



Programare orientată pe obiecte



1. Mai multe despre noua I/E Java
2. Fire de lucru Java

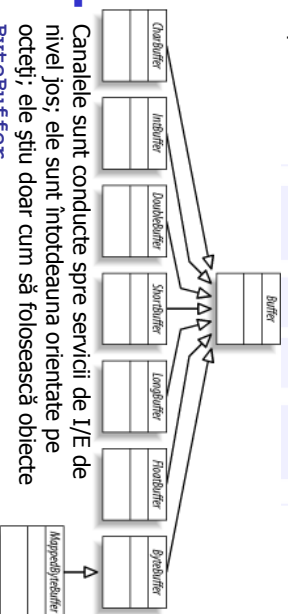
OOP13 - M. Joldes - T.U. Cluj

1



Buffer

- **Buffer** (zone tampon) au fost create în primul rând pe post de containere pentru datele trimise/receptionate pe/de pe canale.



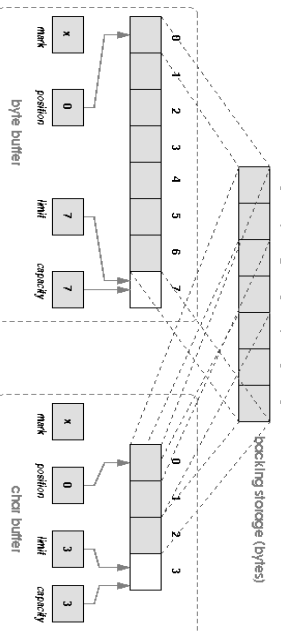
OOP13 - M. Joldes - T.U. Cluj

3



TECHNICAL UNIVERSITY

■ Vedere CharBuffer a unui ByteBuffer

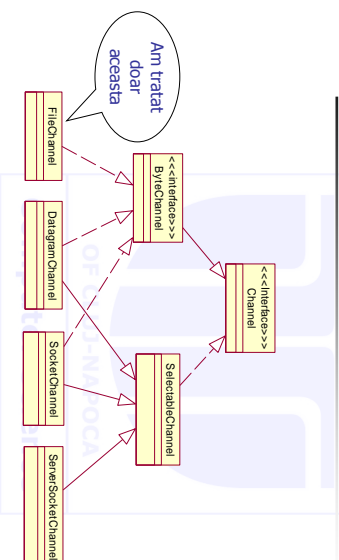


OOP13 - M. Joldes - T.U. Cluj

5



Iterahie simplificată pentru channel



OOP13 - M. Joldes - T.U. Cluj

2



Vederi pentru Buffer

- Presupunem că avem un fișier care conține caractere Unicode stocate ca valori pe 16 biți (codificare UTF-16 nu UTF-8)
 - UTF = Unicode Transformation Format
- Pentru a citi o bucată din acest fișier în zona tampon putem crea o vedere **CharBuffer** a octeților respectivi:
 - `CharBuffer charbuffer = ByteBuffer.allocate(1024).asCharBuffer();`
 - Creează o vedere a `ByteBuffer` care se comportă ca un `CharBuffer` (Combină fiecare pereche de octeți din tampon într-o valoare caracter pe 16 biți)
- Clasa `ByteBuffer` are metode de acces ad hoc la valorile primitive. De., pentru a accesa ca întreg patru octeți dintr-o zonă tampon:


```
int fileSize = ByteBuffer.getInt();
```

OOP13 - M. Joldes - T.U. Cluj

4



Exemplu de vederi pentru Buffer

```
import java.nio.*;

public class Buffers {
    public static void main(String[] args) {
        try {
            float[] floats = {
                6.612297E-39F, 9.918388E-39F,
                1.1093785E-38F, 1.092858E-38F,
                1.0469399E-38F, 9.183596E-39F
            };
            ByteBuffer bb =
                ByteBuffer.allocate(floats.length * 4);
        }
    }
}
```

```
FloatBuffer fb = bb.asFloatBuffer();
fb.put(floats);
CharBuffer cb = bb.asCharBuffer();
System.out.println(cb.toString());
} catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
}
```

