

Cap. IV - Programação Concorrente

[**Magee 1999**] *Concurrency – State Models and Java Programs*, *Jeff Magee, Jeff Kramer*, John Wiley 1999.

[**Gosling**] “The Java Language Specification” James Gosling, Bill Joy and Guy Steele, Addison-Wesley. <http://java.sun.com/docs/books/jls/index.html>

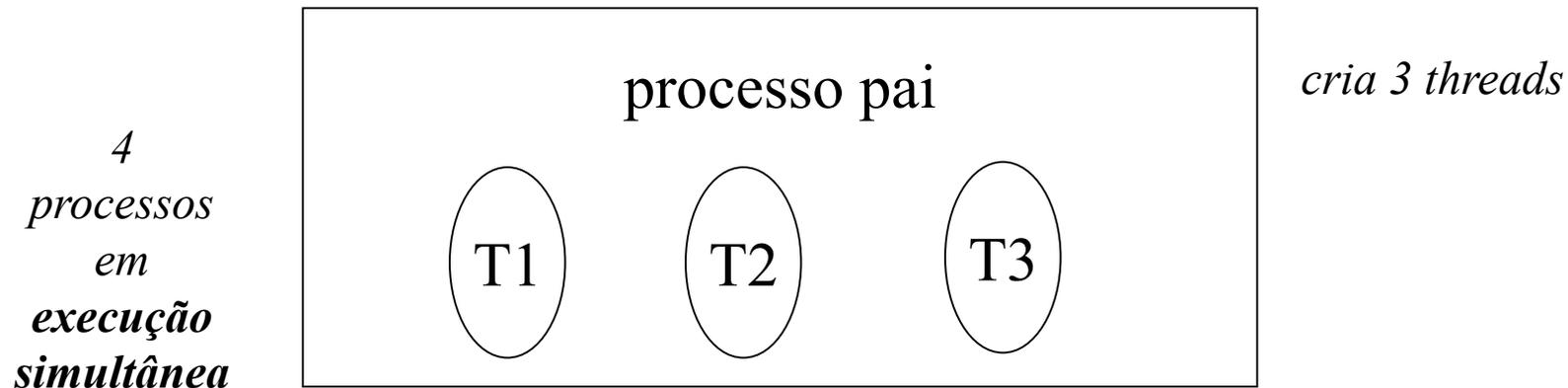
[**Berg99**] “Advanced Techniques for Java Developers” *Daniel J. Berg and J. Steven Fritzinger*, Jhon Willey & Sons, 1999.

Programação Concorrente

Definição:

“Thread” – sequência de execução independente

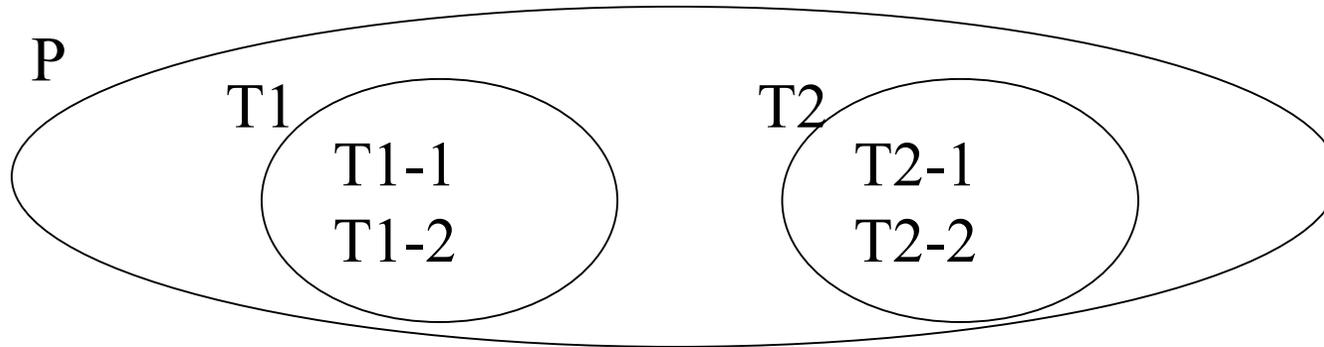
As várias threads de um processo partilham o mesmo espaço de endereçamento que o processo (pai) que lhe deu origem.



