Bendrų kintamųjų apsauga naudojant monitorius

Monitorių paskirtis ir savybės

Monitorius

- **Monitor** = įspėjėjas (lot.) – įtaisas kam nors stebėti, testuoti, užrašinėti.
- programavime: C.A.Hoare 1974 m.:
  C. A. R. Hoare, "Monitors: An Operating System Structuring Concept", Communications of the ACM, Vol. 17, No. 10, October, 1974, pp. 549-557.
- kritinis regionas, monitorius – programos kodo dalis, kuri **visada** vykdoma su tarpusavio išskyrimu.

Monitoriaus struktūra

Monitorių sudaro:

- bendrieji saugomi duomenys;
- atominių veiksmų (metodų), skirtų saugomiems duomenims apdoroti, rinkinys;
- sąlyginių kintamųjų rinkinys.

C.A.R. Hoare monitorius

```
single resource: monitor
begin busy: Boolean;
    nonbusy: condition;
  procedure acquire;
    begin
      if busy then nonbusy.wait;
      busy := true
    end;
  procedure release;
    begin
      busy := false;
      nonbusy.signal
    end;
  busy := false; comment initial value;
end single resource
```

Monitoriaus sąlygos kintamieji

- **`condition`** tipas; // `nonbusy:condition;`
- **`nonbusy.wait;`**
  // išlaisvinti monitorių ir blokuoti procesą,
  // vykdantį `wait`, prie sąlygos `nonbusy`
- **`nonbusy.signal;`**
  // išlaisvinti vieną procesą iš laukiančiųjų
  // prie sąlygos `nonbusy`

Monitoriaus ypatybės ir naudojimo problemos

- visi procesų atominiai veiksmai yra vienoje vietoje;
- monitoriaus viduje gali būti kreipiniai į kitus monitorius.
- metodas, vykdomas vieno monitoriaus viduje, gali kviesti kito monitoriaus metodą;
- procesas, vykdantis **`signal`**, yra monitoriuje, o procesas, išblokuojamas **`signal`**, turi grįžti į monitorių (*signal-and-exit* ar *signal-and-continue*).

Java monitoriai

Java monitorių realizacija (1)

- Kiekvienas Java objektas turi su juo susietą monitorių.
- Kiekviena Java klasė turi su ja susietą monitorių.
- Monitorius nerealizuojamas, jei nenaudojami **sinchronizuoti** (*synchronized*) metodai.
- Monitorius – tai užraktas (*lock*), nustatantis objekto (klasės) panaudojimo tvarką ir vienu metu leidžiantis jį naudoti tik vienai gijai.

Java monitorių realizacija (2)

- Tarpusavio išskyrimas (*mutual exclusion*) - "užrakinant" objektą su **`synchronized`** (realizuoja JVM).
- Sąlyginė sinchronizacija (*condition synchronization*) – naudojant `Object` klasės **`wait`**, **`notify`**, **`notifyAll`** metodus.