OOP13 - M. Joldes - T.U. Cluj

6



Interschimbarea octeților

- **Endian-ness** : ordinea de combinare a octeților pentru a forma valori numerice mai mari
- Când octetul cel mai semnificativ ca ordine numerică este primul stocat în memorie (la adresa mai mică) avem ordinea **big-endian**
- Cazul opus, în care cel mai puțin semnificativ octet apare primul, este **little-endian**



00P13 - M. Joldes - T.U. Cluj

7



Zone tampon directe

- Datele încapsulate de o zonă tampon pot fi stocate în câteva moduri:
 - într-un tablou privat creat de obiectul **Buffer** (alocare),
 - într-un tablou furnizat de programator (împachetare), sau,
 - în cazul zonelor tampon *directe*, în **spațiul de memorie nativ** din alara JVM.
- La crearea unei zone tampon directe (prin apelul **ByteBuffer.allocateDirect()**), se alocă memorie sistem nativă și se împachetează într-o zonă tampon.
- Scopul principal al zonelor tampon directe este efectuarea de I/O peste canale.
- Implementările canalelor pot face ca operațiile de I/E de la nivelul SO să acționeze direct asupra spațiului de memorie nativă al unei zone tampon directe.

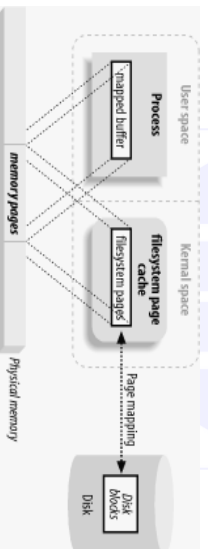
00P13 - M. Joldes - T.U. Cluj

9



Fișiere mapate pe memorie

- Alte procese care rulează în spațiul utilizator vor mapa pe același spațiu de memorie fizică, prin același cache al sistemului de fișiere și, de aceea pe aceleași date de pe disc.
- Fiecare dintre procese va vedea modificările efectuate de oricare alt proces.
- Aceasta poate fi exploatată ca o formă de memorie partajată persistentă.



00P13 - M. Joldes - T.U. Cluj

11



Vederi ale Buffer și Endian-ness

- Fiecare obiect tampon are o setare a **ordinii octeților**.
 - Cu excepția lui **ByteBuffer**, proprietatea poate fi numai citită și nu se poate schimba.
- Setarea ordinii octeților la obiectele **ByteBuffer** poate fi modificată oricând.
 - Aceasta afectează ordinea rezultată pentru orice vederi create pentru acel obiect **ByteBuffer**.
 - Dacă datele Unicode din fișier au fost codificate ca UTF-16LE (little-endian), trebuie să setăm ordinea pentru **ByteBuffer** înainte de a crea vederea **CharBuffer**:


```
ByteBuffer.order(ByteOrder.LITTLE_ENDIAN);
CharBuffer charBuffer = ByteBuffer.asCharBuffer();
```
- Noua vedere moștenește ordinea lui **ByteBuffer**
- Setarea ordinii octeților la momentul apelului **afectează** modul de combinare pentru formarea valorii returnate sau divizate pentru ceea ce este stocat în zona tampon.

00P13 - M. Joldes - T.U. Cluj

8



Fișiere mapate pe memorie

- **MappedByteBuffer** este o formă specializată de **ByteBuffer**.
 - În majoritatea sistemelor de operare se poate *mapa pe memorie* un fișier folosind un *apel sistem* de genul lui **mmap()** (sau ceva asemănător – atenție, acest apel nu este Java) pe un descriptor de fișier deschis.
 - Apelul lui **mmap()** returnează un pointer la un segment de memorie care reprezintă de fapt conținutul fișierului.
 - Extragerile din locațiile de memorie din zona respectivă vor întoarce datele din fișier de la deplasamentul **corespunzător**.
 - Modificările efectuate asupra spațiului de memorie sunt scrise în fișierul de pe disc.

