Programmazione Concorrente in Java

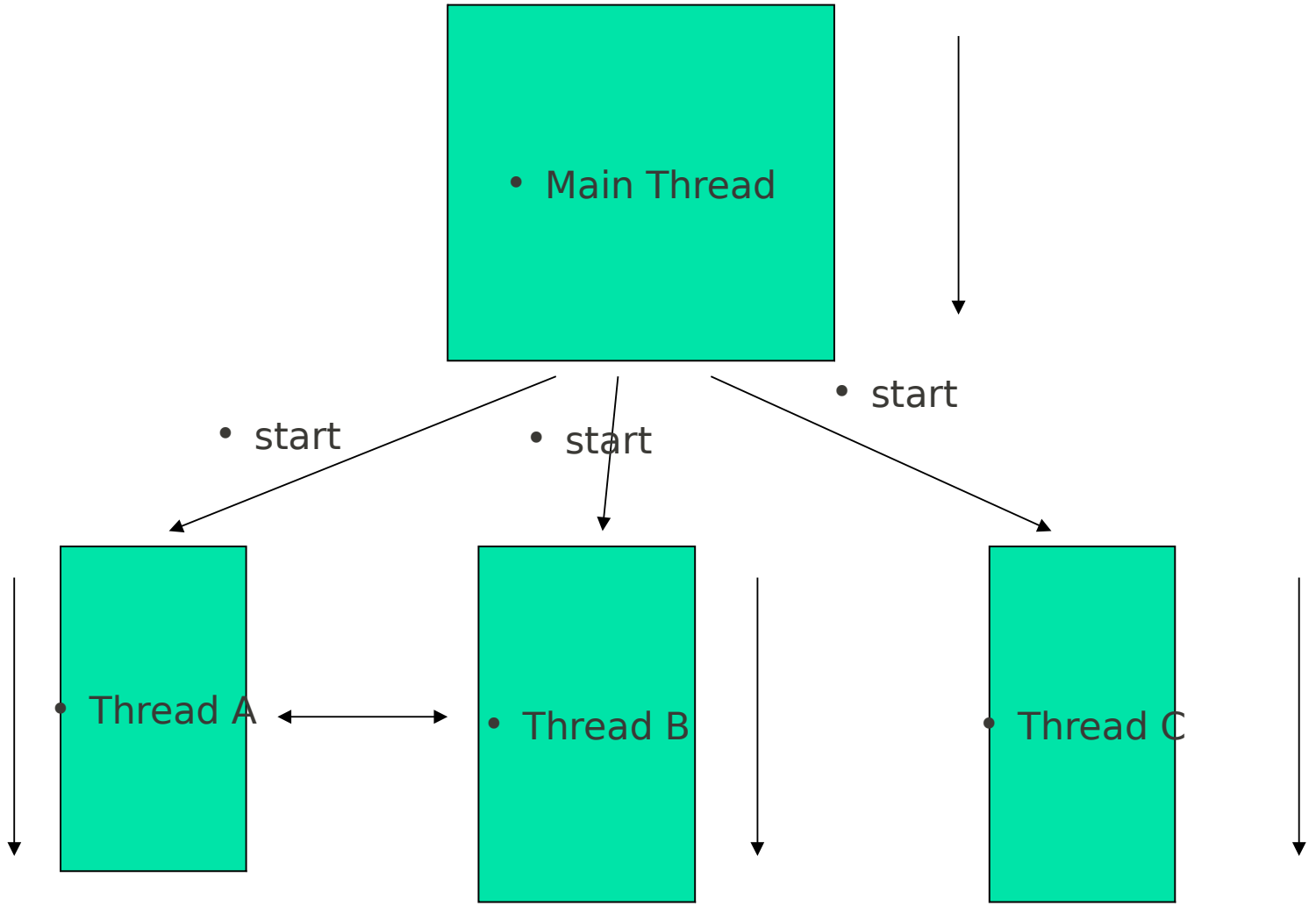
E MUMOLO DIA

Multithreading in Java

- Ogni thread e' un oggetto, creato come istanza della classe *java.lang.Thread*
- La classe Thread contiene tutti i metodi per gestire i threads
- L'utente implementa il metodo *run*()
- Uno dei metodi piu' importanti e' il metodo *start*() che lancia il thread utilizzando il metodo run definito dall'utente
- Ogni istanza di *Thread* deve quindi essere associata ad un metodo run
- Ci sono due metodi per realizzare un thread:
  - Implementando l'interfaccia Runnable
  - L'interfaccia contiene il solo metodo **public void run()**
  - Estendendo la classe java.lang.Thread e sovrascrivendo il metodo *run*()
- Un thread termina quando
  - Finisce
  - Viene eseguito il metodo *stop*() del thread
  - Scatta una eccezione
- Lo scheduling e' basato sullle priorita' (metodo *setPriority*())
- A parita' di priorita' → round-robin
- Preemptive

