n", kdo, vsebina);
        notifyAll();
        return vsebina;
    }
    public synchronized void vstavi(String kdo, int vrednost) {
        while (polna == true) {
            try {
                wait();
            } catch (InterruptedException e) { }
       }
        vsebina = vrednost;  // damo vrednost v skatlo
        polna = true;
        System.out.format("Proizvajalec %s vstavil  %d%n", kdo, vsebina);
        notifyAll();
    }
}
```

*Razred Object nudi metode **wait, notify, notifyAll**, s katerimi lahko niti čakajo na nek pogoj in obvestijo druge metode, kdaj ta pogoj nastopi.*

DEMO

Primer: sodelujoče niti

- Niti, ki sodelujejo, potrebujejo eksplicitno sinhronizacijo in koordinacijo.
- Problem: Simulacija trgovine, kjer imamo prodajalca in vrsto čakajočih kupcev. Uporabimo napravo s številkami za dodeljevanje vrstnega reda strežbe.
- Katere razrede imamo:
  - Trgovina: glavni program, sproži niti.
  - Stevec: pomni, kdo je naslednji.
  - Prodajalec: streže naslednjega kupca
  - Kupec: čaka v vrsti.

Razred *Stevec* (verzija 1)

Souporabljen vir: metoda ***naslednjaStevilka****, jo souporablja več niti.*

```
class Stevec {
    private int next = 0;      // Next place in line
    private int serving = 0;  //koga serviram

    public synchronized int naslednjaStevilka() {
        next = next + 1;
        return next;
    }

    public int naslednjiKupec() {
        ++serving;
        return serving;
    }
}
```

Sinhronizirane metode ne moremo predkupiti.

Uporabljajo kupci.

Uporablja prodajalec.

Opomba: Predkup = za asna prekinitev

Razred Kupec

Kupci vzamejo listek z naslednjo številko

Spremenljivka razreda

edinstven ID.

Ta kupec bo morda moral čakati.

```
public class Kupec extends Thread {
    private static int number = 10000;     // ID oznaka prvega kupca
    private int id;
    private Stevec stevilcnik;
    public Kupec( Stevec st) {
        id = ++number;
        stevilcnik = st;
    }
    public void run() {
        try {
            sleep( (int)(Math.random() * 1000 ) );
            System.out.println("Kupec " + id + " vzame listek" +
                stevilcnik.naslednjaStevilka());
        }
        catch (InterruptedException e) { }
    }
}
```

Razred Prodajalec

Prodajalec ponavlja strežbo z naslednjim kupcem

```
public class Prodajalec extends Thread {
    private Stevec stevilcnik;
    public Prodajalec(Stevec s) {
      stevilcnik = s;
    }
    public void run() {
        while (true) {
            try {
                sleep( (int)(Math.random() * 50));
                System.out.println("prodajalec streze listek" +
                                stevilcnik.naslednjiKupec());
            } catch (InterruptedException e) { }
        }
    }
}
```

Neskončna zanka

Streži naslednjemu kupcu.

Razred Trgovina

Trgovina sproži niti prodajalec in kupec in jim posreduje referenco na Stevec.

```
public class Trgovina {
    public static void main(String args[ ]) {
        System.out.println( “Prozenje niti prodajalca in kupcev" );
        Stevec stevilcnik = new Stevec();
        Prodajalec prodajalec = new Prodajalec(stevilcnik);
        prodajalec.start();
        for (int k = 0; k < 5; k++) {
            Kupec kupec = new Kupec(stevilcnik);
            kupec.start();
        }
    }
}
```

1 prodajalec

5 kupcev

Demo

Problem: neobstoječi kupci

Problem, prodajalec ne čaka na kupce

```
Prozenje niti prodajalca in kupcev"
Prodajalec streze listek 1
 Prodajalec streze listek 2
 Prodajalec streze listek 3
 Prodajalec streze listek 4
 Prodajalec streze listek 5
 Kupec 10004 vzame listek 1
 Kupec 10002 vzame listek 2
 Prodajalec streze listek 6
 Kupec 10005 vzame listek 3
 Prodajalec streze listek 7
 Prodajalec streze listek 8
 Prodajalec streze listek 9
 Prodajalec streze listek 10
 Kupec 10001 vzame listek 4
 Kupec 10003 vzame listek 5
```

Nit *Prodajalec* bi moral čakati, da *Kupec* vzame številko.

To bi lahko dosegli z naslednjo metodo v razredu Stevec:

```
public synchronized boolean cakanjeKupca() {
      return (next > serving);
   }
```

Ta pogoj bi morali preverjati v razredu Prodajalec, preden bi šli na naslednjega kupca

Asinhrona narava niti

- *Niti so asinhrone – Čas njihovega izvajanja in vrstni red so nepredvidljivi.*
- *Ni mogoče napovedati, kdaj bo neka nit začasno prekinjena (preempted).*

En stavek Jave

```
int N = 5 + 3;
```

Več strojnih instrukcij

(1) Fetch 5 from memory and store it in register A.
(2) Add 3 to register A.
(3) Assign the value in register A to N.

Ko lahko nastopi prekinitev.

Konkurenčnost niti – kritične sekcije

Če pride do začasne prekinitve (predkupa) potem, ko je kupec vzel številko in preden pride do izpisa njene številke, bi lahko dobili kaj takega:

```
Prozenje niti prodajalca in kupcev"
Prodajalec streze listek 1
 Prodajalec streze listek 2
 Prodajalec streze listek 3
 Kupec 10004 vzame listek 4
 Prodajalec streze listek 4
 Prodajalec streze listek 5
 Kupec 10001 vzame listek 1
 Kupec 10002 vzame listek 2
 Kupec 10003 vzame listek 3
 Kupec 10005 vzame listek 5
```

Logično naša koda zagotavlja, da prodajalec ne more postreči, dokler ne pride do prevzema listka, vendar ...

… ta izpis ne odraža pravega stanja simulacije.

Kritična sekcija je segment kode niti, ki ga ne smemo predkupiti (začasno prekiniti)

Predelajmo razred Stevec

Izpisi naj bodo v razredu Stevec (znotraj kritičnih sekcij)

```
public class Stevec {
    private int next = 0;     // nasled. mesto v vrsti
    private int serving = 0;  // naslednji postrezen kupec

    public synchronized int naslednjaStevilka( int custId) {
        next = next + 1;
        System.out.println( "Kupec " + custId + " vzame listek " + next );
        return next;
    }

    public synchronized int naslednjiKupec() {
        ++serving;
        System.out.println("  Prodajalec streze listek " + serving );
        return serving;
    }

    public synchronized boolean cakanjeKupca () {
        return (next > serving);
    }
}
```

Kritične sekcije: sinhronizirane enote ne morejo biti predkupljene.

Predelati moramo tudi razreda Kupec in Prodajalec

Prodajalec in kupci ne povzročajo izpisov

```
public void run() { // Kupec.run()
    try {
        sleep((int)(Math.random() * 2000));
        stevilcnik.naslednjaStevilka(id);
    } catch (InterruptedException e) {}
} // run()
```

Samo vzemi številko.

```
public void run() {    // Prodajalec.run()
    for (int k = 0; k < 10; k++) {
        try {
            sleep( (int)(Math.random() * 1000));
            if (stevilcnik.cakanjeKupca())
               stevilcnik.naslednjiKupec();
        } catch (InterruptedException e) { }
    } // for
} // run()
```

Samo postreži kupca.

Demo

Koordinirane niti: pravilni izpis

Prozenje niti prodajalca in kupcev" Kupec 10001 vzame listek 1
Prodajalec streze listek 1
Kupec 10003 vzame listek 2
Kupec 10002 vzame listek 3
Prodajalec streze listek 2
Kupec 10005 vzame listek 4
Kupec 10004 vzame listek 5
Prodajalec streze listek 3
Prodajalec streze listek 4
Prodajalec streze listek 5

Kupci so postreženi v pravilnem zaporedju ne glede na to, kako prihajajo.

Učinkovito programiranje: Uporaba kritičnih sekcij za zagotovitev medsebojnega izločanja in koordinacijo niti

Koordinacija niti z *wait/notify*

- Metoda wait() postavi nit v stanje čakanja, metoda notify() pa prestavi nit iz tega stanja v stanje ready.
- Alternativa programiranja: prodajalec čaka (*wait*), da ga bo kupec opozoril (*notify*).
- Za to pa moramo razreda *Stevec* in *Prodajalec* predelati.
- *Model "Producer/Consumer"* : dve niti souporabljata nek vir, ena ga proizvaja, druga ga porablja.

Predelan razred Stevec