00P13 - M. Joldes - T.U. Cluj

10



Exemplu MappedByteBuffer: inversarea octeților dintr-un fișier

```
import java.nio.*;
import java.nio.channels.*;
import java.io.*;

public class ReverseBytes {
    public static void main(String
    args[]) throws IOException {
        // verifica argumentul din
        if (args.length != 1) {
            System.err.println(
            "Lipsește numele fișierului");
            System.exit(1);
        }
        // maparea fișierul la buffer
        MappedByteBuffer mbb =
        FileChannel.MapMode.READ_W
        RTTE, 0, fc.size());
        // inverseaza octeții din fișier
        int len = (int)fc.size();
        for (int i = 0, j = len - 1;
            i < j;
                i++, j--)
            byte b = mbb.get(i);
            mbb.put(i, mbb.get(j));
            mbb.put(j, b);
        }
        // finish up
        fc.close();
        rat.close();
    }
}
FileChannel fc = rat.getChannel();}
```

00P13 - M. Joldes - T.U. Cluj

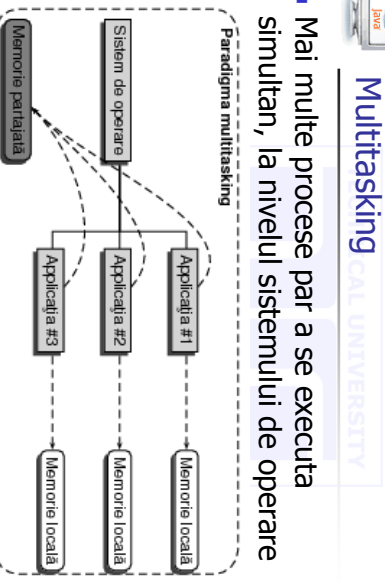
12



Introducere în firele de lucru Java

00P13 - M. Joldes - T.U. Cluj

19



00P13 - M. Joldes - T.U. Cluj

20

Multitasking

- Mai multe procese par a se executa simultan, la nivelul sistemului de operare



Sarcini ale firelor de lucru

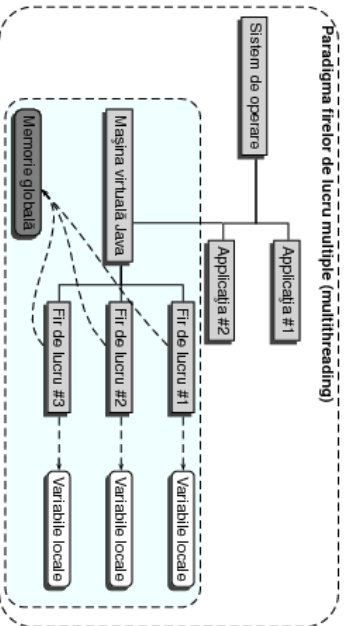
- Firele de lucru sunt utile în mai multe feluri:
 - Acolo unde trebuie să se întâmple mai multe lucruri simultan.
 - De ex., o aplicație multimedia poate necesita ca procesele audio, video și de control să se execute în paralel. Există adesea perioade de așteptare a răspunsului sistemelor de IE mai lente, timp în care procesorul poate face altceva.
 - Programe cum sunt sistemele server/client sunt mult mai ușor de protejată și scris folosind fire de lucru.
 - Algoritmi matematici, cum sunt sortarea, căutarea numerelor prime etc. sunt potrivite pentru prelucrarea paralelă.
 - Pe sistemele multi-procesor, mașinile virtuale Java pot rula firele de lucru pe procesoare diferite și pot obține astfel prelucrare paralelă adevărată și creșteri de performanțe semnificative față de platformele mono-procesor.

