# Java nuoseklių procesų komunikavimas (Java CSP)

Communicating Sequential[1] Processes for Java (JCSP)

[1] vykdomų lygiagrečiai

# Java CSP paketas

# Java CSP literatūra

- Communicating Sequential Processes for Java
  http://www.cs.kent.ac.uk/projects/ofa/jcsp/
  (jcsp-1.1-rc4.jar)
- CSP for Java programmers, Part 1
  http://www.ibm.com/developerworks/java/library/j-csp1.html
- CSP for Java programmers, Part 2
  http://www.ibm.com/developerworks/java/library/j-csp2

# jcsp-1.1-rc4.jar paketai

**org.jcsp.awt** This provides CSP extensions for all awt components – GUI events and widget configuration map to channel communications.

**org.jcsp.lang** This provides classes and interfaces corresponding to the fundamental primitives of CSP.

**org.jcsp.net** This is main package for JCSP.NET.

**org.jcsp.net.remote** Supports the remote spawning of processes at other nodes.

. . . kiti

# Java CSP procesai

# JCSP procesai

`public interface` **`CSProcess`**

- Tai JCSP sąsaja kurti procesus – aktyvius komponentus, turinčius savyje duomenis ir metodus tiems duomenims apdoroti.
- CSP procese tiek duomenys, tiek jų apdorojimo metodai yra **privatūs**.
- Procesai sąveikauja per CSP kanalus (*channels*).
- Procesas – tai klasės, realizuotos naudojant *CSProcess* sąsają, objektas. Jo veikimas nusakomas **run** metodu: `void run()` – aprašo procese vykdomus veiksmus.

# Lygiagrečiai vykdomų procesų rinkinys (1)

```
public class Parallel extends Object
                          implements CSProcess
```

Sudaro lygiagretų `CSProcess` rinkinį (sudėtinį procesą).
Konstruktoriai:

- **`Parallel()`** – sukuria naują tuščią `Parallel` klasės objektą;
- **`Parallel(CSProcess[] procesai)`** – sukuria nurodytų procesų rinkinį;
- **`Parallel(CSProcess[][] procesai)`** – sukuria nurodytų procesų rinkinį.

# Lygiagrečiai vykdomų procesų rinkinys (2)

`class` **`Parallel`**

Metodai:

- **`void addProcess(CSProcess procesas)`** – įtraukia į rinkinį nurodytą procesą;
- **`public void addProcess(CSProcess[] procesai)`** – įtraukia į rinkinį nurodytą procesų masyvą;
- **`public void run()`** – lygiagrečiai vykdo visus rinkinio procesus.

# Procesų pavyzdys (a)

```
import org.jcsp.lang.*;
class Procesas implements CSProcess {
  . . .
  Procesas(...) { ... }
  ...
  public void run() {
    ...
  }
}
```

# Procesų pavyzdys (b)

```
public class Procesai {
  Procesas P0;
  Procesas[] P = new Procesas[10];
  Parallel visi = new Parallel();
  Procesai() {
    ...
  }
  public void vykdyti() {
    ...
  }
  public static void main(String[] args) {
    new Procesai().vykdyti();
  }
}
```

# Procesų pavyzdys (c)

```
Procesai() {
  P0 = new Procesas(...);
  for (int i = 0; i < 10; i++)
    P[i] = new Procesas(...);
}
public void vykdyti() {
  visi.addProcess(P0);
  visi.addProcess(P);
  visi.run();
}
```

# Procesų komunikavimas "many-to-one"

This implements a any-to-one object (integer) channel, safe for use by many writers and one reader. The reading process may ALT on this channel.

# Informacijos perdavimas JCSP kanalu

# Any-to-one kanalų klasės ir sąsajos (1)

- public class **Channel**
  - public static Any2OneChannel **any2one**()
  - public static Any2OneChannelInt **any2oneInt**()
- public interface **Any2OneChannel**
  - AltingChannelInput **in**()
  - ChannelOutput **out**()
- public interface **Any2OneChannelInt**
  - AltingChannelInputInt **in**()
  - ChannelOutputInt **out**()

# Komunikavimo pavyzdys (a)

```
class Siuntėjas implements CSProcess {
  private String vardas;
  private int x;
  private ChannelOutputInt kan;
  Siuntėjas( ... ) {
    ...
  }
  public void run() {
    kan.write(x);
  }
}
```

# Komunikavimo pavyzdys (b)

```
class Gavėjas implements CSProcess {
  private String vardas;
  private int a;
  private ChannelInputInt kan;
  Gavėjas( ... ) {
    ...
  }
  public void run() {
    for(int i=0; i<2; i++) {
      a = kan.read();
      ...
    }
  }
}
```

# Komunikavimo pavyzdys (c)

```
public class ReadWrite {
  Any2OneChannelInt k = Channel.any2oneInt();
  Gavėjas gav;   Siuntėjas s1, s2;
  Parallel visi = new Parallel();
  ReadWrite() {
    gav = new Gavėjas("Gavėjas", k.in());
    s1 = new Siuntėjas("1 siunt", k.out());
    s2 = new Siuntėjas("2 siunt", k.out());
  }
  public void vykdyti() {
    visi.addProcess(gav);
    visi.addProcess(s1);
    visi.addProcess(s2);
    visi.run();
} }
```