Programação Concorrente

Seja um processo P constituído pelas threads T1 e T2.

A execução de P é uma sequência pertencente ao conjunto formado por todas as sequências de combinações possíveis de execução de T1 e T2



Programação Concorrente

Sequências de execução possíveis:

S1: T1-1, T1-2, T2-1, T2-2

S2: T2-1, T2-2, T1-1, T1-2

S3 IT1-1, T2-1, T1-2, T2-2

S4: T1-1, T2-1, T2-2, T1-2

S5: T2-1, T1-1, T1-2, T2-2

S6: T2-1, T1-1, T2-2, T1-2

Programação Concorrente

Criação de Threads em Java

Hipótese 1:

subclasse de Thread

```
public class MinhaThread_1 extends Thread {
```

```
    public MinhaThread_1() {  
        super(); //construtor da superclasse
```

```
        start();
```

```
    }
```

*método definido na classe Thread, invoca o método **run()***

Programação concorrente em Java

Ex.lo

```
public void run(){
    while (true) {
        ....
        if (isInterrupted() )
            break;
    }
} // run
} //classe MinhaThread_1

public class Teste{
    public static void main (String [] args){
        MinhaThread_1 T1 = new MinhaThread_1();
    }
}
```

define o código a ser executado pela Thread

Programação concorrente em Java

Hipótese 2:

```
public class MinhaThread_2 extends Thread {  
    public void run () {  
        ...  
    }  
}  
                                     ? onde está a chamada ao super() ?  
public class Teste {  
    public static void main (String [] args) {  
        MinhaThread_2 Ta, Tb;  
        Ta = new MinhaThread_2();  
        Tb = new MinhaThread_2();  
        Ta.start();  
        Tb.start(); ← iniciar a execução da thread na classe Teste:  
    }  
}
```

Programação concorrente em Java

Se quisermos aceder a variáveis do processo principal?

- Passamos como parâmetros as referências dessas variáveis (objetos)

```
public class MinhaThread extends Thread {  
    ObjectoPartilhado O;  
    public MinhaThread ( ObjectoPartilhado o ){  
        super();  
        O = o ;  
        start();  
    }  
    public void run(){  
        ...  
        O.... ← aceder ao objecto partilhado  
    }  
}
```

*endereço do
objecto partilhado*

Programação concorrente em Java

Hipótese 3: - se a classe já for subclasse de outra classe?

```
public class MinhaThread_3 implements Runnable {  
    public void run() {  
        ...  
    }  
}
```

classe que implementa o método run()

```
public class Teste {  
    public static void main (String [] args) {  
        MinhaThread_3 Tc;  
        Tc = new MinhaThread_3();  
        Thread T = new Thread (Tc);  
        T.start();  
    }  
}
```