http://java.sun.com/docs/books/tutorial/essential/concurrency/

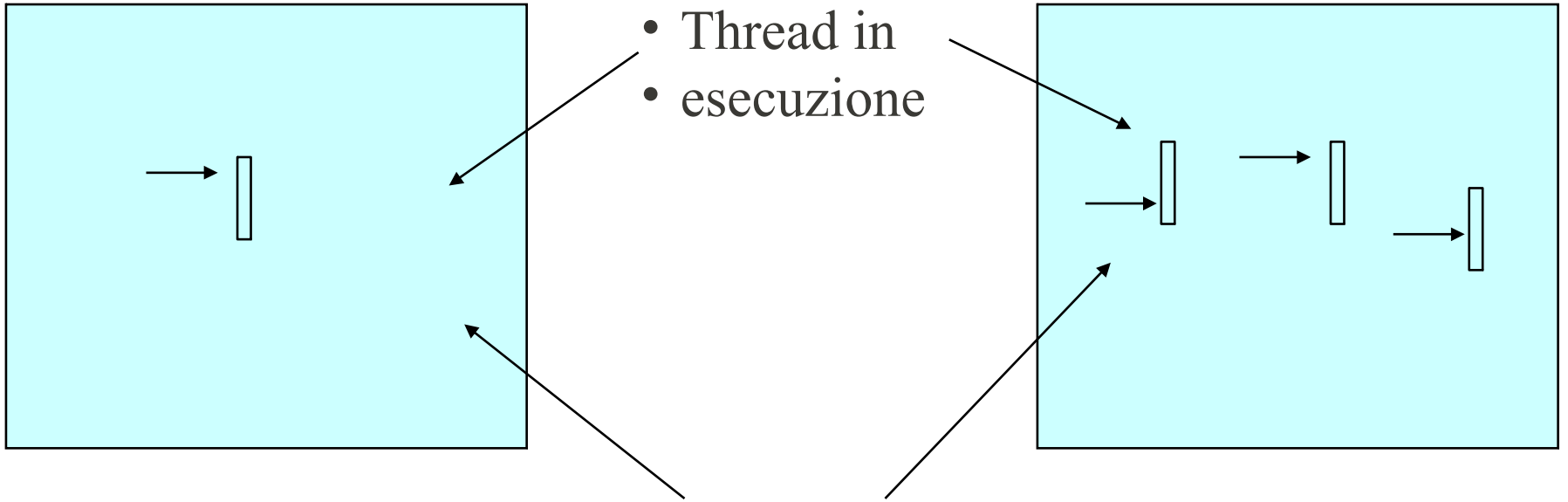
Un programma Multithreaded



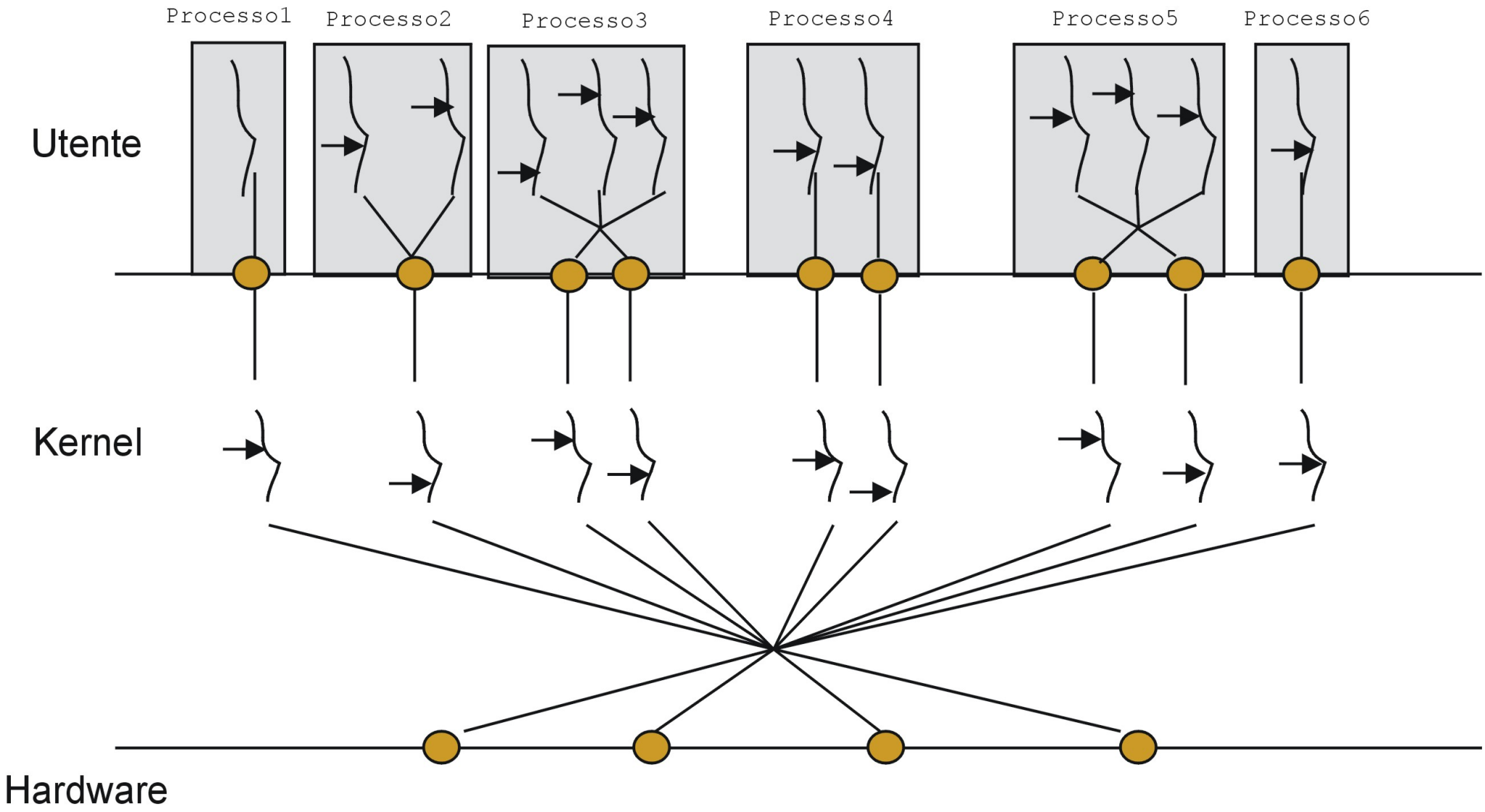
Processi a Thread singolo e a Thread multipli

- I Thread sono dei processi che eseguono all'interno di un processo

- Processo con singolo Thread
- Processo con Thread multipli



- Spazio di indirizzamentocomune


Processo1
Processo2
Processo3
Processo4
Processo5
Processo6
Utente
Kernel
Hardware

