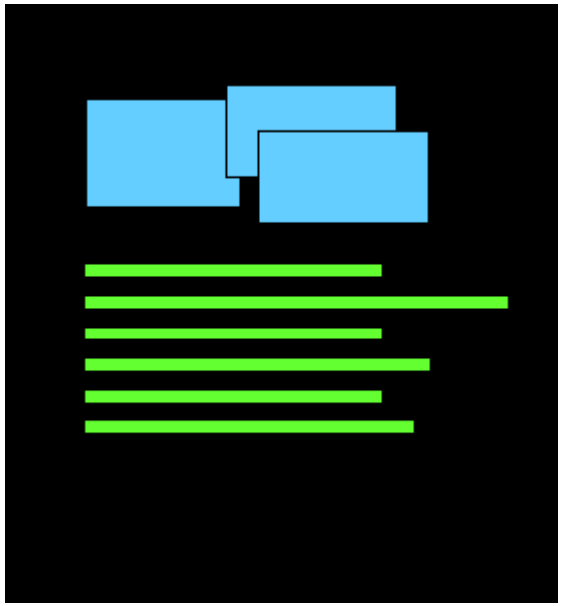
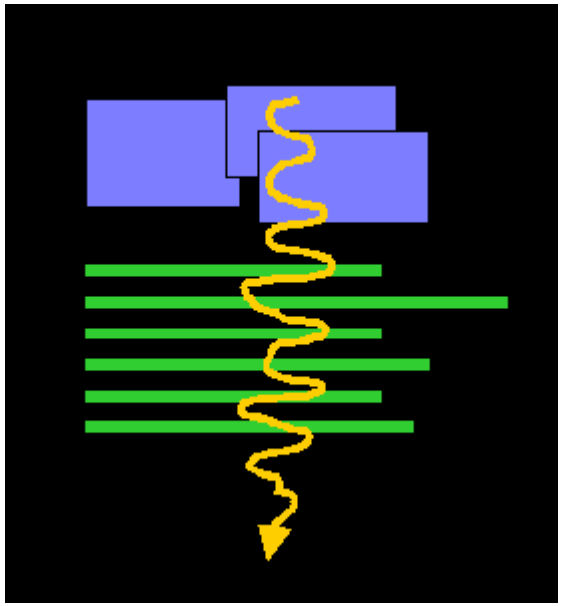
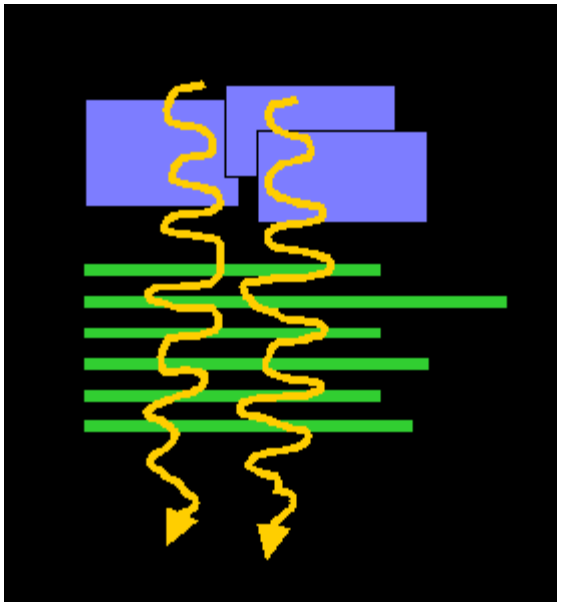
Niti

