IMPLEMENTAREA CONCURENTEI IN LIMBAJE DE PROGRAMARE

CONCURENTA IN JAVA

Ioana Leustean

➢ Sincronizarea thread-urilor

- **CountDownLatch**: sincronizarea thread-urilor care au executat anumite task-uri; la creare se transimte un contor (numarul de taskuri), iar thread-urile care apeleaza **latch.await()** sunt blocate pana cand valoarea contorului devine 0; fiecare thread care executa un task decrementeaza contorul apeland **latch.countDown()**;

- **CyclicBarrier**: sincronizarea thread-urilor care asteapta la bariera respective; thread-urile care apeleaza barrier.await() poate trece mai departe numai dupa ce toate au ajuns in punctul respectiv.

➢ Colectii concurente: asigura concurenta si sincronizare, fara a fi blocate de un singur lacat

- **BlockingQueue -** folosite in special pentru modelul **producator-consummator**

- **ConcurrentHashMap –** permite citiri concurente dar si un numar de scrieri concurente

java.util.concurrent (Java SE 23 & JDK 23)

➢ public class ConcurrentHashMap<K,V>

- colectia este impartita in fragmente care sunt prelucrate in parallel
- colectia poate fi create cu un anume nivel de concurenta; numarul de fragmente prelucrate in paralel este dat de nivelul de concurenta
- cand un thread executa o operatie care blocheaza, este blocat numai fragmentul corespunzator
- actualizarile sunt operatii care blocheaza, regasirile nu blocheaza; este regasita ultima valoare modificata de o actualizare care s-a finalizat
- actiunile dintr-un thread care plaseaza un obiect in colectie sunt in relatie happens-before cu actiunile dintr-un alt thread care urmeaza accesarii/stergerii elementului din colectie
- colectia nu este ordonata, elementele pot fi procesate in paralel in ordini diferite
- piscina de thread-uri este creata cu ForkJoinPool.commonPool()

https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html

```
public class ConcHM{

private static volatile boolean ok=true;

public static void main(String[] args) throws Exception {
          ConcurrentHashMap<Integer, String> map = new ConcurrentHashMap<>();
          CountDownLatch controller = new CountDownLatch(4);
          (new WriterThread(map, "Writer-1",controller)).start();
          (new WriterThread(map, "Writer-2",controller)).start();
          (new ReaderThread(map, "Reader-1")).start();
          (new ReaderThread(map, "Reader-2")).start();
          (new ReaderThread(map, "Reader-3")).start();
          controller.await();
          ok=false;
          }
private static class WriterThread extends Thread {…}
private static class ReaderThread extends Thread {…}

}
```