00P13 - M. Joldes - T.U. Cluj

22



Multitasking vs Multithreading



00P13 - M. Joldes - T.U. Cluj

21



Multithreading în Java

- Proprietățile firelor de lucru multiple din programele Java:
 - Fiecare fir de lucru își începe execuția la o locație bine cunoscută, predefinită
 - Fiecare fir de lucru își execută codul începând de la locația de start, într-o secvență ordonată, predefinită (pentru un set de date de intrare dat)
 - Fiecare fir de lucru își execută codul independent de celelalte fire de lucru din program
 - Firele de lucru par a avea un anumit grad de simultaneitate în execuție
 - Firele de lucru au acces la diferite tipuri de date

00P13 - M. Joldes - T.U. Cluj

23



Multithreading în Java

- Toate programele Java în afara aplicațiilor simple cu intrare-ieșire pe consolă sunt aplicații multithreading.
- Procesele *grele* (*heavyweight*) se rulează direct sub sistemul de operare al mașinii locale.
- **Fir de lucru:** un singur curs secvențial al controlului, numit și proces ușor (*lightweight*), lansat din procesul principal
- Sunt procese paralele care rulează *înăuntrul* unui program
- În Java se pot crea fire de lucru din program, așa cum se pot rula programe sub un sistem de operare
- Java: creează fire de lucru în două moduri:
 - Clasa extinde clasa `Thread` și îi suprascrie metoda `run()`.
 - Clasa implementează interfața `Runnable`, care are o singură metodă: `run()`.
- Clasa îi transmite o referință la sine însuși când creează un fir de lucru
- Apoi firul apelează înapoi metoda `run()` din clasă.

00P13 - M. Joldes - T.U. Cluj

24



Sub-clasarea clasei Thread

- Metoda `run()` din fir corespunde metodei `main()` pentru o aplicație.
- La pornirea firului, acesta invocă metoda `run()`, iar atunci când procesul revine din metoda `run()` firul de lucru *moare*.
 - Nu se poate "învia" un fir de lucru mort. În loc de "înviere", trebuie creată o nouă instanță de fir de lucru.
- Subclasa trebuie să suprascrie metoda `run()`.
- În metoda `run()` se pune codul care trebuie executat în paralel cu programul principal

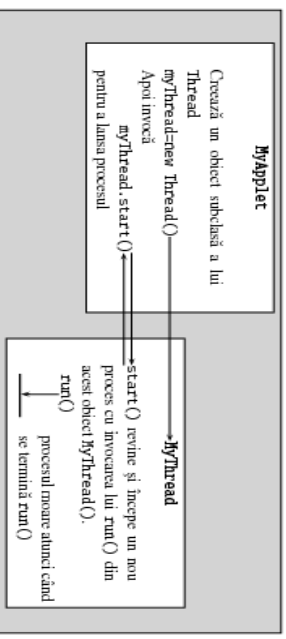
00P13 - M. Joldos - T.U. Cluj

25



Exemplu de subclasare a clasei Thread

- Demo: subclass



00P13 - M. Joldos - T.U. Cluj

26



Implementarea interfeței Runnable

- Implementăm interfața `Runnable` și îi suprascriem metoda `run()`
- Transmitem o referință la o instanță a respectivei implementări a `Runnable` spre o instanță de fir de lucru și firul de lucru *apelează înapoi (calls back)* metoda `run()` din obiectul `Runnable`.
- Firul de lucru moare, ca la procedeuul anterior, la revenirea procesului din `run()`.
- Procedeuul este convenabil în cazurile în care dorim să creăm un singur tip de fir de lucru, cum este, spre exemplu, *animația dintr-un applet*.

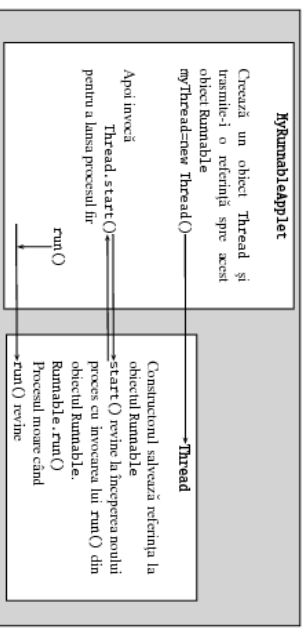
00P13 - M. Joldos - T.U. Cluj

27



Exemplu de implementare a interfeței Runnable

- Demo: runnable



00P13 - M. Joldos - T.U. Cluj

28



Interfața Runnable: Schiță de implementare

```
public class ClaasToRun extends SomeClass implements
Runnable
{
    ...
    public void run()
    {
        // Completati aici dacă ClaasToRun
        // a fost derivată din Thread
    }
    public void startThread()
    {
        Thread t=Thread( this );
        t.start();
    }
    ...
}
```