konkurenčno programiranje

Koda, proces, niti



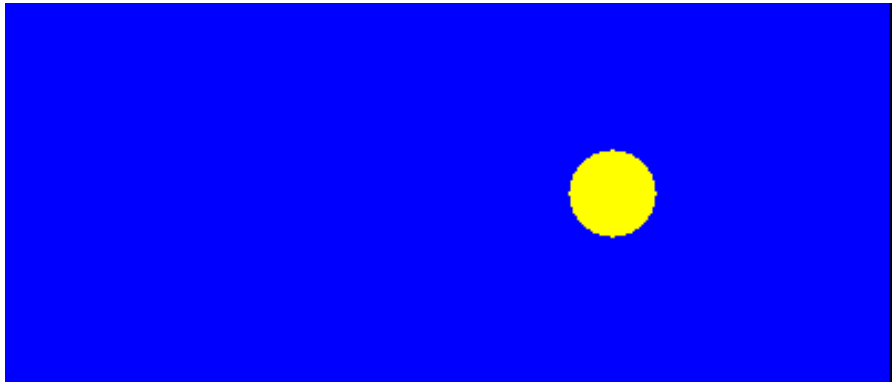




Koda programa **proces** **ve niti**

Za za etek preprosta animacija

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class Nit1 extends Applet implements Runnable {
  Thread nit=null;
  int w, h, x, y, dx, dy, radius = 20;
  //*****************************************************
  public void init(){
      x = radius; y = radius;  dx = 1; dy = 1;
      w = this.getSize().width;
      h = this.getSize().height;
  }
  //*****************************************************
  public void paint(Graphics g){
    g.setColor(Color.blue); g.fillRect(0,0,w,h);
    g.setColor(Color.yellow); g.fillOval(x-radius,y-radius, 2*radius, 2*radius);
  }
  //*****************************************************
  public void start(){
    if(nit==null){
       nit =new Thread(this); nit.start();
    }
  }
  //*****************************************************
  public void run(){
    while(nit != null){
      try {Thread.sleep(10);}catch(InterruptedException e){}
      x = x + dx; y = y + dy;
      if (x > (w - radius)) { x = w - radius; dx = -1; }
      if (y > (h - radius)) { y = h - radius; dy = -1; }
      if (x < radius)       { x = radius; dx = 1;      }
      if (y < radius)       { y = radius; dy = 1;      }
      repaint();
    }
  }
}
```



Animacija žogice je “hello world” ra unalniške animacije

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.

Thread je objekt tipa *Runnable*

```
public class NumberPrinter implements Runnable {
    int num;

    public NumberPrinter(int n) {
        num = n;
    }

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

} // NumberPrinter
```

Runnable objekt implementira metodo run().

```
Thread number1;
number1 = new Thread(new NumberPrinter(1));
number1.start();
```

Tvorba objekta Runnable.

Tvorba niti

Dva načina

1. Implementiramo vmesnik Runnable
   Implementiramo metodo run ()

2. Podedujemo razred Thread
   Prekrijemo metodo run ()

V metodo run( ) vnesemo “obnašanje”, ki ga želimo

Primer z dedovanjem razreda Thread

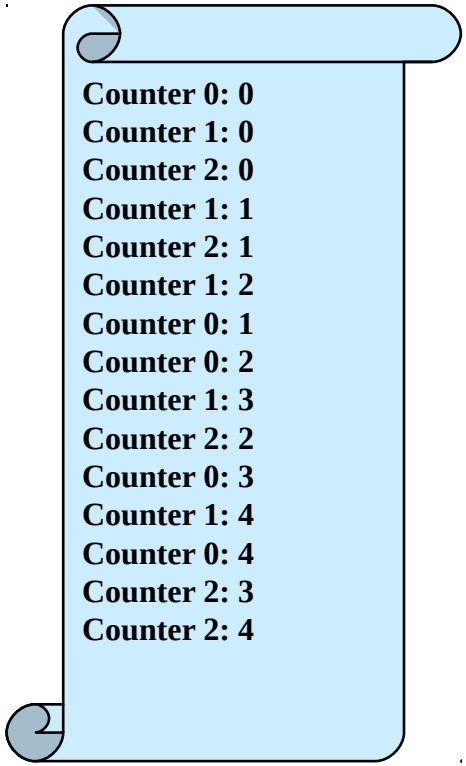
```
public class Counter extends Thread {
   private static int totalNum = 0;
   private int currentNum, loopLimit;

 public Counter(int loopLimit) {
       this.loopLimit = loopLimit;
       currentNum = totalNum++;
   }

   private void pause(double seconds) {
       try { Thread.sleep(Math.round(1000.0*seconds)); }
       catch(InterruptedException ie) { }
   }
   /** When run finishes, the thread exits. */
   public void run() {
       for(int i=0; i<loopLimit; i++) {
           System.out.printf("Counter %s: %s%n", currentNum, i);
           pause(Math.random()); // Sleep for up to 1 second
       }
   }
}
```

Preskus razreda Counter