iniciar a execução

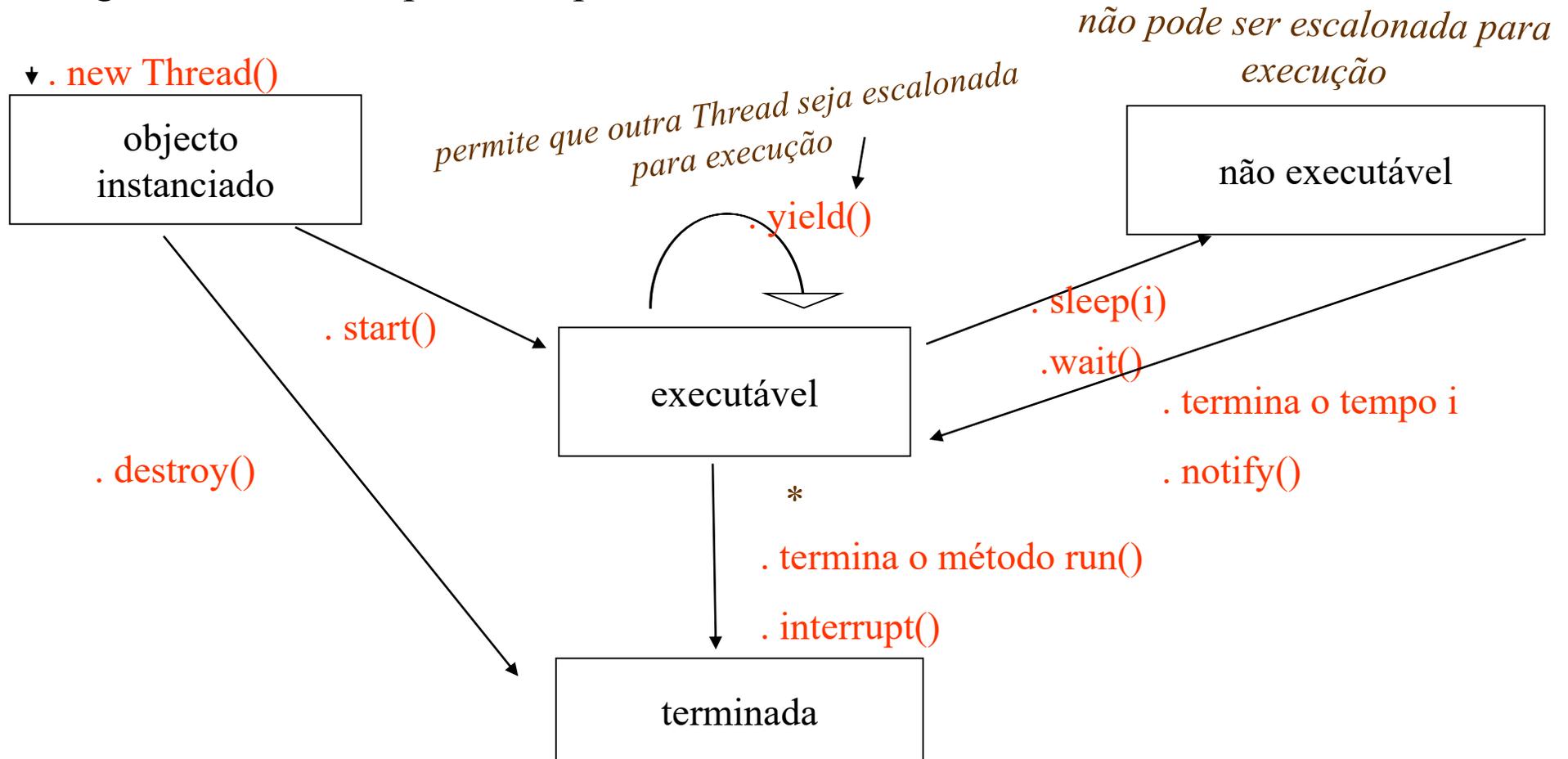
“runnable object”

```
public interface Runnable {  
    public abstract void run();  
}
```

Instanciar um objeto do tipo Thread,
passando como parâmetro um
“runnable” objeto.

Programação concorrente em Java

Diagrama de estados possíveis para uma Thread



* *uma Thread no estado executável, não está necessariamente em execução, apenas pode ser escalonada para execução*

Programação concorrente em Java

Exercício 1:

Pretende-se construir um programa que permita adicionar arrays de grande dimensão. Para isso, vamos dividir os arrays a somar em várias porções menores e somar cada porção numa thread independente.

a) Construir o código de uma thread, ThreadSoma, que receba como parâmetros as “referências” de 3 arrays, A, B e C e 2 valores inteiros, p e u. A thread deverá fazer a soma dos arrays, A e B, desde a posição p até à posição u-1, colocando o resultado em C.

$$C[i] = A[i] + B[i] \quad \text{com } i = p, p+1, p+2, \dots, u-1 .$$

Programação concorrente em Java

b) Construa um programa onde deverá criar dois arrays de inteiros com valores aleatórios. Os arrays, A e B, devem ter a mesma dimensão. Para somar os dois arrays, colocando a soma num array C com a mesma dimensão, divida-os em dois e cada metade deverá ser somada por uma instância da classe ThreadSoma que criou na alínea anterior.

c) Queremos agora que no programa anterior, o programa principal calcule a soma de todos os valores do array resultado, C. Essa soma só deve ser feita após ambas as instâncias de ThreadSoma terminarem a execução. Explore os métodos da classe Thread para ver como pode fazer com que uma Thread espere que outra termine a execução.