Multithreading in Java

- Creazione ed esecuzione di un Thread in Java:

```
Thread t = new Thread();
t.start();
```

- Questo thread non ha codice e si ferma subito. Come si indica il codice del Thread? Ci sono due modi:

Mediante creazione di una classe derivata:

```
public class MioThread extends Thread {
public void run(){
   System.out.println("Sono MioThread ");
}
}
MioThread t = new MioThread();
t.start();
```

Mediante implementazione classe Runnable

```
public class MioRunnable implements Runnable {
public void run(){
   System.out.println("Sono MioRunnable ");
}
}
Thread t = new Thread(new MioRunnable());
t.start();
```

Esempio mediante classe derivata

```
class MyThread extends Thread {      // il thread
        public void run() {
                System.out.println(" … questo  thread sta eseguendo ... ");
         }
} //  class MyThread

class ThreadEx1 {       // un programma che utilizza il thread
        public static void main(String [] args  ) {
            MyThread t = new MyThread();
            // ora chiamo start e questo chiama il metodo run.
             t.start();
        } // end main()
}
```

Esempio mediante implementazione interfaccia

```
class MyThread implements Runnable  {
        public void run() {
                System.out.println("..questo thread è in esecuzione... ");
        }
} // fine class MyThread

class ThreadEx2 {
        public static void main(String [] args  ) {
                Thread t = new Thread(new MyThread());
                        // ora chiamo start e questo chiama run
                t.start();
        }
}
```

Altro esempio mediante creazione subclass 1

```
import java.io.*;
public class PingPong2{   // il main crea e lancia i thread
    public static void main(String[] a){
                Ping2 c=new Ping2(); c.start(); Pong2 t=new Pong2(); t.start();
    }
}
class Ping2 extends Thread{
    public void run(){
                while(true) {
                        try{ Thread.sleep(800); } catch(InterruptedException e) {}
                        System.out.println("sono un thread creato dal main");
                }
    }
}
class Pong2 extends Thread{
    public void run(){
                while(true) {
                    try{ Thread.sleep(990); } catch (InterruptedException e){}
                    System.out.println("\tsono un thread creato dal main");
                }
    }
}
```

Altro esempio mediante creazione subclass 2

```
import java.io.*;
public class PingPong3{
     public static void main(String[] a){
                Ping3 c=new Ping3();  Pong3 t=new Pong3();
     }
}
class Ping3 extends Thread{ //ogni thread si lancia da solo
     public void run(){
                while(true) {
                        try{ Thread.sleep(800); } catch(InterruptedException e) {}
                        System.out.println("sono un thread creato dal main");
                }
     }
    Ping3(){ start(); }
}
class Pong3 extends Thread{
     public void run(){
                while(true) {
                    try{ Thread.sleep(990); } catch (InterruptedException e){}
                    System.out.println("\tsono un thread creato dal main");
                }
     }
    Pong3(){ start(); }
}
```

Altro esempio:
mediante implementazione Runnable 1

```
import java.io.*;
public class PingPong{    //il main crea e lancia i thread
       public static void main(String[] a){
            Ping c=new Ping(); Pong t=new Pong();
  –      Thread th1=new Thread(c); th1.start();
          Thread th2=new Thread(t); th2.start();
       }
}
class Ping implements Runnable{
        public void run(){
                while(true) {
                        try{ Thread.sleep(800); } catch(InterruptedException e) {}
                        System.out.println("sono un thread creato dal main");
                }

        }
}
class Pong implements Runnable{
         public void run(){
                while(true) {
                    try{ Thread.sleep(990); } catch (InterruptedException e){}
                    System.out.println("\tsono un thread creato dal main");
                }
     }
}
```