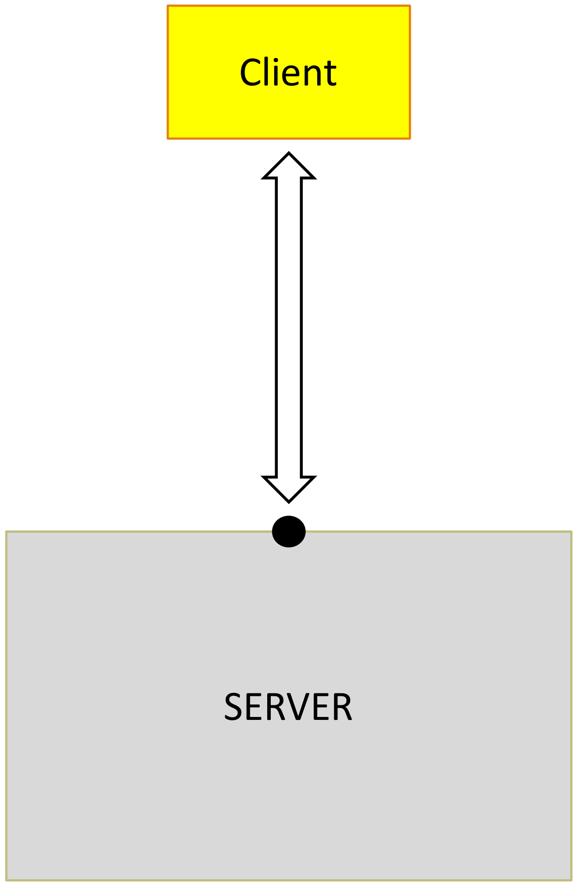
```
private static class ReaderThread extends Thread {
        private ConcurrentHashMap<Integer, String> map;
        private String name;
        public ReaderThread(ConcurrentHashMap<Integer, String> map, String threadName) {
        this.map= map; this.name = threadName;}
        public void run() {
        while (ok) {
        long time = System.currentTimeMillis(); String output = time + ": " + name + ": ";
        for (Integer key : map.keySet()) {
        String value = map.get(key);
        output += "("+key + "," + value + ") ";
        }
        System.out.println(output);
        try { Thread.sleep(ThreadLocalRandom.current().nextInt(300));} catch (InterruptedException ex) {}
        }}
        }
```

```
private static class WriterThread extends Thread {
    private ConcurrentMap<Integer, String> map;
    private CountDownLatch controller;
    private String name;
    public WriterThread(ConcurrentMap<Integer, String> map, String threadName, CountDownLatch c) {
    this.map= map; this.name = threadName; this.controller=c;}
    public void run() {
    String value = name;
    while (ok) {
    for (int i = 0; i < 5; i++) {
    Integer key = ThreadLocalRandom.current().nextInt(100);
    if(map.putIfAbsent(key, value) == null) {
    System.out.println(System.currentTimeMillis()+":"+name+" has put["+key+"=>"+value+"]") ;}
    else {
        System.out.println(System.currentTimeMillis()+":"+name+" duplicate") ;}}
        controller.countDown();
    try { Thread.sleep(ThreadLocalRandom.current().nextInt(500));} catch (InterruptedException ex) {}

    }}}
```

```
> java ConcHM
1742358981486:Writer-2 has put[67=>Writer-2]
1742358981490:Writer-1 has put[0=>Writer-1]
1742358981532:Writer-1 has put[4=>Writer-1]
1742358981532:Writer-2 has put[77=>Writer-2]
1742358981532:Writer-1 has put[5=>Writer-1]
1742358981533:Writer-2 has put[48=>Writer-2]
1742358981533:Writer-1 has put[13=>Writer-1]
1742358981533:Writer-2 has put[24=>Writer-2]
1742358981534:Writer-2 has put[32=>Writer-2]
1742358981534:Writer-1 duplicate
1742358981487: Reader-1: (0,Writer-1) (67,Writer-2) (4,Writer-1) (5,Writer-1) (24,Writer-2) (77,Writer-2) (13,Writer-1)
1742358981487: Reader-3: (0,Writer-1) (67,Writer-2) (4,Writer-1) (5,Writer-1) (24,Writer-2) (77,Writer-2) (13,Writer-1)
1742358981487: Reader-2: (0,Writer-1) (67,Writer-2) (4,Writer-1) (5,Writer-1) (24,Writer-2) (77,Writer-2) (13,Writer-1)
1742358981567: Reader-1: (0,Writer-1) (48,Writer-2) (32,Writer-2) (67,Writer-2) (4,Writer-1) (5,Writer-1) (24,Writer-2)
 (77,Writer-2) (13,Writer-1)
1742358981671: Reader-3: (0,Writer-1) (48,Writer-2) (32,Writer-2) (67,Writer-2) (4,Writer-1) (5,Writer-1) (24,Writer-2)
 (77,Writer-2) (13,Writer-1)
```

```
1742358981720:Writer-2 has put[14=>Writer-2]
1742358981720:Writer-2 has put[33=>Writer-2]
1742358981721:Writer-2 has put[66=>Writer-2]
1742358981721:Writer-2 has put[93=>Writer-2]
1742358981722:Writer-2 duplicate
1742358981735: Reader-1: (0,Writer-1) (32,Writer-2) (33,Writer-2) (66,Writer-2) (67,Writer-2) (4,Writer-1) (5,Writer-1)
(77,Writer-2) (13,Writer-1) (14,Writer-2) (48,Writer-2) (24,Writer-2) (93,Writer-2)
1742358981791:Writer-1 duplicate
1742358981791:Writer-1 has put[2=>Writer-1]
1742358981791:Writer-1 has put[80=>Writer-1]
1742358981792:Writer-1 has put[28=>Writer-1]
1742358981792:Writer-1 has put[54=>Writer-1]
>
```