00P13 - M. Joldos - T.U. Cluj

29



Sub-clasarea vs. Runnable

- Tehnica folosind `Runnable` este în special convenabilă dacă dorim să creăm un singur fir care să efectueze o anumită sarcină de lucru.
 - Metoda `run()` va avea acces la variabilele instanță din obiectul `Runnable`.
 - D.e., pentru animație în applet, faceți applet-ul `Runnable`.
 - Metoda `run()` are atunci acces la variabilele applet-ului, care pot fi parametri transmiși prin etichetele applet-ului sau stabilități de utilizator prin intermediul interfeței grație metodei `startThread()`.
- Dacă doriți sa creați mai multe fire de lucru, atunci de obicei e mai bine să derivați din clasa `Thread`.
 - Acasta ajută la conceptualizarea mai bună a firilor ca obiecte independente
 - Puteți seta valorile oricăror parametri de care ar fi nevoie folosind constructorii sau metode "setter" (de setare)

00P13 - M. Joldos - T.U. Cluj

30



Stoparea/Punerea în pauză a unui fir de lucru

- Un fir se oprește în trei moduri:
 - Dacă revine normal din `run()`. [Cea mai bună cale]
 - Se apelează metoda `stop()` a firului. (Acum depreciată (**deprecated**). Nu o folosiți.)
 - Dacă este întrerupt de o excepție neinterceptată.
- Dacă metoda `run()` conține o buclă de durată sau una infinită – opriți buclarea atunci când se modifică o variabilă care poate fi modificată de procesul principal. De- o valoare `boolean` setată la `false` sau o variabilă referință setată la `null` pentru oprire
- Opriți explicit întotdeauna firele din `applet-uri` la apelul metodei `stop()` a `applet-ului`.
 - În caz contrar, firele pot continua să ruleze chiar dacă browserul încarcă o noua pagină de Web

00P13 - M. Joldes - TLU Cluj

31



Stoparea/Punerea în pauză a unui fir de lucru

- Pentru cazul `Runnable`, puteți porni un nou fir cu setările variabilelor la valorile pe care le-au avut în momentul opririi firului precedent.
- În acest caz, oprirea unui fir și pornirea unuia nou acționează ca acțiuni *pauză/start*.
- Metodele clasei `Thread` *suspend()* și *resume()* au fost depreciate (**deprecated**) pentru a evita problemele de interblocare a firelor de lucru

00P13 - M. Joldes - TLU Cluj

32



Thread.sleep

- `Thread.sleep` este o metodă statică din clasa `Thread` care pune în pauza un fir de lucru care conține invocarea
 - Pune în pauză firul timp de numărul de milisecunde dat ca argument
 - Remarcați că metoda poate fi invocată dintr-un program obișnuit pentru a insera o pauză în singurul fir al programului respectiv
- Metoda poate arunca o excepție verificată, `InterruptedException`, care trebuie declarată sau interceptată
 - Atât clasa `Thread` cât și `InterruptedException` se află în pachetul `java.lang`