Altro esempio:
mediante implementazione Runnable 2

```
import java.io.*;
public class PingPong1{     // file PingPong1.java
     public static void main(String[] a){
         Ping1 c=new Ping1(); Pong1 t=new Pong1();
     }
}
class Ping1 implements Runnable{ //file Ping1.java. Ogni oggetto crea e lancia il proprio thread
     Thread th;
     public void run(){
               while(true) {
                       try{ Thread.sleep(800); } catch(InterruptedException e) {}
                       System.out.println("sono un thread creato dal main");
               }
     }
      Ping1() {th=new Thread(this); th.start();}
}
class Pong1 implements Runnable{ //file Pong1.java
     Thread th;
     public void run(){
               while(true) {
                   try{ Thread.sleep(990); } catch (InterruptedException e){}
                   System.out.println("\tsono un thread creato dal main");
               }
     }
          Pong1(){ th=new Thread(this); th.start(); }
}
```

Alcuni metodi della classe Thread

- `public void start()` `//lancia il thread`
- `public void run()` `//esegue il codice`
- `public final void stop()` `//distrugge il thread`
- `public final void suspend()` `//sospende il thread`
- `public final void resume()` `//riattiva il thread`
- `public static void sleep(long n)` `//sospende il thread per n ms`
- `public final void setPriority(int priority)` `//modifica la priorita'`
- `public final int getPriority()` `//ottiene la priorità corrente`
- `public static void yield()` `//rischedula`

`public final native boolean isAlive()` `//esce con true se il thread è vivo`

...

Esempio

```
public class MyThread extends Thread{
    private Thread ThrPrincipale;

    public MyThread(){
        ThrPrincipale = Thread.currentThread();
    }

    public void run(){
        for (int i=0;i<10; i++) {
            PrintMsg();
        }
    }

    public void PrintMsg(){
            Thread t = Thread.currentThread();
            if( t== ThrPrincipale)
                System.out.println("sono il thread principale");
            else if ( t == this )
                System.out.println("sono il nuovo thread!");
            else System.out.println("Non so chi sono!");
    }

    public static void main(String[] args) {
            MyThread tt = new MyThread();
            tt.start();
            for(int i=0; i<10;i++) tt.PrintMsg();
    }
}
```

Suspend/Stop/**Resume**

```
import java.io.*;

class Ping implements Runnable{
        public void run(){
                while(true) {
                        try{ Thread.sleep(800);
                 } catch(InterruptedException e) {}
                        System.out.println("Ping");
                }
        }

        public static void main(String[] a) throws Exception{
          Ping c = new Ping();
         Thread th1=new Thread(c);
             System.out.println("esegue");
             th1.start();
                System.out.println("sospende");
                th1.suspend();
// fai qualcosa...
                Th1.resume();
       }
}
```

Suspend/**Stop**/Resume

```
import java.io.*;

class Ping implements Runnable{
        public void run(){
                while(true) {
                        try{ Thread.sleep(800);
                 } catch(InterruptedException e) {}
                        System.out.println("Ping");
                }
         }

        public static void main(String[] a) throws Exception{
             Ping c = new Ping();
             Thread th1=new Thread(c);
             System.out.println("stop");
             th1.start();
//
             th1.stop();
             System.out.println("il thread e' stato fermato");
             th1.join();
       }
}
```

Stop/Suspend/Resume

Attenzione:

Stop() e suspend(): rischio di STALLO!!
Supponiamo che il thread avesse acquisito una risorsa in mutua esclusione!
Dal JDK 1.2 stop(), suspend() e resume() → deprecated
Usare wait() e notify() !
La JVM associa ad ogni oggetto che chiama l'istruzione synchronized un LOCK.