```
public class CounterTest {
    public static void main(String[] args) {
        Counter c1 = new Counter(5);
        Counter c2 = new Counter(5);
        Counter c3 = new Counter(5);
        c1.start();
        c2.start();
        c3.start();
    }
}
```



Kako podedujemo vmesnik Runnable

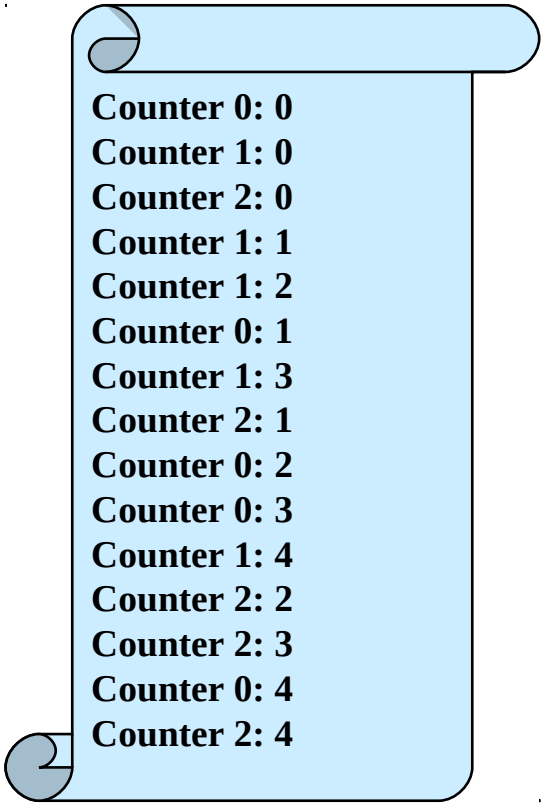
```
public class ThreadedClass extends AnyClass implements Runnable {
    public void run() {
        // Thread behavior here
        // If you want to access thread instance
        // (e.g. to get private per-thread data), use
        // Thread.currentThread().
    }
    public void startThread() {
        Thread t = new Thread(this);
        t.start(); // Calls back to run method in this
    }
...
}
```

Primer z uporabo vmesnika Runnable

```
public class Counter2 implements Runnable {
    private int totalNum, loopLimit;
    public Counter2(int loopLimit) {
        this.loopLimit = loopLimit;
        for(int i=0; i<3; i++) {
            Thread t = new Thread(this);  t.start();
        }
    }
    private void pause(double seconds) {
        try { Thread.sleep(Math.round(1000.0*seconds)); }
        catch(InterruptedException ie) {}
    }
    public void run() {
        int currentNum = totalNum++; // Race condition? See next example.
        for(int i=0; i<loopLimit; i++) {
            System.out.printf("Counter %s: %s%n", currentNum, i);
            pause(Math.random()); // Sleep for up to 1 second.
        }
    }
}
```

Preskus števca