```
public synchronized int naslednjiKupec() {
    try {
        while (next <= serving)  {
            System.out.println("  Prodajalec caka");
            wait();
        }
    } catch(InterruptedException e) {}
    finally {
        ++serving;
        System.out.println("  prodajalec streze listku " + serving );
        return serving;
    }
}
```

Ko prodajalec kliče to metodo, mora čakati (wait) na kupca .

```
public synchronized int naslednjaStevilka (int custId) {
    next = next + 1;
    System.out.println( “Kupec " + custId +  " vzame listek " + next );
    notify();
    return next;
}
```

Ko kupec kliče to metodo, opozori (notify) prodajalca, da je prišel.

Predelan razred Prodajalec

Metoda Prodajalec.run() je sedaj poenostavljena

```
public void run() {
    while (true) {
        try {
            sleep((int)(Math.random() * 1000));
            stevilcnik.naslednjiKupec();
        } catch (InterruptedException e) {}
    }
}
```

Neskončna zanka.

Novi izpis

Prodajalec bo opozorjen, ko pride novi kupec.

Prozenje niti prodajalca in kupcev"   Kupec 10004 vzame listek1
Kupec 10002 vzame listek 2
 Prodajalec streze listek 1
 Prodajalec streze listek 2
Kupec 10005 vzame listek 3
Kupec 10003 vzame listek 4
 Prodajalec streze listek 3
Kupec 10001 vzame listek 5
 Prodajalec streze listek 4
 Prodajalec streze listek 5
Prodajalec caka

Omejitve mehanizma wait/notify

- Obe metodi, wait() in notify(), sta metodi razreda Object. To omogoča zaklepanje objektov.
- Metodo wait() lahko uporabimo znotraj katerekoli sinhronizirane metode, ne le samo v niti (Thread).
- Obe metodi, wait() and notify(), moramo uporabljati v sinhroniziranih metodah, sicer bi lahko prišlo do izjeme IllegalMonitorStateException s sporočilom “current thread not owner.”
- Ko v sinhronizirani metodi uporabimo wait(), se dani objekt odklene in tako dopusti drugim metodam, da kličejo sinhronizirane metode objekta.

Povzetek

- Vmesnik *Runnable* spremeni obstoječi razred v nit.
- Nit z višjo prioriteto prekine (predkupi, *preempt)* nit z nižjo prioriteto.
- Implementacija niti je platformno odvisna.
- Nit z visoko prioriteto, ki nikdar ne prepusti CPE, lahko povzroči "stradanje" (*starvation*) niti z nižjo prioriteto.

Življenski cikel niti

Alive

Running

new Thread1();

while (…) { … }

New Thread

Runnable

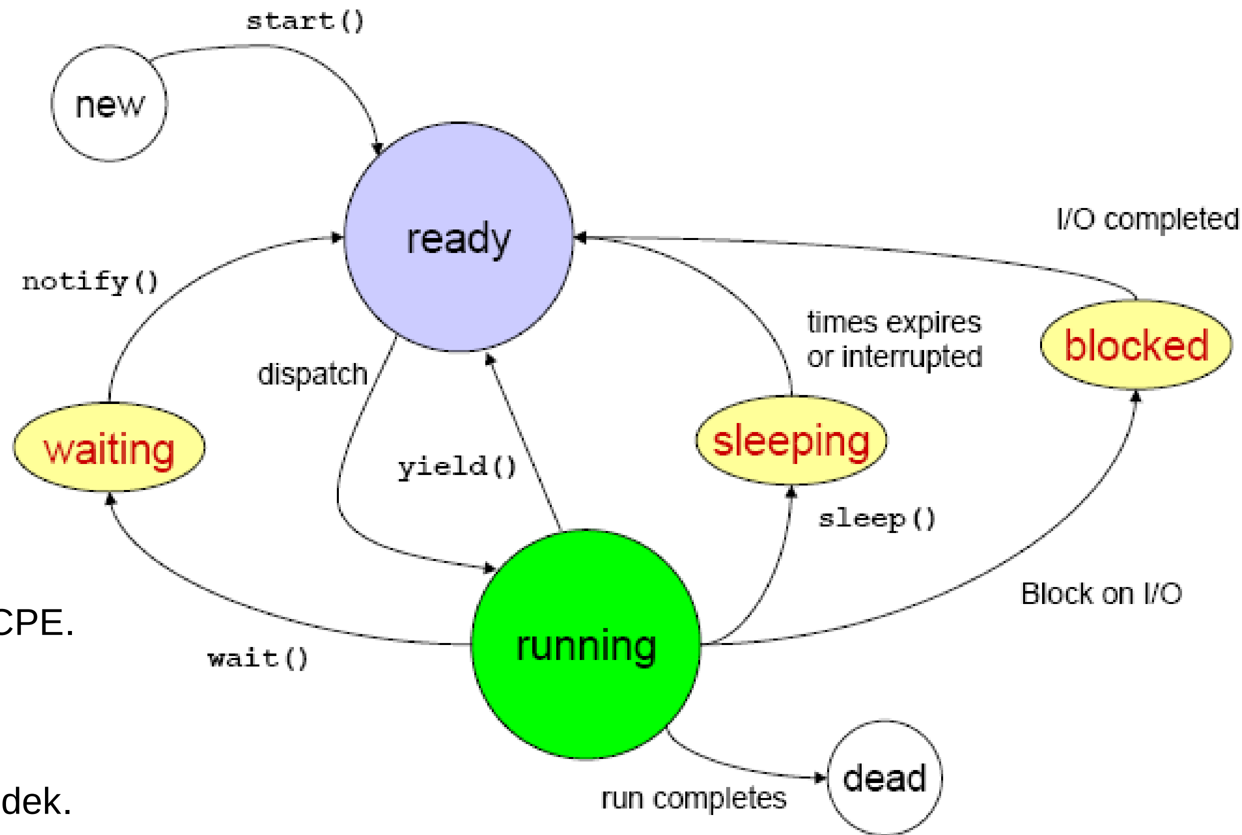
Dead Thread

Thread1.start();

Blocked

Object.wait()
Thread.sleep()
Blokirana s klicem vhoda/izhoda
aka na nek monitor

Stanja niti – bolj podrobno



Stanje	Opis
Ready	Nit je pripravljena za izvajanje in čaka na CPE.
Running	Nit teče.
Waiting	Nit čaka na nek dogodek.
Sleeping	Nit nekaj časa spi.
Blocked	Čaka na konec neke vhodno izhodne operacije.
Dead	Konec življenja niti

Iz javanske knjižnice: razred Thread