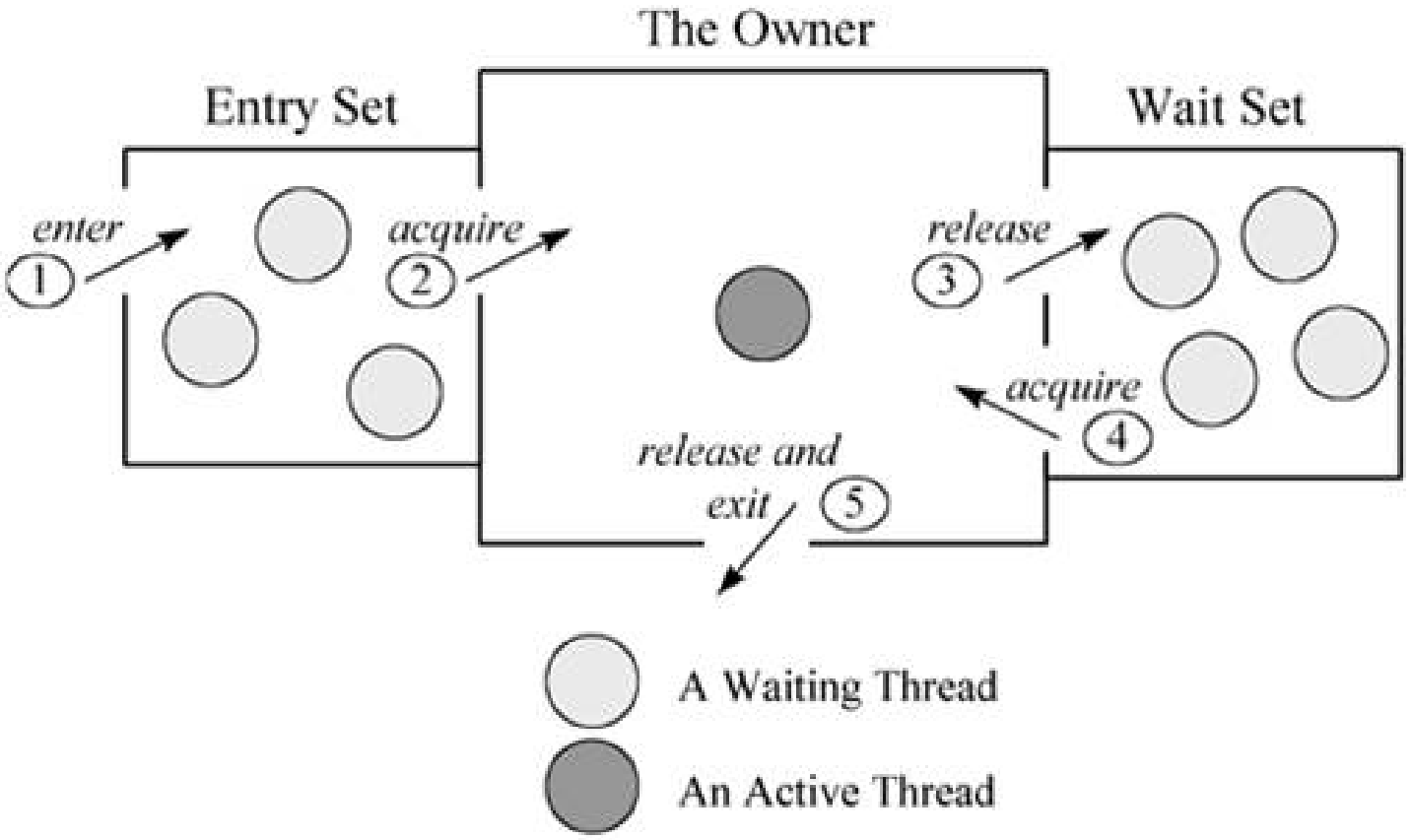
Reikalingi `Object` klasės metodai

- **`notify()`**
  Wakes up a single thread that is waiting on this object's monitor.
- **`notifyAll()`**
  Wakes up all threads that are waiting on this object's monitor.
- **`wait()`**
  Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
- **`wait(long timeout)`**
  Waits for notification or until the timeout period has elapsed. timeout is measured in milliseconds.

Gijų būsenos, naudojant monitorių

- "entering the monitor",
- "acquiring the monitor",
- "owning the monitor",
- "releasing the monitor",
- "exiting the monitor".

Java monitorių veikimas (1)



Java monitorių veikimas (2)

- JVM naudoja "***signal-and-continue***" ("*wait-and-notify*") tipo monitorius.
- Gija, naudojanti monitorių, gali sustabdyti save vykdydama **`wait`** metodą.
- Gija vykdydama **`wait`**, palieka monitorių ir pereina į laukiančiųjų eilę (*a wait set*).
- Gija lieka laukiančiųjų eilėje iki tol, kol kita gija neįvykdys **`notify`** metodo.
- Gija, įvykdžiusi **`notify`**, lieka monitoriuje.
- Laukianti gija gali pereiti į monitorių po to, kai gija, įvykdžiusi **`notify`**, palieka monitorių.

Gijos ir monitorius

- Jei gija, paliekanti monitorių, nevykdė **`notify`**, dėl monitoriaus varžosi tik gijos iš "*entry set*".
- Jei gija, paliekanti monitorių, vykdė **`notify`**, dėl monitoriaus varžosi gijos iš "*entry set*" ir "*wait set*".
- Jei gija vykdė **`wait`** metodą su nurodytu laiko intervalu, praėjus nurodytam laikui JVM įvykdo **`notify`** šiai gijai.

`notify` veikimas

- **`notify`** metodas "prižadina" vieną giją iš "*wait set*",
  `notifyAll` metodas "prižadina" visas gijas iš "*wait set*".
- Nuo JVM realizacijos priklauso:
  - kuri gija iš "*wait set*" bus "prižadinta" vykdant **`notify`**,
  - kokia tvarka "prižadinamos" gijos iš "*wait set*" vykdant **`notifyAll`**,
  - kokia tvarka gijos iš "*entry set*" įgyja monitorių,
  - kaip pasirinkti tarp "*entry set*" ir "*wait set*" gijų vykdant **`notify`**.

`wait` veikimas (1)

Gija, įvykdžiusi **`wait`**, pereina į laukimo būseną iki tol, kol:

- kita gija įvykdys **`notify`** ir planuotojas išrinks šią giją tolimesniam vykdymui,
- kita gija įvykdys **`notifyAll`**,
- kita gija pertrauks (*interrupt*) šią giją,
- praeis **`wait`** nurodytas laiko tarpas.