# Procesų komunikavimas "point-to-point"

# Point-to-point kanalų klasės ir sąsajos (1)

- public interface **ChannelOutput**
  - void **write**(Object objektas)
- public interface **ChannelOutputInt**
  - void **write**(int i)
- public interface **ChannelInput**
  - Object **read**()
- public interface **ChannelInputInt**
  - int **read**()

# Point-to-point kanalų klasės ir sąsajos (2)

- public class **Channel**
  - public static One2OneChannel **one2one**()
  - public static One2OneChannelInt **one2oneInt**()
- public interface **One2OneChannel**
  - AltingChannelInput **in**()
  - ChannelOutput **out**()
- public interface **One2OneChannelInt**
  - AltingChannelInputInt **in**()
  - ChannelOutputInt **out**()

# Komunikavimo pavyzdys (a)

```
class Siuntėjas implements CSProcess {
  private String vardas;
  private int x;
  private ChannelOutputInt kan;
  Siuntėjas(String v, ChannelOutputInt k) {
    vardas = v; x = 9999;
    kan = k;
  }
  public void run() {
    kan.write(x);
  }
}
```

# Komunikavimo pavyzdys (b)

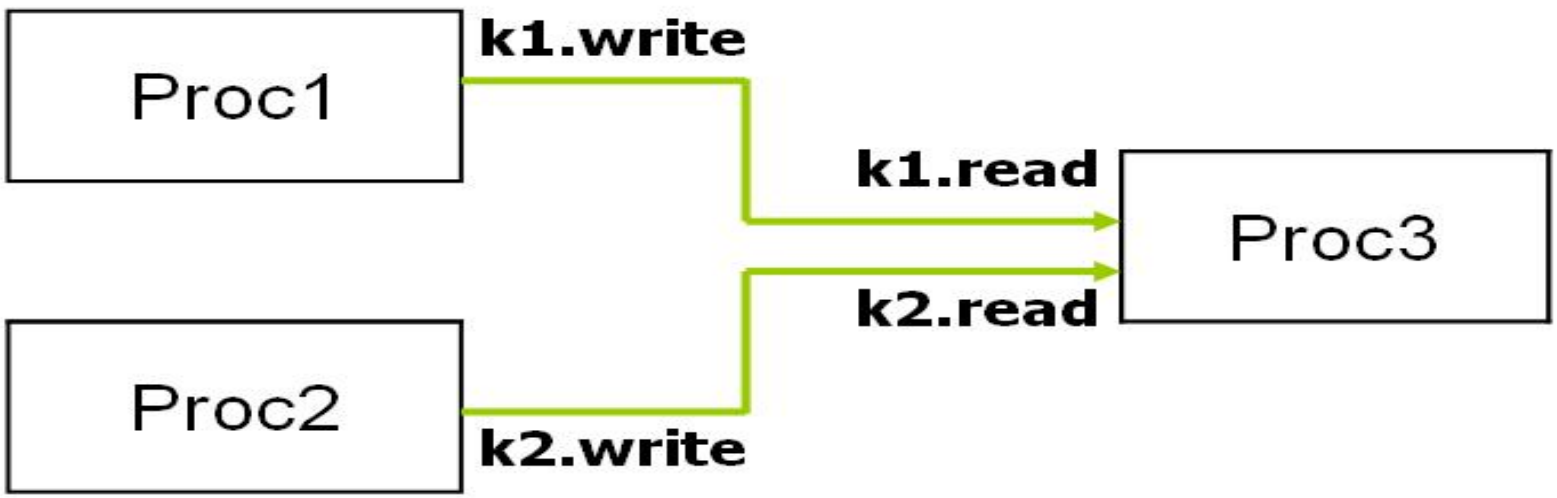
```
class Gavėjas implements CSProcess {
  private String vardas;
  private int a;
  private ChannelInputInt kan;
  Gavėjas(String v, ChannelInputInt k) {
    vardas = v; a = 0;
    kan = k;
  }
  public void run() {
    a = kan.read();
  }
}
```

# Komunikavimo pavyzdys (c)

```
public class ReadWrite {
  One2OneChannelInt k1 = Channel.one2oneInt();
  Gavėjas gav;   Siuntėjas siunt;
  Parallel visi = new Parallel();
  ReadWrite() {
    gav = new Gavėjas("Gavėjas", k1.in());
    siunt = new Siuntėjas("Siunt", k1.out());
  }
  public void vykdyti() {
    visi.addProcess(gav);
    visi.addProcess(siunt);
    visi.run();
  }
}
```

# Alternatyvos

# Alternatyvus duomenų priėmimas



# Alternatyvaus priėmimo klasės ir sąsajos (1)

- public abstract class **Guard**
  - **Guard**();
- public class **CSTimer** extends Guard
  - public void **after**(long msecs);
  - public long **read**();
  - public void **sleep**(long msecs);
  - public void **setAlarm**(long msecs);
  - public long **getAlarm**();
- public class **Skip** extends Guard
  - public void **run**();
- public class **Stop** extends Guard
  - public void **run**();