Client
SERVER

```
public class ServerMT implements Runnable {
private Socket clientSocket;

public static void main(String args[])
   {  ServerSocket serverSocket = new ServerSocket(9090);
      System.out.println("Server is running");
      while (true){
      Socket cSocket = serverSocket.accept();
      System.out.println("Client connected!");
      Thread clientThread = new Thread (new ServerMT (cSocket));
      clientThread.start(); }}

 public ServerMT(Socket s){this.clientSocket =s;}

 public void run()  {
      BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(
      PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
       String message = in.readLine();  System.out.println("From client: " + message);
       while (! message.equals("bye")) {
out.println("Message received!");  message = in.readLine(); System.out.println("From client: " + mes
 }
      clientSocket.close();}}
```

```
>java Client
hi!
From server: Message received!
hi!
From server: Message received!
bye
>
```

```
>java Client
buna!
From server: Messag
buna!
From server: Message received!
buna!
From server: Message received!
bye
>
```

```
a Server
er is running.
Client connected!
From client: buna!
From client: buna!
Client connected!
From client: hi!
From client: buna!
From client: hi!
From client: bye
From client: bye
```

Clientii sunt serviti **concurent**!

Server multithreaded

```
public class ServerMT implements Runnable {
private Socket clientSocket;
public static void main(String args[])
   {  ServerSocket serverSocket = new ServerSocket(9090);
      System.out.println("Server is running");
      ExecutorService  pool = Executors.newCachedThreadPool();

      while (true){
      Socket cSocket = serverSocket.accept();
      System.out.println("Client connected!");
      pool.execute(new ServerMT (cSocket)); }}

 public ServerMT(Socket s){this.clientSocket =s;}

 public void run()  {
      BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
      PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
       String message = in.readLine();  System.out.println("From client: " + message);
       while (! message.equals("bye")) {
        out.println("Message received!");  message = in.readLine(); System.out.println("From client: " + message);}
       clientSocket.close();}}
```

am creat o piscina **de thread-uri**

JEP 444: Virtual Threads

The thread-per-request style
Server applications generally handle concurrent user requests that are independent of each other, so it makes sense for an application to handle a request by dedicating a thread to that request for its entire duration. This thread-per-request style is easy to understand, easy to program, and easy to debug and profile because it uses the platform's unit of concurrency to represent the application's unit of concurrency. […]Unfortunately, the number of available threads is limited because the JDK implements threads as wrappers around operating system (OS) threads.

Improving scalability with the asynchronous style
Some developers wishing to utilize hardware to its fullest have given up the thread-per-request style in favor of a thread-sharing style. In the asynchronous style, each stage of a request might execute on a different thread, and every thread runs stages belonging to different requests in an interleaved fashion

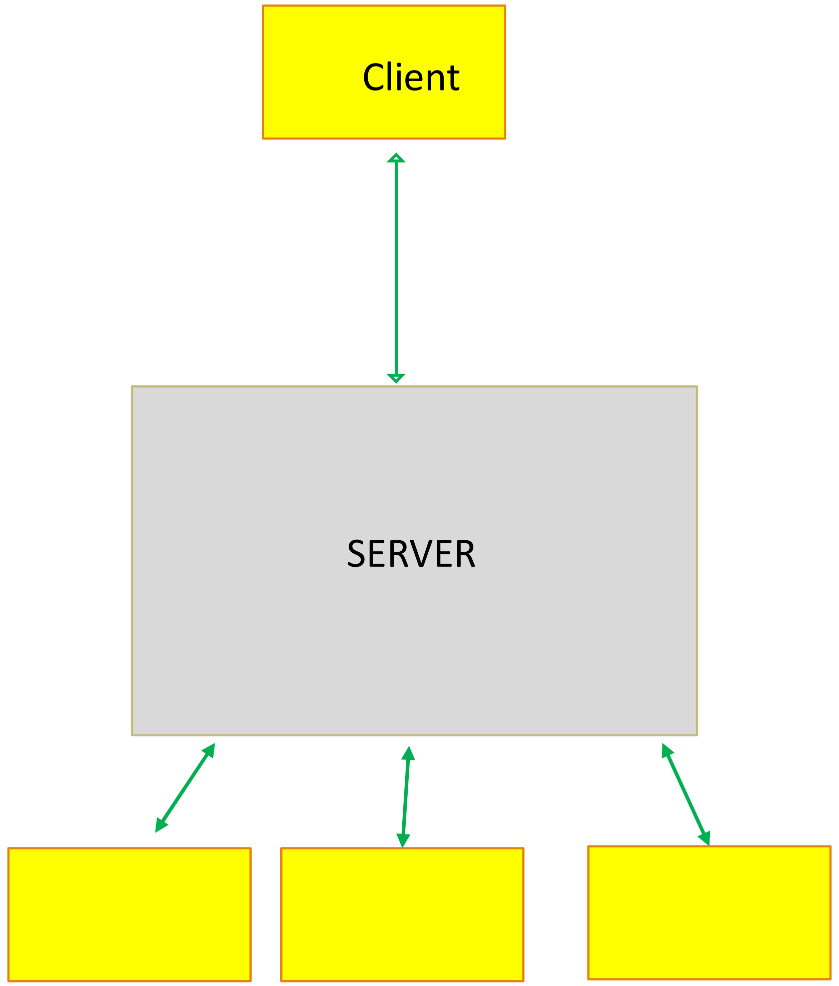
➢ NIO (New Input/Output)