```
public class Counter2Test {
    public static void main(String[ ] args) {
        Counter2 c1 = new Counter2(5);
    }
}
```

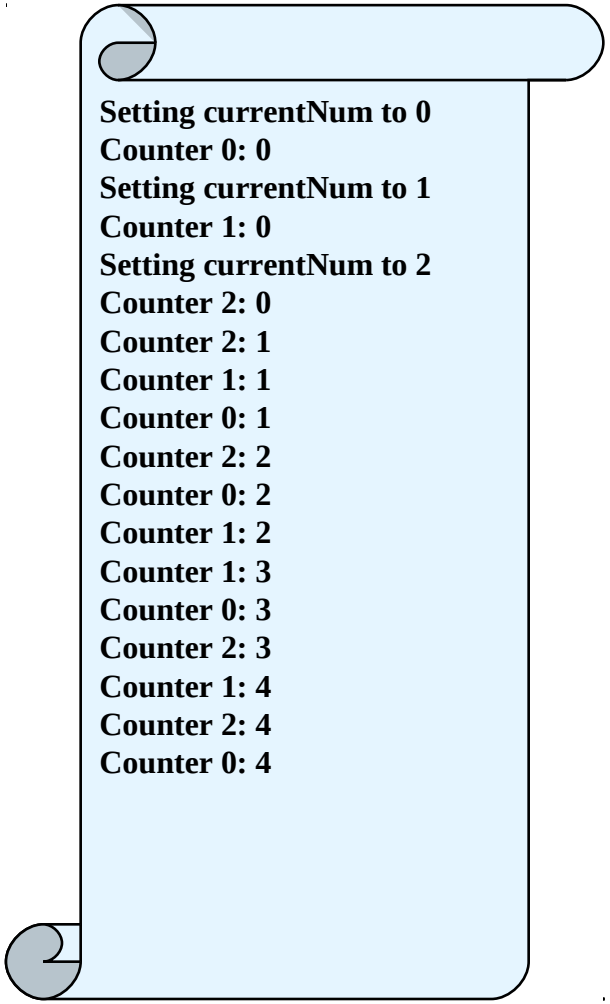


Problem: tekmovalne razmere (race condition)

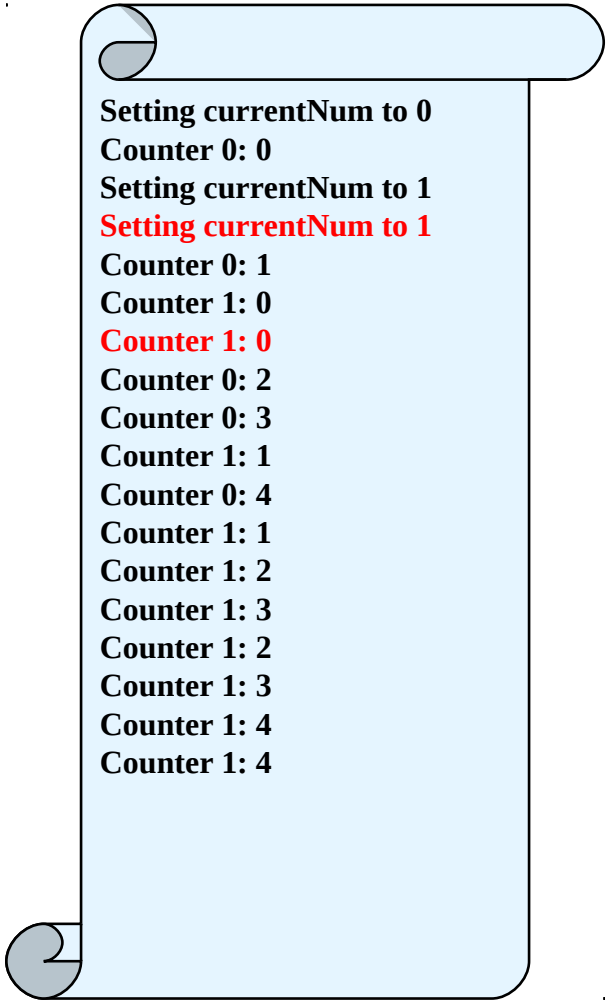
```
public class BuggyCounterApplet extends Applet implements Runnable{
    private int totalNum = 0; int loopLimit = 5;
    public void init() {
        Thread t;
        for(int i=0; i<3; i++) {
            t = new Thread(this);   t.start();
        }
    }
   private void pause(double seconds) {
      try { Thread.sleep(Math.round(1000.0*seconds)); }
      catch(InterruptedException ie) { }
   }
    public void run() {
        int currentNum = totalNum;
        System.out.printf("Setting currentNum to %s%n“, currentNum);
        totalNum = totalNum + 1;
        for(int i=0; i<loopLimit; i++) {
            System.out.printf("Counter %s: %s%n",  currentNum, i);
            pause(Math.random());
        }
    }
}
```

Kaj je narobe s to kodo ?

Izpis ni vedno enak



Obi ajen izpis



V asih pa dobimo to

Tekmovalne razmere: rešitev

```
public void run() {
    int currentNum = totalNum++;
    System.out.printf("Setting currentNum to %s%n", currentNum);
    for(int i=0; i<loopLimit; i++) {
        System.out.printf("Counter %s: %s%n", currentNum, i);
        pause(Math.random());
    }
}
```

Vse naredimo v enem koraku

Splošna rešitev tekmovanja za vire

Sinhroniziramo del kode

```
synchronized(nekObjektt) {
   koda
}
```

Fiksiranje prejšnjih tekmovalnih razmer:

```
public void run() {
     synchronized(this) {
          int currentNum = totalNum;
          System.out.printf("Setting currentNum to %s%n", currentNum);
          totalNum = totalNum + 1;
      }
      for(int i=0; i<loopLimit; i++) {
           …
     }
}
```

Poenostavljena razlaga
Ko nit vstopi v sinhronizirani del kode, ne more vanjo vstopiti novena 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

Primer: sodelujo e niti

- Niti, ki sodelujejo, potrebujejo eksplicitno sinhronizacijo in koordinacijo.
- Problem: Simulacija trgovinekjer 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