# Alternatyvaus priėmimo klasės ir sąsajos (2)

- public abstract class **AltingChannelInput** extends Guard
  - public abstract boolean **pending**();
- public abstract class **AltingChannelInputInt** extends Guard
  - public abstract boolean **pending**();
- public class **Alternative**
  - **Alternative**(Guard[] guard);
  - int **select**(); int **fairSelect**(); int **priSelect**();
  - int **select**(boolean[] preCondition);
    int **fairSelect**(boolean[] preCondition);
    int **priSelect**(boolean[] preCondition);

# Alternatyvų pavyzdys (a)

```
class Siuntėjas implements CSProcess {
  private String vardas;
  private int x;
  private ChannelOutputInt kan;
  Siuntėjas(String v, int xx,
            ChannelOutputInt k) {
    vardas = v; x = xx;
    kan = k;
  }
  public void run() {
    kan.write(x);
  }
}
```

# Alternatyvų pavyzdys (b)

```
class Gavėjas implements CSProcess {
  private String vardas;
  private int a;
  private AltingChannelInputInt kan1, kan2;
  Gavėjas(String v, AltingChannelInputInt k1,
          AltingChannelInputInt k2) {
    vardas = v; a = 0;
    kan1 = k1; kan2 = k2;
  }
  public void run() {
    ...
  }
}
```

# Alternatyvų pavyzdys (c)

```
// Gavėjas
public void run() {
  Guard[] g = new Guard[2];
  g[0] = kan1; g[1] = kan2;
  final Alternative alt = new Alternative (g);
  int i=0; while(i<2) {
    switch (alt.fairSelect()) {
    case 0:
      a = kan1.read(); i++; break;
    case 1:
      a = kan2.read(); i++; break;
    }
  }
}
```

# Alternatyvų pavyzdys (d)

```
public class Alternatyva {
  One2OneChannelInt k1 = Channel.one2oneInt();
  One2OneChannelInt k2 = Channel.one2oneInt();
  Gavėjas gav;
  Siuntėjas s1, s2;
  Parallel visi = new Parallel();
  Alternatyva() {
    gav = new Gavėjas("Gav",k1.in(),k2.in());
    s1 = new Siuntėjas("1 siunt",11,k1.out());
    s2 = new Siuntėjas("2 siunt",22,k2.out());
  }
  ...
}
```

# Alternatyvų variantai

# Saugomos alternatyvos

```
int fairSelect(boolean[] preCondition)
int priSelect(boolean[] preCondition)
int select(boolean[] preCondition)
```

Returns the index of one of the ready guards whose `preCondition` is `true`.

# Saugomos alternatyvos: buferis

```
Guard[] guards = {dėti, imti};
Alternative alt = new Alternative(guards);
for ( ; ; ) {
  galima[0] = (count < n);
  galima[1] = (count > 0);
  switch (alt.Select(galima)) {
    case 0:
      gaminys = dėti.read(); break;
    case 1:
      g = imti.read();
      perduoti.write(gaminys);
      break;
  }
}
```

# Prioritetinė alternatyva: ribotas perdavimas

```
Guard[] guards = {rėžiai, reikšmė};
Alternative alt = new Alternative(guards);
a = rėžiai.read(); b = rėžiai.read();
for ( ; ; ) {
  switch (alt.priSelect()) {
    case 0:
      a = rėžiai.read(); b = rėžiai.read();
      break;
    case 1:  v = reikšmė.read();
      if (v>a && v<b) kout.write(v);
      break;
  }
}
```

# `Timeout` alternatyva

`class` **`CSTimer`**

This is a Guard for setting timeouts in an Alternative.

`public long` **`read`**`()`

Returns the current system time in msecs.

`public void` **`setAlarm`**`(long msecs)`

Sets the absolute timeout value that will trigger an Alternative select operation (when this CSTimer is one of the guards with which that Alternative was constructed).

# `Timeout` alternatyva: darbininkas

```
for ( ; ; ) {
  dirbti();
  pranešti.write(any);
}
```

# `Timeout` alternatyva: prižiūrėtojas

```
CSTimer tim = new CSTimer();
long timeout = tim.read();
Guard[] guards = {pranešti, tim};
Alternative alt = new Alternative(guards);
long timeout = ...;
tim.setAlarm(tim.read() + timeout);
for ( ; ; ) {
  int i = alt.fairSelect();
  if (i == 0) any = pranešti.read();
  else bausti();
}
```

# Klausimai pakartojimui

1. Kaip apibūdinamas JCSP procesas?
2. Kokiu būdu tarpusavyje sąveikauja JCSP procesai?
3. Kokiu būdu sudaromas ir vykdomas lygiagrečiųjų JCSP procesų rinkinys?
4. Kokie veiksmai sudaro komunikavimo porą?
5. Kuo ypatingas many-to-one, point-to-point komunikavimas?
6. Kokie veiksmai gali sudaryti alternatyvų rinkinį?
7. Kada naudojamos saugomos alternatyvos?