Programação concorrente em Java

Sincronização de Threads

. *mecanismo baseado no conceito de monitor*

Existe,

um **lock** associado a cada objecto *lock de objecto*

e

um lock associado a cada classe. *lock de classe*

A instrução

synchronized (expressão)

{ instruções }

referência para o objecto

Programação concorrente em Java

a) após calcular a referência para o objecto, mas antes de executar o corpo de instruções:

- A thread adquire o lock associado ao objecto,
(caso este não pertença já a alguma outra thread)
- executa o corpo de instruções

b) Após executar o corpo de instruções

- liberta o lock

(se a execução do bloco de instruções termina anormalmente i. é, falha a meio, o lock é libertado)

Programação concorrente em Java

Notas:

1. Um método pode ser declarado como `synchronized`,
(*comporta-se como se estivesse contido numa instrução `synchronized`*)
2. O facto de uma Thread adquirir o lock associado a um objecto, não impede que outras Threads acessem aos campos do objecto ou possam invocar métodos não sincronizados.
3. Se o método sincronizado é um método de instância:
 - a Thread adquire o lock do objecto (associado a `this`);
 - todas as Threads que tentem executar esse mesmo método, no mesmo objecto, terão que esperar, competindo pela aquisição do lock.

Notas (cont.):

4. Se o método sincronizado é um método de classe:

- a Thread adquire o lock da classe;
- todas as Threads que tentam executar esse método, em qualquer objecto da classe, terão que esperar que o lock seja libertado.

...

Programação concorrente em Java

Exemplo 1:

```
public class Exemplo {  
    private int x;  
    private static int s;  
    public synchronized void M1()  
    { x++; }  
    public static synchronized void M2()  
    { s++; }  
}
```

É equivalente a ...

Programação concorrente em Java

```
public class Exemplo {
    private int x;
    private static int s;

    public void M1() {
        synchronized (this)
            { x++; }
    }

    public static void M2() {
        try { synchronized (Class.forName( "Exemplo" ) )
            { s++; }
        }
        catch (ClassNotFoundException e) { ... }
    }
}
```

Programação concorrente em Java

Exemplo 2:

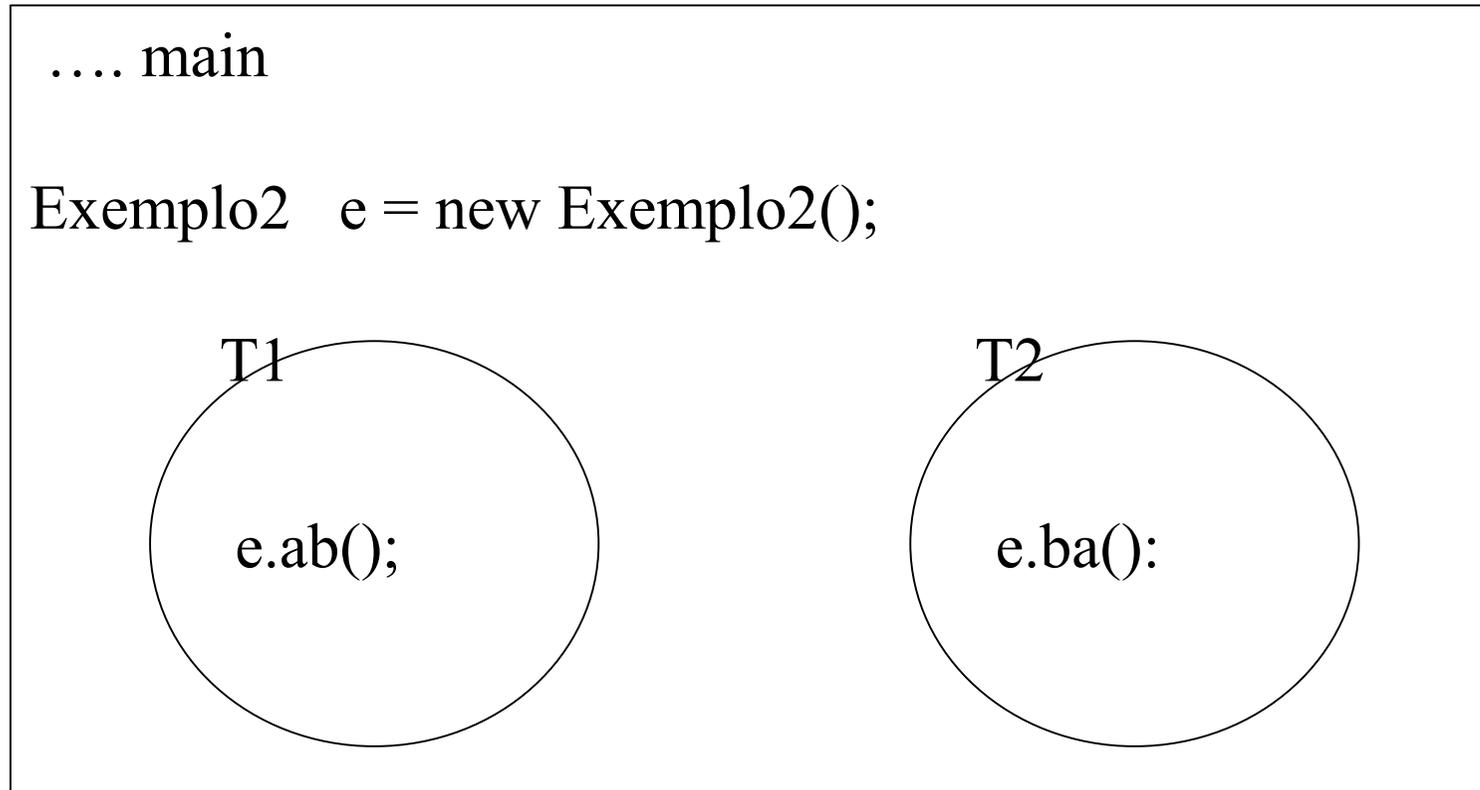
Sejam duas Threads, T1 e T2. Supondo que T1 invoca o método ab e T2 invoca o método ba no mesmo objecto da classe Exemplo2, quais são os possíveis valores finais para a e b em cada caso, i) e ii) ?

```
i) public class Exemplo2 {  
    private int a = 1, b = 2;  
    public void ab()  
    { a = b; }  
    public void ba()  
    { b = a; }  
}
```

Implementar e testar a resposta ... !!

Programação concorrente em Java

Exemplo 2:



Como implementar ?

Qual o output quando o objeto partilhado é um objeto da classe Exemplo2?

Programação concorrente em Java

a=2,b=1 | a=2,b=2 | a=1,b=1

```
ii) public class Exemplo3 {  
    private int a = 1, b = 2;  
    public synchronized void ab()  
    { a = b; }  
    public synchronized void ba()  
    { b = a; }  
}
```

Qual o output quando o objeto partilhado é um objeto da classe Exemplo3?

$$a=2,b=2 \mid a=1,b=1$$

E nos casos iii, iv e v quais são os **outputs possíveis**, supondo que T1 invoca o método M1 e T2 invoca o método M2 no mesmo objeto da classe Exmplo4?

```
iii) public class Exemplo4 {  
    private int a = 1, b = 2;  
    public void M1()  
    { a = 3; b = 4; }  
    public void M2()  
    { System.out.println ("a=" + a + "b=" + b); }  
}
```

$a=3,b=4 \mid a=3,b=2 \mid a=1,b=2 \mid a=1,b=4$

```
iv) public class Exemplo5 {  
    private int a = 1, b = 2;  
    public synchronized void M1()  
    { a = 3; b = 4; }  
    public void M2()  
    { System.out.println ("a=" + a + "b=" + b); }  
}
```

Programação concorrente em Java

$a=3,b=4 \mid a=3,b=2 \mid a=1,b=2 \mid a=1,b=4$

→ o facto de um método ser sincronizado não faz com que se comporte como se fosse uma instrução atómica

```
v) public class Exemplo6{  
    private int a = 1, b = 2;  
    public synchronized void M1()  
    {a = 3; b = 4; }  
    public synchronized void M2()  
    { System.out.println ("a=" + a + "b=" + b); }  
}
```

$$a=3,b=4 \mid a=1,b=2$$

➤ “Multithreaded servers”

E se quisermos que um servidor atenda vários clientes em simultâneo?