Funzione della wait():

- rilascia il LOCK → wait() deve essere inserita in un metodo synchronized
- inserisce il thread in coda d'attesa
- mette il thread in stato NotRunnable

Esempio:

```
Public synchronized void Esempio(){
    if(!condizione) try
    {
        wait();
    } catch(InterruptedException e)
        {System.out.println("eccezione di wait!");}
}
```

Setpriority/getpriority

```
import java.io.*;

class Ping implements Runnable{
        public void run(){
                while(true) {
               //ricava la  Priorita' del thread
                       System.out.println( th.getName()+" attuale "+th.getPriority());
               //cambia la priorita'
                       th.setPriority(Thread.MAX_PRIORITY);
               //ricava la  Priorita' del thread
                       System.out.println(th.getName()+" nuova "+th.getPriority());
            }
         }

        public static void main(String[] a) throws Exception{
             Ping c = new Ping();
             Thread th1=new Thread(c);
             System.out.println("esegue");
             th1.start();
         }
}
```

Yield

```
import java.util.List;
public class YieldEsempio {
    public static void main(String a[]){
        for(int i=0;i<3;i++){
            Simple tt = new Simple(i+5);
            tt.start();
        }
     }
}

class Simple extends Thread{
    private int stopCount;
    public Simple(int count){
        this.stopCount = count;
    }
    public void run(){
        for(int i=0;i<15;i++){
            if(i%stopCount == 0){
                System.out.println(" in " + i + ", " + getName() + " sta cedendo il
                    controllo a ...");
                yield();
            }
        }
    }
}
```

isAlive

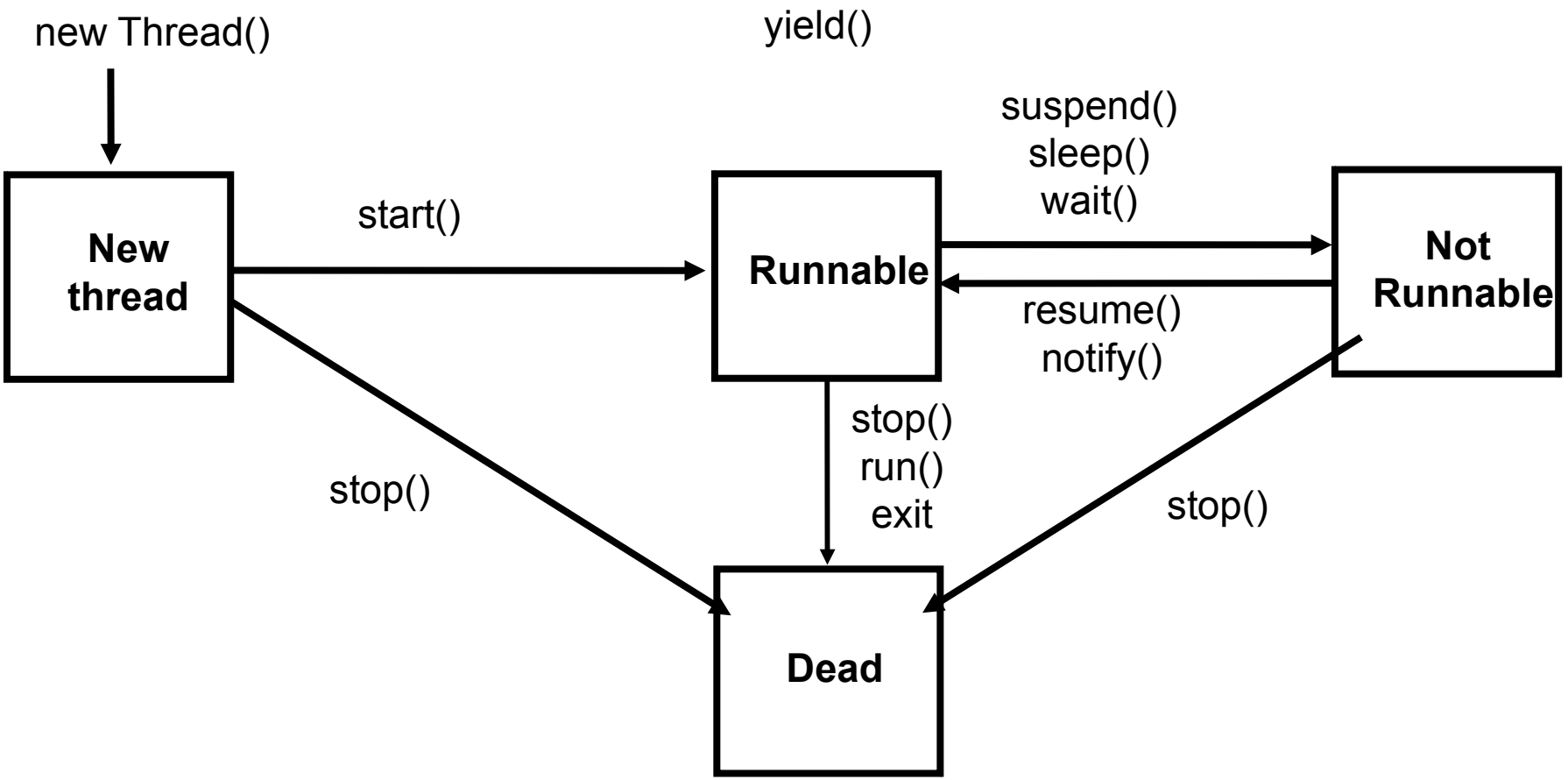
```
import java.io.*;

class Ping implements Runnable{
        public void run(){
                Thread th = Thread.currentThread();
                //Tests if this thread is alive
                System.out.println(th.isAlive());
            }

        public static void main(String[] a) throws Exception{
           Ping c = new Ping();
           Thread th=new Thread(c);
           th.start();
           th.join();
           System.out.println(th.isAlive());
        }
}

Uscita:
$java Ping
true
false
$
```