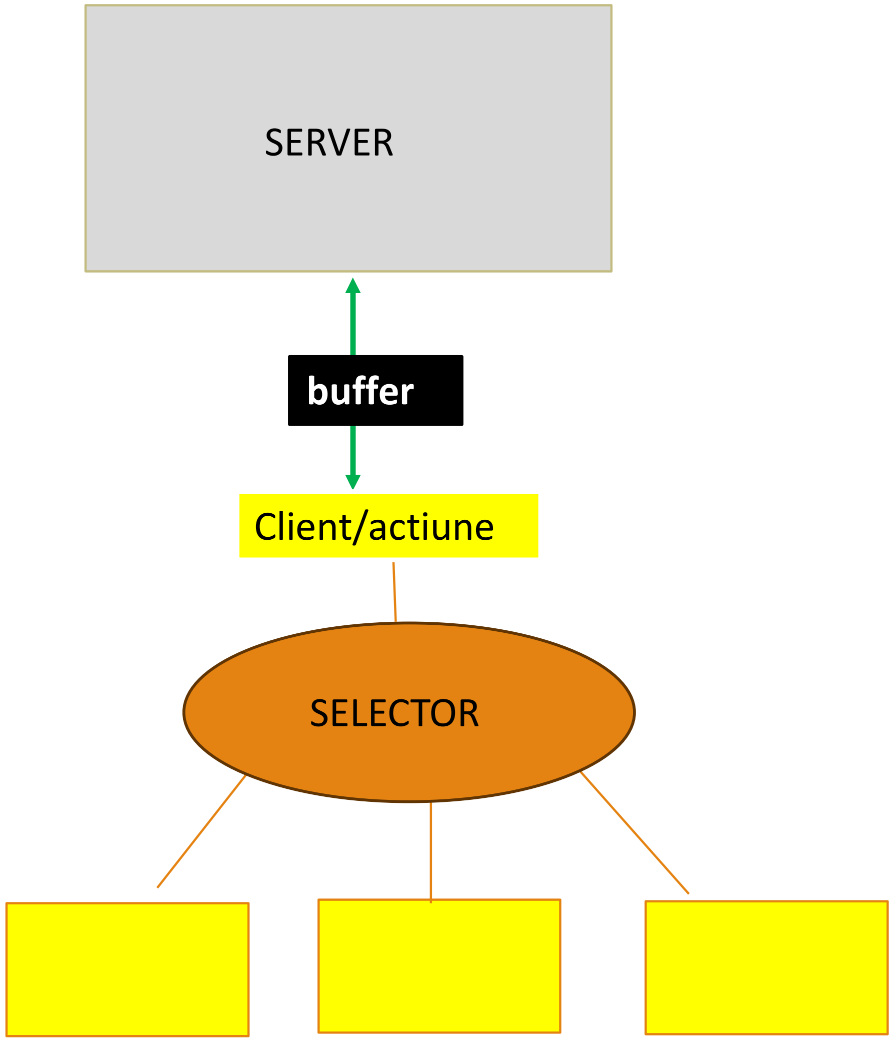
- Bufferele sunt blocuri de memorie in care datele sunt scrise pentru a fi prelucrate ulterior. Operatiile de citire/scriere a bufferelor se fac fara blocarea thread-urilor (spre deosebire de operatiile de citire/scriere a stream-urilor).