```
public class Thread extends Object implements Runnable {
  // konstruktorji
    public Thread();
    public Thread(Runnable target);
    public Thread( String name );
  // metode razreda
    public static native void sleep(long ms) throws InterruptedException;
    public static native void yield();
  // Metode instanc
    public final String getName();
    public final int getPriority();
    public void run();                    // edina metoda vmesnika Runnable
    public final void setName();
    public final void setPriority();
    public synchronized native void start();
    public final void stop();      // opuščena metoda (deprecated)
}
```

Uporabne metode iz razreda Thread

- `run()`
  - Run it! (Runnable)
- **start()**
  - Activates the thread and calls `run()`.
- `stop()`
  - Forces the thread to stop.
- `suspend()`
  - Temporarily halt the thread.
- `resume()`
  - Resume a halted thread.
- `destroy()`
  - equivalent to UNIX's "kill -9"
- `isAlive()`
  - Is it running?
- `yield()`
  - Let another thread run.
- `join()`
  - Wait for the death of a thread.
- `sleep(long)`
  - Sleep for a number of milliseconds.
- `interrupt()`
  - Interrupt sleep,wake up!

Metode za komunikacijo med nitmi

- Inter-thread communication methods are declared in `java.lang.Object`.
- Each object could be associated with a monitor (a sort of thread lock).

- **`wait()`**
  - Suspend the thread.
  - Wait can also be time limited.
- **`notify()`**
  - Unlock the first monitored thread.
  - (The first that called wait() within the monitor.)
- **`notifyAll()`**
  - Unlocks all monitored threads.
  - Highest prioritised first!

Še en primer: pajek in muha

Problem:

Simuliramo 2 neodvisni živali, brenčečo muho in pajka, ki sanja, kako bi jo ujel.

Primer je le nakazan, dopolni ga v popolno igrico

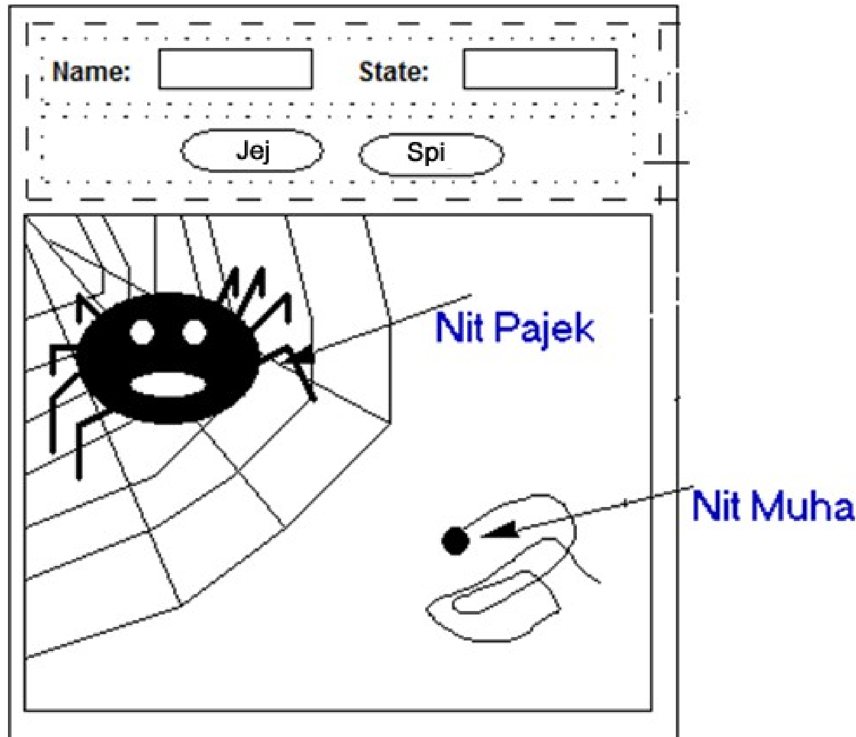
Demo

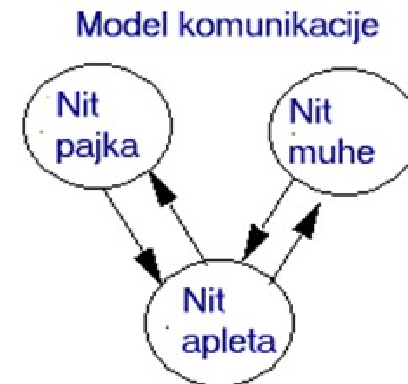
Source

Organizacija programa

3 objekti, 3 ločene niti:

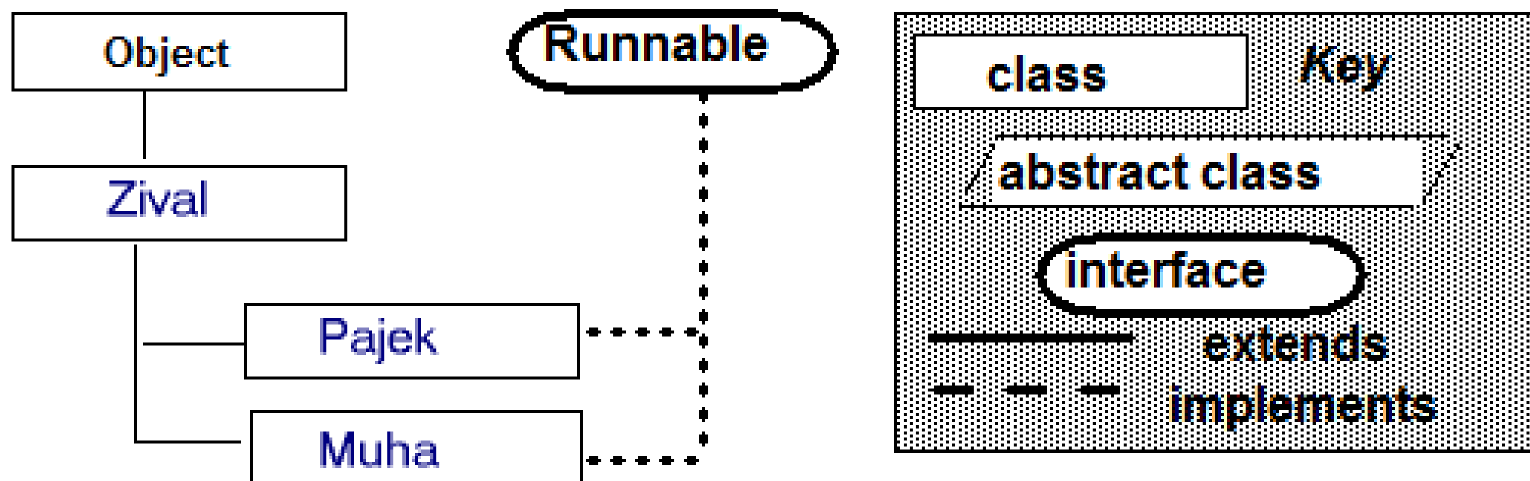
- Aplet: implementira vmesnik.
- Pajek: naš miljenček
- Muha: brenčeča muha





Hierarhija razredov

Večkratno dedovanje: Pajek in Muha dedujeta lastnosti Zival in lastnosti Thread.



public class Muha extends Zival implements Runnable { ... }

public class Pajek extends Zival implements Runnable { ... }

Razred Zival