00P13 - M. Joldes - TLU Cluj

33



Stările firelor de lucru

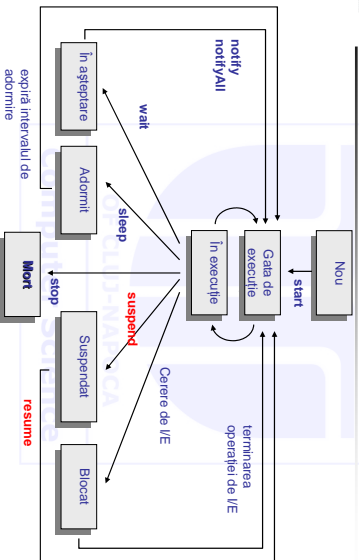
- `getState()` : (Java 5) metoda întoarce un `Enum` de `Thread.States`. Stări:
 - `NEW` – fir "proaspăt" care nu și-a început încă execuția.
 - `RUNNABLE` – fir în curs de execuție în JVM.
 - `BLOCKED` – fir blocat care așteaptă după un zăvor monitor.
 - `WAITING` – fir care așteaptă să fie notificat de un alt fir.
 - `TIMED_WAITING` – fir care așteaptă un anumit timp să fie notificat de un alt fir.
 - `TERMINATED` – fir a cărui metodă `run` s-a terminat.

00P13 - M. Joldes - TLU Cluj

34



Ciclu de viață al unui fir de lucru



00P13 - M. Joldes - TLU Cluj

35



Planificatorul de fire

- Planificatorul de fire execută fiecare fir de lucru pentru un interval de timp scurt (o *felie de timp* (*time slice*))
- Apoi planificatorul activează un alt fir
- Întotdeauna vor fi variații ale timpilor de rulare, în special la apelul serviciilor sistemului de operare (d.e. pentru operații de intrare și de ieșire)
- Nu sunt garanții referitoare la ordinea de execuție a firelor

00P13 - M. Joldes - TLU Cluj

36



Terminarea firelor de lucru

- Un fir de lucru se termină la terminarea metodei sale `run()`
- Nu terminați** un fir de lucru folosind metoda depreciată **`stop`**
- În loc de aceasta, notificați firul de lucru că ar trebui să se termine folosind

```
t.interrupt();
```

- `interrupt()` nu face ca firul să se termine – metoda setează un câmp boolean în structura de date a firului



Terminarea firelor de lucru

- Metoda `sleep` aruncă o excepție de tipul `InterruptedException` atunci când un fir "adormit" este întrerupt

- Interceptează excepția
- Termină firul de lucru

```
try {
    for (int i = 1; i <= REPEATITIONS; i++)
    {
        Do work
    }
} catch (InterruptedException exception) {
    Clean up
}
```



Animatii

- Animatiile sunt o sarcină uzuală pentru firele de lucru: ele sunt folosite la controlul animației

- Un fir de lucru poate dirija desenarea fiecărui cadru în timp ce alte aspecte, cum sunt
- Răspunsul la interacțiunea cu utilizatorul poate continua în paralel

- Demo: `clock`, `drop2d`, `sunsort`



Terminarea firelor de lucru

- Metoda `run()` ar trebui să verifice ocazional dacă firul de lucru a fost întrerupt

- Folosiți metoda `interrupted()`
- Un fir de lucru care a fost întrerupt ar trebui să elibereze resursele pe care le folosește, să "curețe" și să se termine

```
public void run()
{
    for (int i = 1;
        i <= REPEATITIONS && !Thread.interrupted(); i++)
    {
        Do work
    }
    Clean up
}
```



Terminarea firelor de lucru

- Java nu forțează terminarea unui fir atunci când acesta este întrerupt
- Este treaba firului de lucru ce anume face atunci când este întrerupt
- Întreruperea reprezintă un mecanism general pentru a obține "atenția" firului de lucru întrerupt



Rezumat

- Mai mult despre noua IE Java
 - Buffer (zonă tampon)
 - Vederi
 - Endian-ness
 - Direct ~
 - Memory mapped ~
 - Canale la fișiere
 - Fișiere mapate pe memorie
 - Transferuri directe
 - Zăvorărea
 - Clasa `string` – expresii regulate
- Introducere la firele de lucru Java
 - Multitasking vs. multithreading
 - Crearea firelor de lucru:
 - Sub-clasarea `Thread`
 - Implementarea `Runnable`
 - Stoparea/punerea în pauză a firelor
 - Terminarea firelor
 - Animatie simplă cu fire de lucru