- Canale sunt conexiuni intre buffere si entitati care efectueaza operatiile de intrare/iesire (fisiere sau socket-uri): datele sunt scrise de pe canal intr-un buffer sau citite dintr-un buffer pe un canal. **SocketChannel** sunt conectate la un socket iar **ServerSocketChannel** poate accepta conexiuni (TCP)

- Selectori care realizeaza multiplexarea operatiilor de intrare-iesire de pe canale selectabile. Astfel, un sigur thread poate servi mai multe canale. Pentru a fi procesate cu ajutorul unui selector canelele sunt inregistrate de selectorul respectiv, iar metoda **select()** a selectorului blocheaza thread-ul pana cand cel putin un canal inregistrat este gata pentru prelucrare.

java.nio (Java SE 24 & JDK 24)


Client
SERVER


SERVER
buffer
Client/actiune
SELECTOR

➢ **Server asincron NIO (un singur thread)**

```
public class NioServer {
  public static void main(String[] args) {
    try{
    Selector selector = Selector.open();
    ServerSocketChannel serverSocket =   ServerSocketChannel.open();
    serverSocket.bind(new InetSocketAddress("localhost", 9090));
    serverSocket.configureBlocking(false);
    serverSocket.register(selector, SelectionKey.OP_ACCEPT);
    System.out.println("Server looking for clients ...");
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    while (true) {
      selector.select();
      Set<SelectionKey> selectedKeys = selector.selectedKeys();
      for (SelectionKey key : selectedKeys) {
                            …
                        } }
     } catch (IOException e) {
        throw new RuntimeException(e);
     }   }}
```

➢ Server asincron NIO (un singur thread)

```
public class NioServer {
  public static void main(String[] args) {
    try{
    Selector selector = Selector.open();
    ServerSocketChannel serverSocket =   ServerSocketChannel.open(
    serverSocket.bind(new InetSocketAddress("localhost", 9090));
    serverSocket.configureBlocking(false);
    serverSocket.register(selector, SelectionKey.OP_ACCEPT);
    System.out.println("Server looking for clients ...");
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    while (true) {
      selector.select();
      Set<SelectionKey> selectedKeys = selector.selectedKeys();
      for (SelectionKey key : selectedKeys) {
                              …
                          } }
     } catch (IOException e) {
        throw new RuntimeException(e);
     }  }}
```

```
if (key.isAcceptable()) {
            SocketChannel client = serverSocket.accept();
            client.configureBlocking(false);
            client.register(selector, SelectionKey.OP_READ);
          }
          else if (key.isReadable()) {
            SocketChannel client = (SocketChannel) key.channel();
            buffer.clear();
            var bytesRead = client.read(buffer);
            if (bytesRead == -1) { client.close();
              System.out.println("Client disconnected!");}
            else {String result = new String(buffer.array()).trim();
            System.out.println("Message received: " + result);}
                }
          else {
            throw new RuntimeException("Unknown channel");
          }
          selectedKeys.remove(key);
```

➢ Client - Server asincron NIO (un singur thread)

```
public class NioClient{
    public static void main(String[] args) {
        var scanner = new Scanner(System.in);
        try (var serverChannel = SocketChannel.open()) {
            serverChannel.connect(new InetSocketAddress(9090));
            serverChannel.configureBlocking(false);
            System.out.println("Connection established! Stop with \"quit\" ");
           var buffer = ByteBuffer.allocate(1024);
            while (true) {
                var line = scanner.nextLine();
                if (line.equalsIgnoreCase("quit")) {
                    break;
                }
                 buffer.clear().put(line.getBytes()).flip();
                serverChannel.write(buffer);
}
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- **clear()** makes a buffer ready for a new sequence of channel-read or relative put operations:
  it sets the limit to the capacity and the position to zero.

- **flip()** makes a buffer ready for a new sequence of channel-write or relative get operations: it sets the limit to the current position and then sets the position to zero.

➢ Client - Server asincron NIO (un singur thread)

```
public class NioServer {
  public static void main(String[] args) {
    try{
    Selector selector = Selector.open();
    ServerSocketChannel serverSocket =   ServerSocketChanne
    serverSocket.bind(new InetSocketAddress("localhost", 909
    serverSocket.configureBlocking(false);
    serverSocket.register(selector, SelectionKey.OP_ACCEPT);
    System.out.println("Server looking for clients ...");
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    while (true) {
      selector.select();
      Set<SelectionKey> selectedKeys = selector.selectedKeys()
      for (SelectionKey key : selectedKeys) {
                                ...
                             } }
      } catch (IOException e) {
          throw new RuntimeException(e);
      }   }}
```