```
public class Kupec extends Thread {
    private static int number = 10000;      // ID oznaka prvega kupca
    private int id;
    private Stevilo stevilcnik;
    public Kupec( Stevec s) {
        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) { }
    }
}
```

Spremenljivka razreda

edinstven ID.

Ta kupec bo morda moral čakati.

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

Problem: neobstoje i 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.

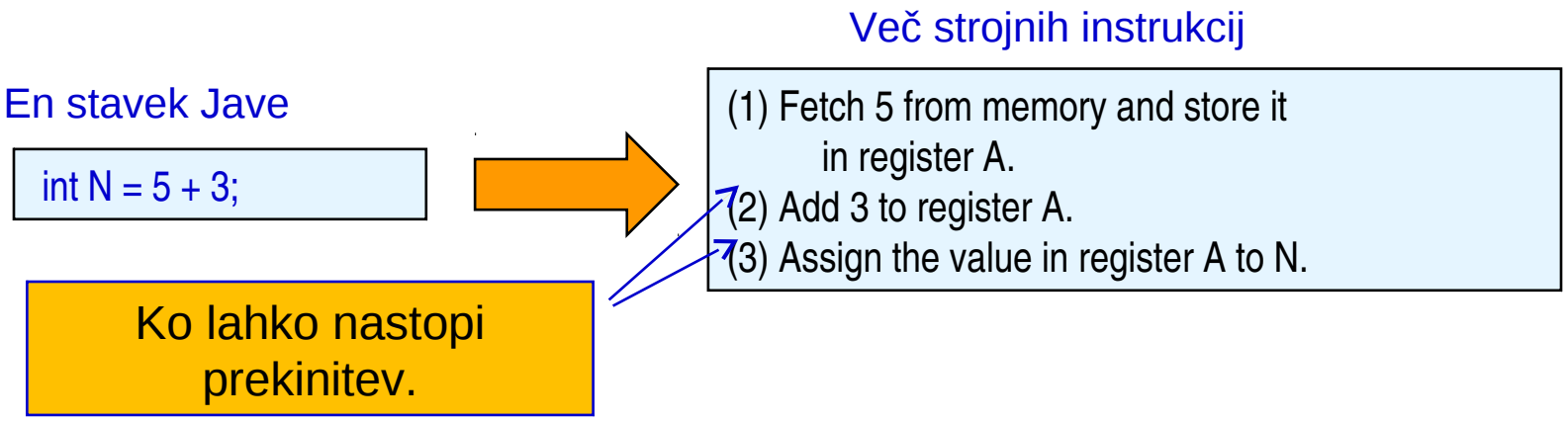
To bi lahko dosegli z naslednjo metodo v razredu Stevec:

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

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

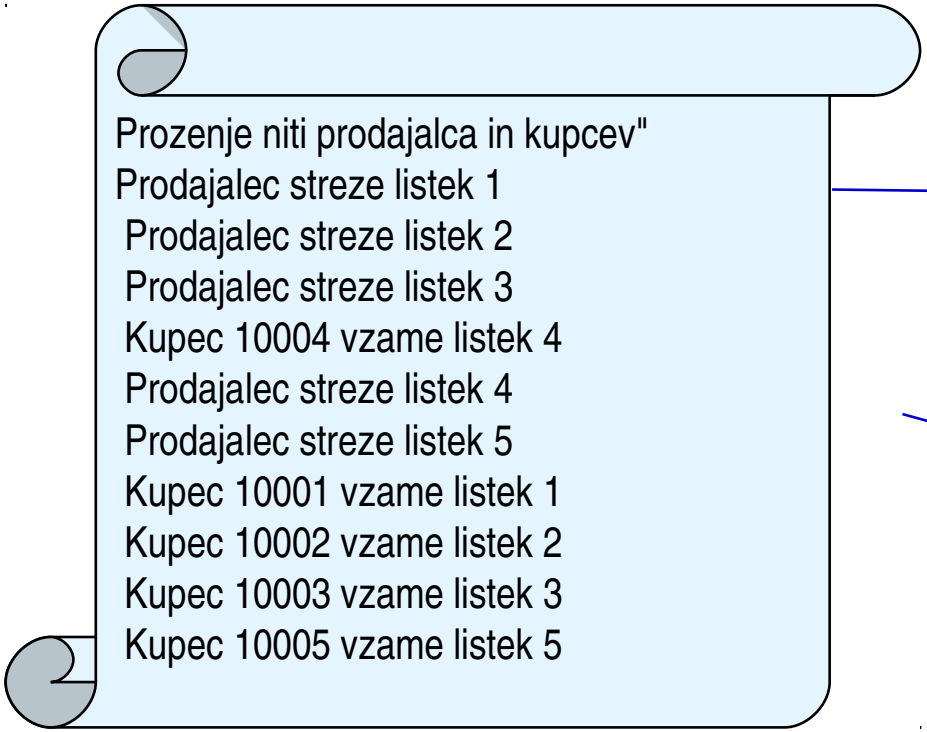
Asinhrona narava niti

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



Konkuren nost niti – kriti ne 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:



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 (takeANumber.customerWaiting())
                stevilcnik.naslednjiKupec();
        } catch (InterruptedException e) { }
    } // for
} // run()
```

Samo postreži kupca.