```
public class Zival {
    protected int state;
    protected String name;
    public static final int MRTEV = -1;
    public static final int JE_HRANO = 0;
    public static final int SPI = 1;
    public static final int SANJA= 3;
    public static final int LETI = 4;     // za letece zivali

    public void dream() {
        state = SANJA;
    }
    protected void delay(int N) {            // delay for N milliseconds
        try {
            Thread.sleep(N);
        } catch (InterruptedException e) {
            System.out.println(e.toString());
        }
    }
}
```

State in name ne smeta biti privatna, če ju hočemo dedovati

Novo stanje in metoda dream

Tako pajek kot muha uporabljata metodo delay().

Razred Muha

```
import java.awt.*;
public class Muha extends Zival implements Runnable {
    private static final int XMIN = 225;   // podrocje letenja
    private static final int XMAX = 300;
    private static final int YMIN = 245;
    private static final int YMAX = 305;
    private static final int SIDE = 5;              // velikost muhe
    private static final int MAX_RANGE = 15;  // gibanje muhe
    private static final int MIN_DELTA = -10;
    private IgraPajekMuha applet;   // Referenca na vmesnik
    private Point location;              // koordinate muhe

    public Muha (IgraPajekMuha app) {
        applet = app;
        location = new Point(XMAX, YMAX); // zacetna lokacija muhe
        state = FLYING;
    }

    public void run() {
        while (state != MRTEV) {
            letenje();
            delay(125);
        }
    }
}
```

Konstante omejujejo položaj in gibanje muhe.

Pozor: referenca na aplet

Muha brenči, dokler ne umre

Metoda Muha.letenje( )

```
public synchronized void letenje() {
    state = FLYING;
    Graphics g = applet.getGraphics();
    g.setColor(Color.white);            // Erase current image
    g.fillRect(location.x, location.y, SIDE, SIDE);
                                        // Calculate new location
    int dx = (int)(MIN_DELTA +  Math.random() * MAX_RANGE);
    int dy = (int)(MIN_DELTA + Math.random() * MAX_RANGE);
    if (location.x + dx >= XMIN)  location.x = location.x + dx;
    else  location.x = XMIN;
    if (location.y + dy >= YMIN)  location.y = location.y + dy;
    else  location.y = YMIN;
    if (location.x + dx <= XMAX)  location.x = location.x + dx;
    else    location.x = XMAX;
    if (location.y + dy <= YMAX)  location.y = location.y + dy;
    else location.y = YMAX;
                                // Draw new image at new location
    g.setColor(Color.red);
    g.fillRect(location.x, location.y, SIDE, SIDE);
}
```

Brisanje

Gibanje znotraj meja

Ponovno risanje

Komunikacija med nitmi

Če naj simulirani pajek poje muho, mora poznati njen položaj

```
public Point getLocation() {  // vrni lokacijo muhe
        return location;
}
```

Muha mora umreti, ko je pojedena

```
public synchronized void umri() {

        state = MRTEV;
}
```

To bo povzročilo, da se zanka run() zaključi.

Metoda Pajek.run ( )

Pajek lahko je, razmišlja ali spi

Obnašanje pajka je avtonomno in naključno.