Stati di un Thread



Passaggio di parametri tra Thread

- Mediante visibilità delle variabili (attribuzione public static)
- Possibile solo per piccoli programmi

```
import java.io.*;
public class PingPong2{ // il main crea e lancia i thread
    public static  int numero=0;

    public static void main(String[] a){
       Ping2 c=new Ping2(); c.start();
       Pong2 t=new Pong2(); t.start();
    }
}
class Ping2 extends Thread{
    public void run(){
      while(true) {
        try{ Thread.sleep(800); } catch(InterruptedException e) {}
        PingPong2.numero++; System.out.println("Ping "+PingPong2.numero);
      }
    }
}
class Pong2 extends Thread{
    public void run(){
      while(true) {
        try{ Thread.sleep(990); } catch (InterruptedException e){}
        PingPong2.numero++; System.out.println("\tPong " +PingPong2.numero);
      }
    }
}
```

Passaggio di parametri tra Thread

- PER PROGRAMMI PIU' GRANDI → CONDIVISIONE DI UNA CLASSE DATI
- Tutti i thread devono condividere un oggetto, che contiene i dati e i metodi
- La condivisione viene effettuata mediante definizione del puntatore all'oggetto in ciascun thread, e mediante l'inizializzazione del puntatore all'oggetto
- L'oggetto in realta' viene allocato nella classe principale (quella che contiene il main)
- Esempio: due thread – p1 e p2 – che si scambiano 5 reali, con ritardo (genera sequenzialita' quindi in questo caso non ci sono problemi di mutua esclusione)

```
public class pth{
    public static void main(String[] a){
                z buf=new z();
                p1 c=new p1(buf); p2 t=new p2(buf);
                c.start(); t.start();
    }
}
```

- Definizione della classe z:

```
import java.io.*;
public class z{
        float b[]= new float[10];
        void put(int i, float f)
             {  b[i]=f;  }
        float get(int I)
             { return(float)b[i];  }
}
```