`wait` veikimas (2)

`wait` metodas gali sukelti tokias išimtis (*Exception*):

- *IllegalArgumentException* – blogai nurodytas laiko intervalas,
- *IllegalMonitorStateException* – gija nėra objekto monitoriaus savininkė,
- *InterruptedException* – giją pertraukia kita gija.

`synchronized` taikymas

`synchronized` gali būti taikoma:

- objektų metodams (išskyrus konstruktorių),
- statiniams klasės metodams,
- sakinių blokams.

`volatile` kintamieji

The **`volatile`** keyword is used on variables that may be modified simultaneously by other threads.
This warns the compiler to fetch them fresh each time, rather than caching them in registers.
This also inhibits certain optimisations that assume no other thread will change the values unexpectedly.
Since other threads cannot see local variables, there is never any need to mark local variables volatile.

Monitorių panaudojimas

Skaitiklis (1)

```
class SimpleBoundedCounter {
 protected long count = MIN;
 public synchronized long count()
      {return count;}
 public synchronized void inc()
      throws InterruptedException {...}
 public synchronized void dec()
      throws InterruptedException {...}
 protected void setCount(long newValue) {...}
 protected void awaitUnderMax()
      throws InterruptedException {...}
 protected void awaitOverMin()
      throws InterruptedException {...}
}
```

Skaitiklis (2)

```
public synchronized void inc()
            throws InterruptedException {
  awaitUnderMax();
  setCount(count + 1);
}
public synchronized void dec()
            throws InterruptedException {
  awaitOverMin();
  setCount(count - 1);
}
protected void setCount(long newValue) {
  count = newValue;
  notifyAll();
   // wake up any thread depending on new value
}
```

Skaitiklis (3)

```
protected void awaitUnderMax()
            throws InterruptedException {
  while (count == MAX) wait();
}
```

```
protected void awaitOverMin()
            throws InterruptedException {
  while (count == MIN) wait();
}
```

Skaitiklis (4). Klaidingas sinchronizavimas

```
void badInc() // Nenaudoti!
            throws InterruptedException {
  synchronized(this) {while (count >= MAX) wait();}
  // (*)
  synchronized(this) {++count; notifyAll();}
}


void badSetCount(long newValue) // Nenaudoti!
            throws InterruptedException {
  synchronized(this) {notifyAll(); }
  // (*)
  synchronized(this) {count = newValue;}
}
```

Gamintojas – Vartotojas (1)

```
class RibotasBuferis{ ... }
class Gamintojas implements Runnable { ... }
class Vartotojas implements Runnable { ... }

public class Gamykla {
  public static RibotasBuferis buffer =
                                  new RibotasBuferis();
  public static void main(String args []) {
    Thread gam  = new Thread(new Gamintojas(10));
    Thread vart = new Thread(new Vartotojas(10));
    vart.start();
    gam.start();
  }
}
```

G–V. RibotasBuferis (2a)

```
class RibotasBuferis {
  private static final int buferioDydis = 5;
  private int[] buferis;
  private int į, iš;
  private int yra;              // kiek įdėta
  public RibotasBuferis() { ... }
  public synchronized void įterpti(int daiktas)
                                    { ... }
  public synchronized int paimti() { ... }
}
```

G–V. RibotasBuferis (2b)

```
public RibotasBuferis() {
    į = 0;    iš = 0;  yra = 0;
    buferis = new int[buferioDydis];
}
public synchronized void įterpti(int daiktas) {
  try {
     while (yra == buferioDydis) wait();
  } catch (InterruptedException e) {}
    // padėti gaminį į buferį
  buferis[į] = daiktas;
  į = (į + 1) % buferioDydis;
  yra++;
  notifyAll();
}
```

G–V. RibotasBuferis (2c)

```
public synchronized int paimti() {
  int daiktas = 0;
  try {
    while (yra == 0) wait();
  } catch (InterruptedException e) {}
    // paimti gaminį iš buferio
  daiktas = buferis[iš];
  buferis[iš] = -999;
  iš = (iš + 1) % buferioDydis;
  yra--;
  notifyAll();
  return daiktas;
}
```

G–V. Gamintojas (3)

```
class Gamintojas implements Runnable {
  private int kiekGaminti;
  public Gamintojas(int kiekGaminti) {
    this.kiekGaminti = kiekGaminti;
  }
  public void run() {
    int gaminys = 10;
    for (int i = 0; i<kiekGaminti; i++) {
      // gaminti
      gaminys += 1;
      Gamykla.buferis.įterpti(gaminys);
    }
  }
}
```