Koordinirane niti: pravilni izpis

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

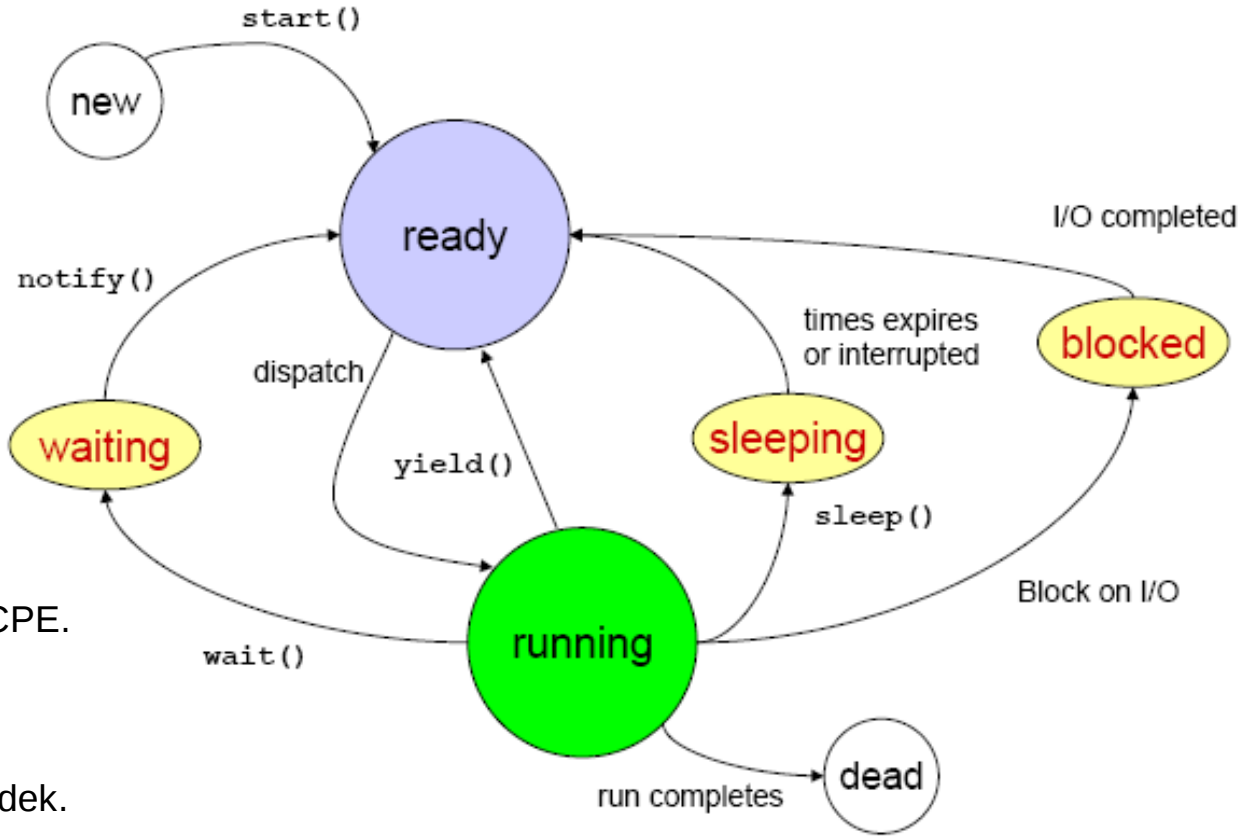
Kupci so postreženi v pravilnem zaporedju ne glede na to, kako prihajajo.

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

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.

Stanja niti



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!

Spanje niti

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

```
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()
```

Zaspi za interval do 1000 milisekund.

Niti tečejo v poljubnem zaporedju.