```
java NioClient
Connection established! Stop
with "quit"
c2m1
c2m2
c2m3
quit
```

```
java NioServer
Server looking for clients ...
Message received: c1m1
Message received: c1m2
Message received: c2m1
Message received: c2m2
Message received: c2m3
Client disconnected!
Client disconnected!
|
```

```
java NioClient
Connection established! Stop
with "quit"
c1m1
c1m2
quit
```

➢ Server canale asincrone NIO2

```
public class Nio2Server{
  public static void main(String[] args) throws Exception {
    AsynchronousServerSocketChannel server  = AsynchronousServerSocketChannel.open();
    server.bind(new InetSocketAddress("localhost", 9090));
      while (true) {
       Future<AsynchronousSocketChannel> acceptFuture = server.accept();
       AsynchronousSocketChannel clientChannel = acceptFuture.get();
       while ((clientChannel != null) && (clientChannel.isOpen())) {
          ByteBuffer buffer = ByteBuffer.allocate(32);
          Future<Integer> readResult  = clientChannel.read(buffer);

          readResult.get();

          buffer.flip();
          Future<Integer> writeResult = clientChannel.write(buffer);

          writeResult.get();
          buffer.clear();
      }
      clientChannel.close();}}}
```

canale care permit operatii de intrare/iesire asincrone.
[AsynchronousChannel (Java SE 24 & JDK 24)]

➢ **Client - Server asincron NIO (un singur thread)**

```
public class Nio2Client {
   public static void main(String[] args) throws Exception {

AsynchronousSocketChannel client = AsynchronousSocketChannel.open();
InetSocketAddress hostAddress = new InetSocketAddress("localhost", 9090);
Future<Void> future = client.connect(hostAddress);    // poate executa alte actiuni
future.get();
System.out.println("Connection established! Stop with \"quit\" ");
var scanner = new Scanner(System.in);
while (true){
var line = scanner.nextLine();
if (line.equalsIgnoreCase("quit")) { break;}
   byte[] byteMsg = new String(line).getBytes();
   ByteBuffer buffer = ByteBuffer.wrap(byteMsg);
   Future<Integer>  writeResult = client.write(buffer);     // poate executa alte actiuni
   writeResult.get();
   buffer.flip();
   Future<Integer>  readResult = client.read(buffer);   // poate executa alte actiuni
   readResult.get();
   String message = new String(buffer.array()).trim();  buffer.clear();System.out.println(message);}
   client.close();}}
```

➢ Server canale asincrone NIO2

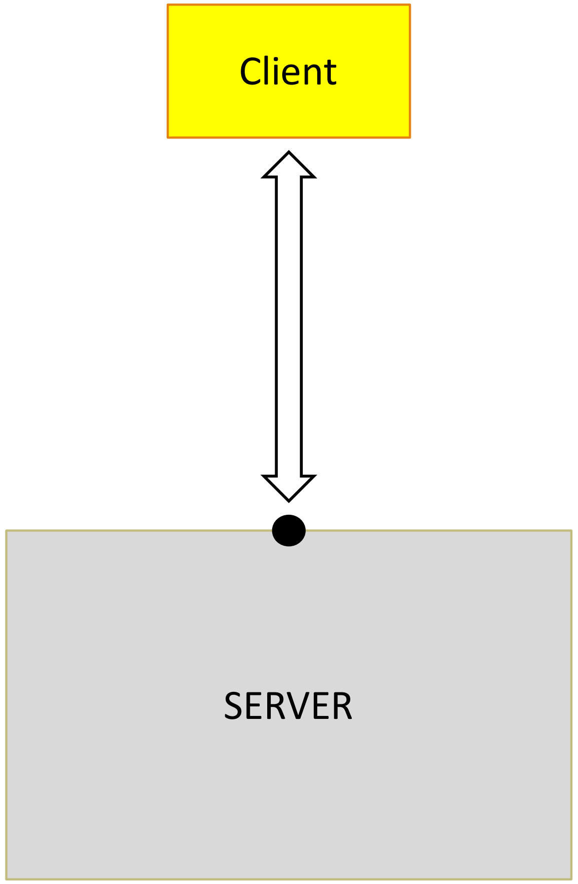
```
public class Nio2Server{
  public static void main(String[] args) throws Exception {
    AsynchronousServerSocketChannel server  = AsynchronousServerSocketChannel.open();
    server.bind(new InetSocketAddress("localhost", 9090));
    ExecutorService  pool = Executors.newCachedThreadPool();
     while (true) {
      Future<AsynchronousSocketChannel> acceptFuture = server.accept();
      AsynchronousSocketChannel clientChannel = acceptFuture.get();
      pool.execute( new Runnable(){public void run() {
        try{
      while ((clientChannel != null) && (clientChannel.isOpen())) {
        ByteBuffer buffer = ByteBuffer.allocate(32);
        Future<Integer> readResult  = clientChannel.read(buffer);