G–V. Vartotojas (4)

```
class Vartotojas implements Runnable {
  private int kiekVartoti;
  public Vartotojas(int kiekVartoti) {
    this.kiekVartoti = kiekVartoti;
  }
  public void run() {
    int gaminys;
    for (int i = 0; i<kiekVartoti; i++) {
      gaminys = Gamykla.buferis.paimti();
      // vartoti
    }
  }
}
```

Pašto dėžutė 1-1 (1a)

```
class PaštoDėžutė {
  private int laiškas;
  ... volatile boolean yra;
  public PaštoDėžutė() {
    laiškas = 0; yra = false;
  }
  public synchronized void padėti(int naujas)
       { ... }
  public synchronized int paimti() { ... }
}
```

Pašto dėžutė 1-1 (1b)

```
public synchronized void padėti(int naujas) {
  try {
     while (yra) wait();
  } catch (InterruptedException e) {}
  laiškas = naujas; yra = true;
  notifyAll();
}
```

Pašto dėžutė 1-1 (1c)

```
public synchronized int paimti() {
  int naujasLaiškas = 0;
  try {
    while (!yra) wait();
  } catch (InterruptedException e) {}
  naujasLaiškas = laiškas;  yra = false;
  laiškas = -999;
  notifyAll();
  return naujasLaiškas;
}
```

Pašto dėžutė 1-n (1a)

```
class PaštoDėžutė {
  private int laiškas, kiekSkaito;
  private volatile boolean gRašyti;
  private volatile boolean gSkaityti[];
  public PaštoDėžutė(int kiekSkaito) {
    this.kiekSkaito = kiekSkaito;
    gSkaityti = new boolean[kiekSkaito];
    laiškas = 0; gRašyti = true;
    for (int i=0; i<kiekSkaito; i++)
      gSkaityti[i]= false;
  }
  public synchronized void padėti(int naujas)
      { ... }
  public synchronized int paimti(int k) { ... }
}
```

Pašto dėžutė 1-n (1b)

```
public synchronized void padėti(int naujas) {
  try {
    while (!gRašyti) wait();
  } catch (InterruptedException e) {}
  laiškas = naujas; gRašyti = false;
  for (int i=0; i<kiekSkaito; i++)
    gSkaityti[i] = true;
  notifyAll();
}
```

Pašto dėžutė 1-n (1c)

```
public synchronized int paimti(int k) {
  int naujasLaiškas = 0;
   try {
     while (!gSkaityti[k]) wait();
  } catch (InterruptedException e) {}
  naujasLaiškas = laiškas;
  gSkaityti[k] = false; gRašyti = true;
  for (int i=0; i<kiekSkaito; i++)
    gRašyti = gRašyti && (!gSkaityti[i]);
  notifyAll();
  return naujasLaiškas;
}
```

Pašto dėžutė 1-n (2)

```
class Rašytojas implements Runnable {
  private int kiekRašyti;
  private int laiškas = 10;
  public Rašytojas(int kiekRašyti) {
    this.kiekRašyti = kiekRašyti;
  }
  public void run() {
    for (int i = 0; i<kiekRašyti; i++) {
      laiškas += 1;
      Pastas.pD.padėti(laiškas);
    }
  }
}
```

Pašto dėžutė 1-n (3)

```
class Skaitytojas extends Thread {
  private int kiekSkaityti, nr;
  private int laiškas = 0;
  public Skaitytojas(int kiekSkaityti, int nr) {
    this.kiekSkaityti = kiekSkaityti;
    this.nr = nr;
  }
  public void run() {
  for (int i = 0; i<kiekSkaityti; i++)
      laiškas = Pastas.pD.paimti(nr);
  }
}
```

Pašto dėžutė 1-n (4)

```
public class Paštas {
  public static int kiekSkaito = 5;
  public static PaštoDėžutė pD =
                    new PaštoDėžutė(kiekSkaito);
  public static void main(String args []) {
    Thread raš = new Thread(new Rašytojas(20));
    Skaitytojas sk[] = new Skaitytojas[kiekSkaito];
    for (int i=0; i<kiekSkaito; i++)
      sk[i] = new Skaitytojas(20,i);
    for (int i=0; i<kiekSkaito; i++)
      sk[i].start();
    raš.start();
  }
}
```

Klausimai pakartojimui

1. Kas yra monitorius (programavime)?
2. Kuo pasižymi monitorius?
3. Kokios yra monitorių naudojimo problemos?
4. Kaip realizuotas monitorius Java kalboje?
5. Kokiu Kuo ypatingi sinchronizuoti Java klasių metodai?
6. Kokiu būdu įgyvendinama sąlyginė sinchronizacija Java monitoriuje?
7. Kiek skirtingų monitorių gali būti vienoje Java programoje?