```
public void run() {
    while (true) {
        int choice = (int)(Math.random() * 4);
        if (choice == 0)
            autoeat();
        else if (choice == 1)
            autosleep();
        else if (choice == 2)
            think();
        else
            dream();
        delay(5000);
    } //while
} // run()
```

```
private void autoeat() {
    Graphics g = applet.getGraphics();
    state = EATING;
    applet.updateStateField();
    for (int k = 0;  k < SLEEPING_IMG; k++) {
        g.drawImage(image[k], 20, 100, applet);
        delay(200) ;
    }
} // autoeat()
```

Metoda Pajek.jej()

Pajku lahko ukažemo, da naj je, vendar ni nujno, da nas vedno uboga

```
public void jej() {
    Graphics g = applet.getGraphics();
    int choice = (int) ( Math.random() * 3 );
    if ( choice == 2 )        // i.e., 1 in 3 chance
        g.drawImage(image[NOT_HUNGRY_IMG], 20, 100, applet);
    else {
        state = JE_HRANO;
        for (int k = 0;  k < SLEEPING_IMG; k++) {
            g.drawImage(image[k], 20, 100, applet);
            delay(200) ;
        }
    } // else
} // eat()
```

Pajek v tretjini primerov ne uboga.

Metoda Pajek.sanjaj( )

Pajek sanja, da je žaba

```
public synchronized void sanjaj() {
    state = DREAMING;
    applet.updateStateField();
    Graphics g = applet.getGraphics();        // risanje slike iz sanj
    g.drawImage(image[DREAMING_IMG], 20, 100, applet);
    delay(5000);
    g.drawImage(image[ZABA], 20, 100, applet);  // spremeni se v zabo
    delay(5000);
    muhaLocation = applet.getMuhaLocation();      // glej, kje je muha
    g.setColor( Color.pink);
    g.drawLine(ZABA_X, ZABA_Y, muhaLocation.x, flyLocation.y);  // pojej muho
    g.drawLine(ZABA_X + 1, ZABA_Y + 1, muhaLocation.x + 1, muhaLocation.y + 1);
    g.drawLine(ZABA_X + 2, ZABA_Y + 2, muhaLocation.x + 1, muhaLocation.y + 1);
    g.drawLine(ZABA_X + 3, ZABA_Y + 3, muhaLocation.x + 1, muhaLocation.y + 1);
    applet.jejMuho();
    delay(250);
    g.drawImage(image[SRECNA_ZABA], 20, 100, applet);
    delay(5000);
    applet.novaMuha();  // ce zelimo gro nadaljevati z novo muho
}
```

Pajek postane žaba...

... In poje muho.

Razred IgraPajekMuha

Ta aplet naredi pajka in muho

```
private Pajek pajkec= new Pajek (“Pajkec", this);          // Tvorba pajka
private Muha muhica = new Muha(this);                       //  in muhe
```

V init() sproži njune niti

```
new Thread(pajkec).start();    // Start thread pajkec (v init())
new Thread(muhica).start();      // Start thread muhica (v init())
```

Posreduje njuno interakcijo

```
public void jejMuho() {
        muhica.umri();
}
public Point getMuhaLocation() {
   return muhica.getLocation();
}
```

To kliče pajek.

Dedovanje in polimorfizem

- Dedovanje: Oba, *Pajek* in *Muha*, podedujeta lastnosti *Zival* in oba dobita lastnosti niti (*Thread*) z implementacijo vmesnika *Runnable*.
- Skupne metode so definirane v *razredu Zival*.
- Polimorfizem: *run()* se obnaša različno v obeh podrazredih *Thread*.