        readResult.get();

        buffer.flip();
        Future<Integer> writeResult

        writeResult.get();
        buffer.clear();
    }
```

Asynchronous channels are safe for use by multiple concurrent threads. Some channel implementations may support concurrent reading and writing
[AsynchronousChannel (Java SE 24 & JDK 24)]


Client
SERVER

Server multithreaded

```
public class ServerMT implements Runnable {
private Socket clientSocket;
public static void main(String args[])
   {  ServerSocket serverSocket = new ServerSocket(9090);
      System.out.println("Server is running");
      ExecutorService  pool = Executors.newCachedThreadPool();

      while (true){
      Socket cSocket = serverSocket.accept();
      System.out.println("Client connected!");
      pool.execute(new ServerMT (cSocket)); }}

 public ServerMT(Socket s){this.clientSocket =s;}

  public void run()  {
      BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
      PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
       String message = in.readLine();  System.out.println("From client: " + message);
       while (! message.equals("bye")) {
        out.println("Message received!");  message = in.readLine(); System.out.println("From client: " + message);}
       clientSocket.close();}}
```

am creat o piscina **de thread-uri**

JEP 444: Virtual Threads

The thread-per-request style
Server applications generally handle concurrent user requests that are independent of each other, so it makes sense for an application to handle a request by dedicating a thread to that request for its entire duration. This thread-per-request style is easy to understand, easy to program, and easy to debug and profile because it uses the platform's unit of concurrency to represent the application's unit of concurrency. […]Unfortunately, the number of available threads is limited because the JDK implements threads as wrappers around operating system (OS) threads.

Improving scalability with the asynchronous style
Some developers wishing to utilize hardware to its fullest have given up the thread-per-request style in favor of a thread-sharing style. In the asynchronous style, each stage of a request might execute on a different thread, and every thread runs stages belonging to different requests in an interleaved fashion. […] *This programming style is at odds with the Java Platform because the application's unit of concurrency — the asynchronous pipeline — is no longer the platform's unit of concurrency.*

Preserving the thread-per-request style with virtual threads
To enable applications to scale while remaining harmonious with the platform, we should strive to preserve the thread-per-request style. We can do this by implementing threads more efficiently, so they can be more plentiful. Operating systems cannot implement OS threads more efficiently because different languages and runtimes use the thread stack in different ways. It is possible, however, for a Java runtime to implement Java threads in a way that severs their one-to-one correspondence to OS threads. Just as operating systems give the illusion of plentiful memory by mapping a large virtual address space to a limited amount of physical RAM, a Java runtime can give the illusion of plentiful threads by mapping a large number of virtual threads to a small number of OS threads.

➢ Server multithreaded cu thread-uri virtuale

```
public class ServerMTv implements Runnable {
private Socket clientSocket;

public static void main(String args[])
   {  ServerSocket serverSocket = new ServerSocket(9090);
       System.out.println("Server is running");
       ExecutorService  pool = Executors.newVirtualThreadPerTaskExecutor();
       while (true){
       Socket cSocket = serverSocket.accept();
       System.out.println("Client connected!");
       pool.execute(new ServerMT (cSocket)); }}

 public ServerMTv(Socket s){this.clientSocket =s;}

  public void run()  {
       BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
       PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        String message = in.readLine();  System.out.println("From client: " + message);
        while (! message.equals("bye")) {
         out.println("Message received!");  message = in.readLine(); System.out.println("From client: " + message);}
        clientSocket.close();}}
```

am creat o piscina **de thread-uri virtuale**