- Em máquinas multicore;
- Em situações em que o servidor faça operações de input / output como os servidores de bases de dados ou de ficheiros;
 - . servir vários clientes em simultâneo pode melhorar significativamente o seu desempenho.
- Isso pode ser feito, por exemplo, criando uma thread para “servir” cada cliente.
- O mesmo é válido para o cliente: é possível melhorar o desempenho de alguns processos clientes criando várias threads para distribuir as tarefas.

Programação concorrente em Java

➤ Várias arquiteturas são possíveis:

1 - Uma thread por pedido (thread-per-request)

- É criada uma thread para cada pedido do cliente.

2 - Uma thread por ligação (thread-per-connection)

- É criada um thread por cada cliente que se liga.

3 - Uma thread por objecto (thread-per-object)

- A cada objecto remoto é associada uma thread.

4 - Thread-pool

- O servidor cria um conjunto inicial de threads.

Programação concorrente em Java

Caso de estudo:

1 - “Multithreaded server”

Suponhamos um servidor, que comunica a cada cliente que o solicita, a data e a hora do sistema:

Cliente

```
import java.net.*;
import java.io.*;
public class Cliente {
    private Socket s;
    public Cliente() {
        try {
            s = new Socket (“127.0.0.1”, 5432);
```

Programação concorrente em Java

- “Multithreaded server” (Cliente - cont.)

```
    ObjectInputStream is = new ObjectInputStream( s.getInputStream() );
    System.out.println( is.readObject() );
    s.close();
} //try
catch ( IOException e)
{   System.out.println(e.getMessage());}
} //construtor

    public static void main (String args []){
        Cliente c = new Cliente();
    }
} //Cliente
```

Servidor:

```
public class Servidor {  
    private ServerSocket ss;  
    private Socket s;  
    private Connection c;  
    public Servidor(){  
        try {  
            ss = new ServerSocket (5432);  
        }  
        catch ( IOException e){  
            ...  
        }  
    }  
}
```

Programação concorrente em Java

Servidor (cont.)

```
try {  
    while (true) {  
        s = ss.accept();  
  
        c = new Connection (s);  
  
    }  
}  
catch (IOException e) {...}  
}  
public static void main (String args[]) {  
    Servidor dataHora = new Servidor();  
}  
}
```

aceita uma ligação pedida por um cliente



- O pedido é tratado por uma nova Thread;

- O ServerSocket pode aceitar outra(s) ligações

Programação concorrente em Java

```
public class Connection extends Thread {
    private Socket S;
    public Connection ( Socket s ) {
        super();
        S = s;
        start();
    }

    public void run () {
        try {
            ObjectOutputStream os = new ObjectOutputStream( S.getOutputStream());
            os.writeObject ( “A data e hora do sistema: ” + new java.util.Date() );
            os.flush();
        }
        catch ( IOException e ) {...}
    }
}
```

Ex.io: Fazer um esquema que represente os vários processos e a comunicação, no ex.io.

Programação concorrente em Java

Exercício 2: Suponha uma aplicação cliente/servidor em que :

- Cada cliente aposta num número inteiro de 0 a 99;
- Servidor gera um número inteiro aleatório entre 0 e 99 e se for igual à aposta do cliente este ganha um determinado prémio. Note que para cada cliente deve ser gerado um novo número.

O valor do prémio é determinado da seguinte forma: é inicializado a 0; para cada cliente que joga é adicionado ao prémio o valor de 1€. Quando um cliente acerta no número gerado ganha 50% do valor acumulado até esse momento no prémio. Após um cliente acertar no prémio, o prémio volta a zero.

- O Servidor devolve uma mensagem com o texto “ Parabéns, ganhou XXX €” caso acerte e em que XXX deve conter o valor ganho. Caso o cliente não acerte no valor gerado, a mensagem deverá ser “Continue a tentar, o prémio já é de YYY €” em que YYY é o valor que poderia ganhar nesse momento.

Programação concorrente em Java

– Implemente a aplicação anterior usando Sockets TCP para comunicação entre o cliente e um servidor multithreaded. Use ObjectStreams.

a) Construa o processo cliente.

b) Construa o processo servidor

c) Construa a Thread que comunica com o cliente.