1452231453214315423152423541243235415523113435451

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
```

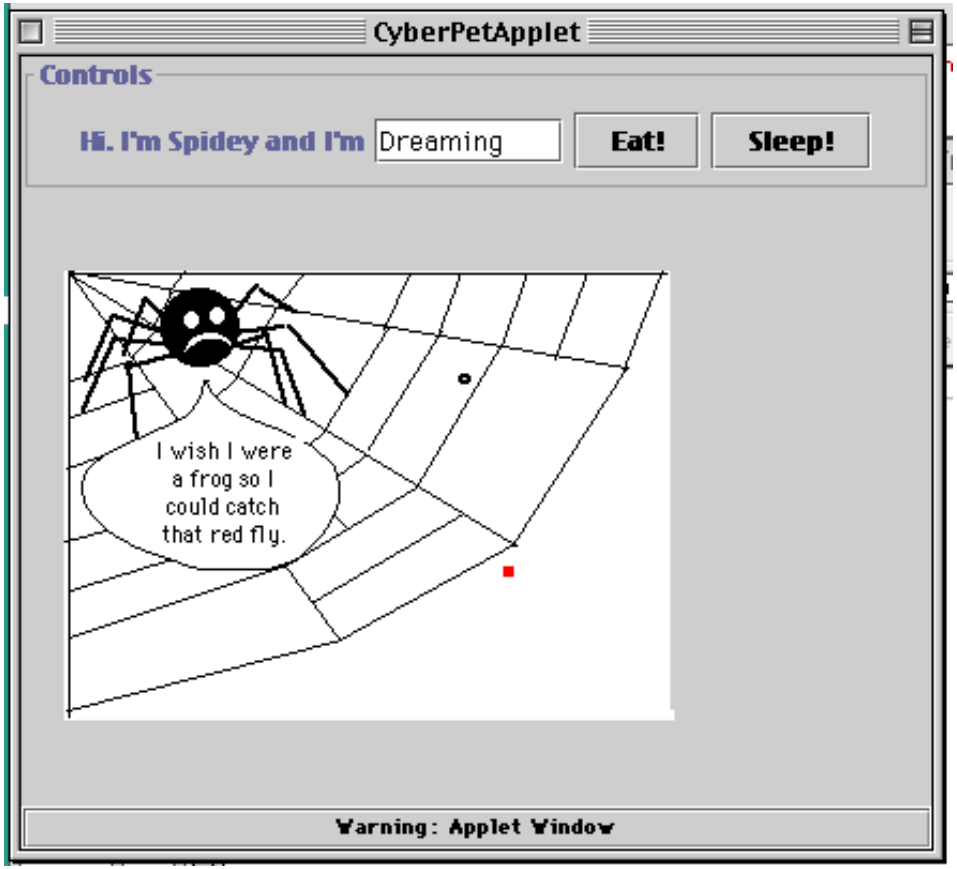
Omejitve mehanizma wait/notify

- 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.

Še en primer: pajek in muha

Problem:

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



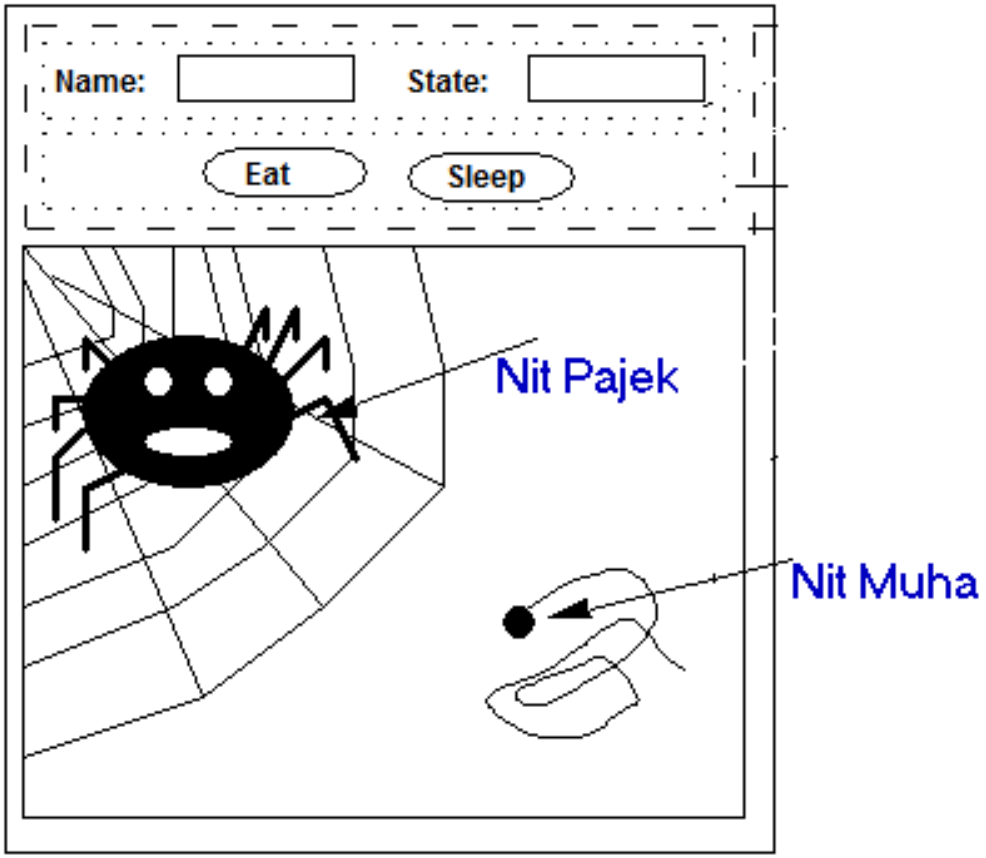
Primer je le nakazan, dopolni ga v popolno igrico

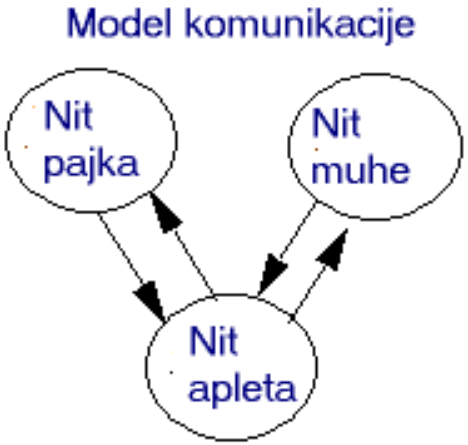
Demo

Organizacija programa

3 objekti, 3 ločene niti:

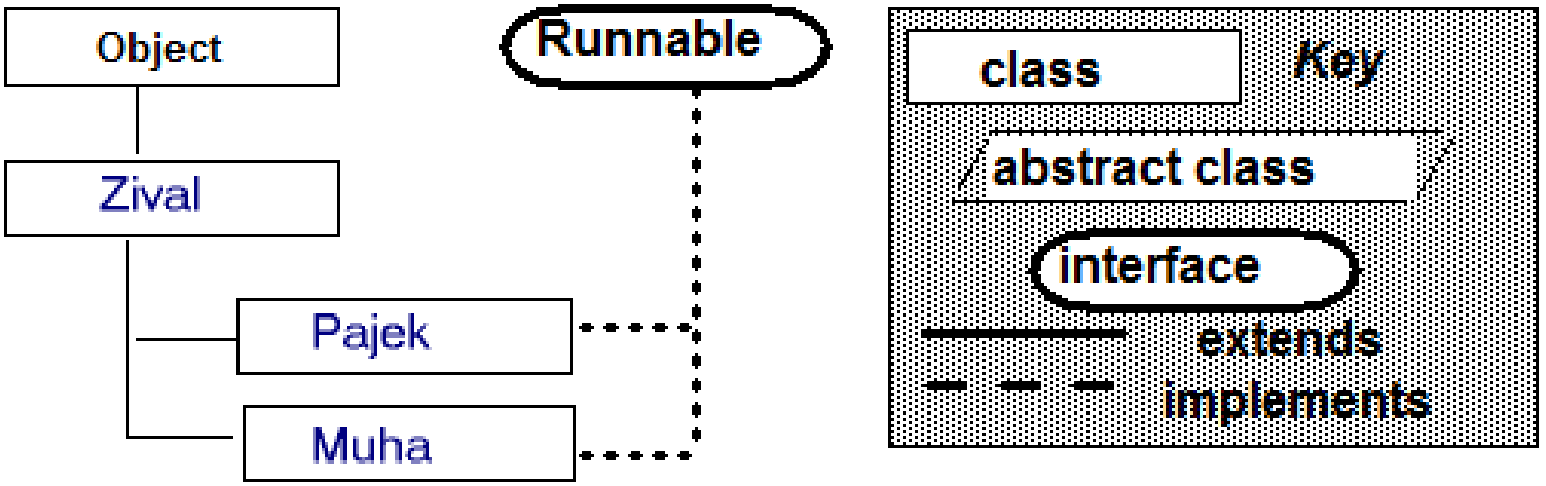
- Aplet: implementira vmesnik.
- Pajek: naš miljenček
- Muha: brenčeča muha





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 DEAD = -1;
    public static final int EATING = 0;
    public static final int SLEEPING = 1;
    public static final int THINKING = 2;
    public static final int DREAMING = 3;
    public static final int FLYING = 4;     // za letece zivali

    public void dream() {
        state = DREAMING;
    }
    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 != DEAD) {
            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 = DEAD;
}
```

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 = EATING;
       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(FROG_X, ZABA_Y, muhaLocation.x, flyLocation.y);  // pojej muho
    g.drawLine(FROG_X + 1, ZABA_Y + 1, muhaLocation.x + 1, muhaLocation.y + 1);
    g.drawLine(FROG_X + 2, ZABA_Y + 2, muhaLocation.x + 1, muhaLocation.y + 1);
    g.drawLine(FROG_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 Pajekr (“Pajkec", this);  // Tvorba pajka
private Muha muhica = new Muha(this);                          //  in muhe
```

V init() sproži njune niti

```
new Thread(pajkecy).start();     // Start spidey thread (in init())
new Thread(muhica).start();      // Start the fly thread (in init())
```

Posreduje njuno interakcijo

To kliče pajek.

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

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*.