Esempio: produttore/consumatore

```
import java.io.*;
import java.util.*;
class mypc   // Classe principale contentene il main
{
  static Buffer buf=new Buffer();
  public static void main(String Arg[])
  {
    Prod  a=new Prod(buf);
    Cons  b=new Cons(buf);
    Cons1 c=new Cons1(buf);
    a.start();
    b.start();
    c.start();
  }
}
class Buffer
// Classe che realizza il buffer condiviso
{
  private int buf[]=new int [5];
  private int in;
  private int out;
  Buffer () { in=0; out=0;}
  int get_in()  { return in;  }
  int get_out() { return out; }
  int get_buf() { return buf[out]; }
  void put_in  (int a) { in=a; }
  void put_out (int a) { out=a; }
  void put_buf (int a) { buf[in]=a; }

}
```

```
class Prod extends Thread // Classe che realizza il thread produttore
{
  Buffer b;
  Random r;
  Prod(Buffer p){ this.b=p; r=new Random(); }
  public void run(){
    int el;
    while (true){
    el=r.nextInt();  // Nuova produzione
    while(b.get_out()==((b.get_in()+1)%5))
             {System.out.print("");};//Buffer pieno?
    b.put_buf(el);
    System.out.println("P  - Ho prodotto:  "+el);
    b.put_in((b.get_in()+1)%5);
    }
  }
}
class Cons extends Thread // Classe che realizza il thread del consumatore1
{
  Buffer b;
  Cons(Buffer p) { this.b=p; }
  public void run()
  {
    while(true)
    {
      int el;
      while(b.get_in()==b.get_out()) {System.out.print("");};//Buffer vuoto?
      el=b.get_buf();
      System.out.println("C1 - Ho consumato: " + el + " OUT=" + b.get_out());
      b.put_out((b.get_out()+1)%5);
    }
  }
}
```

```
class Cons1 extends Thread
// Classe che realizza il thread del consumatore2
{
  Buffer c;

  Cons1(Buffer p) { this.c=p; }

  public void run()
  {
    while(true)
    {
      int el;
      while(c.get_in()==c.get_out()) {System.out.print("");};//Buffer vuoto?
      el=c.get_buf();
      System.out.println("C2- Ho consumato: " + el + " OUT=" + c.get_out());
      c.put_out((c.get_out()+1)%5);
    }
  }
}
```

Esempio: Determinatezza

Due thread, senza vincoli di precedenza, lavorano su un buffer di due elementi.
Il primo thread realizza la seguente funzione: M1=M1+M2, M2=M2.
Il secondo realizza la funzione M1=M1, M2=M1+M2.

```
import java.lang.*;
import java.util.*;

class Determ
{
  public static void main(String argv[])
  {
    Buffer buf=new Buffer();
    Proc1 a=new Proc1(buf);
    Proc2 b=new Proc2(buf);
    a.start();
    b.start();
    for (int i=0; i<100000000; i++);
    System.out.print("\nStato del finale del buffer: ");
    buf.print();
  }
}
class Buffer
{
  private int buf[]=new int[2];

  public Buffer() { buf[0]=2;buf[1]=1; }
  public int get(int i){ return buf[i]; }
  public void put(int m,int i) { buf[i]=m; }
  public void print() { System.out.println("M1="+buf[0]+" M2="+buf[1]); }
}
```

```
class Proc1 extends Thread
{
  private Buffer buf;
  private int temp[]=new int[2];
  Random r=new Random();

  Proc1(Buffer ind) { buf=ind;}
  public void run()
  {
    // -------------------------ritardo casuale-----
    float casuale=r.nextInt(10000);
    for (int cont=0; cont<casuale; cont++) {System.out.print("");};
    // ------------------------------------------------
    temp[0]=buf.get(0);temp[1]=buf.get(1);          // Legge buffer
    buf.put(temp[0]+temp[1],0);buf.put(temp[1],1); // Scrive buffer
  }
}

class Proc2 extends Thread
{
  private Buffer buf;
  private int temp[]=new int[2];

  Proc2(Buffer ind) { buf=ind;}
  public void run()
  {
    temp[0]=buf.get(0);temp[1]=buf.get(1);          // Legge buffer
        buf.put(temp[0],0);buf.put(temp[0]+temp[1],1); // Scrive buffer
    }
}
```