

Thread Executor in Java

1 Il problema dei thread tradizionali

Creare thread manualmente:

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

Problemi:

- creare molti thread **consuma memoria**
- difficile gestire molti thread
- i thread **non vengono riutilizzati**

Per risolvere questo problema Java introduce **Executor Framework**.

Pacchetto:

```
java.util.concurrent
```

2 Cos'è un Executor

Un **Executor** è un sistema che:

- gestisce automaticamente i thread
- usa un **thread pool**
- esegue i task quando un thread è libero

Schema:

```
Task → Executor → Thread Pool → esecuzione
```

Noi inviamo il **task**, l'Executor sceglie il **thread disponibile**.

3 ExecutorService

L'interfaccia più usata è:

```
ExecutorService
```

Si crea con:

```
Executors
```

Esempio:

```
ExecutorService executor = Executors.newFixedThreadPool(3);
```

Significa:

→ il programma usa **3 thread riutilizzabili**.

4 Runnable e Callable (sintesi)

Un **task** è il lavoro che deve eseguire un thread.

Runnable

Interfaccia semplice.

Metodo:

```
void run()
```

Caratteristiche:

- **non restituisce risultati**
- usato per eseguire azioni

Esempio:

```
Runnable task = () -> {  
    System.out.println("Task eseguito");  
};
```

Callable

Simile a Runnable ma **restituisce un risultato**.

Metodo:

```
V call()
```

Caratteristiche:

- restituisce valore
- usato per **calcoli**

Esempio:

```
Callable<Integer> task = () -> 10 + 20;
```

Differenza principale:

Runnable	Callable
run()	call()
nessun risultato	restituisce valore

5 Eseguire un task

```
ExecutorService executor = Executors.newFixedThreadPool(2);

executor.execute(() -> {
    System.out.println(Thread.currentThread().getName());
});

executor.shutdown();
```

6 Tipi di Thread Pool

Fixed Thread Pool

Numero fisso di thread.

```
Executors.newFixedThreadPool(4);
```

Cached Thread Pool

Crea thread quando servono.

```
Executors.newCachedThreadPool();
```

Single Thread Executor

Un solo thread.

```
Executors.newSingleThreadExecutor();
```

7 Chiudere l'Executor

Alla fine bisogna chiuderlo:

```
executor.shutdown();
```

ESERCIZIO

Realizzare un **Mini Pianoforte con GUI**.

Il programma deve:

1. creare una **finestra grafica**
2. avere **7 pulsanti** (C D E F G A B)
3. ogni pulsante rappresenta **una nota musicale**
4. quando si preme un pulsante:
 - viene creato un **task Runnable**
 - il task viene eseguito tramite **ExecutorService**
 - il task suona la nota MIDI
5. usare un **Thread Pool di 4 thread**

Schema logico:

```
Click pulsante  
↓  
creazione Runnable Task  
↓  
ExecutorService.execute()  
↓  
thread pool esegue run()  
↓  
suono nota
```

Codice Java completo

```
import javax.swing.*;
import javax.sound.midi.*;
import java.awt.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class PianoGUI {

    // Thread Pool con 4 thread
    static ExecutorService executor = Executors.newFixedThreadPool(4);

    public static void main(String[] args) {

        // Creazione finestra
        JFrame frame = new JFrame("Mini Piano con Thread Pool");
        frame.setSize(600,200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout(10,10));

        // Etichetta in alto per mostrare la nota premuta
        JLabel label = new JLabel("Premi un tasto", SwingConstants.CENTER);
        label.setFont(new Font("Arial", Font.BOLD, 24));
        frame.add(label, BorderLayout.NORTH);

        // Pannello centrale con pulsanti
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(1,7,5,5)); // 1 riga, 7 colonne, margine

        // Note MIDI e etichette dei tasti
        int[] notes = {60,62,64,65,67,69,71};
        String[] labels = {"C","D","E","F","G","A","B"};

        for(int i=0; i<notes.length; i++){

            int note = notes[i];
            String noteLabel = labels[i];

            JButton button = new JButton(noteLabel);
            button.setFont(new Font("Arial", Font.BOLD, 18));
            button.setBackground(Color.WHITE);
            button.setOpaque(true);
            button.setBorder(BorderFactory.createLineBorder(Color.BLACK));

            // Quando si clicca il pulsante
            button.addActionListener(e -> {
                label.setText("Nota premuta: " + noteLabel);
                // Log click in console
                System.out.println("GUI: Click su " + noteLabel);
                // Esegue il task tramite Thread Pool
                executor.execute(new PianoTask(note, noteLabel));
            });

            panel.add(button);
        }

        frame.add(panel, BorderLayout.CENTER);

        frame.setVisible(true);
    }
}
```

5px

```

    }
}

// Classe Runnable che rappresenta il task di riproduzione audio
class PianoTask implements Runnable {

    private int note;
    private String noteLabel;

    public PianoTask(int note, String noteLabel){
        this.note = note;
        this.noteLabel = noteLabel;
    }

    @Override
    public void run(){

        try{
            // Log inizio task
            System.out.println("Thread " + Thread.currentThread().getName() +
                               " inizia a suonare " + noteLabel);

            // Sintetizzatore MIDI
            Synthesizer synth = MidiSystem.getSynthesizer();
            synth.open();
            MidiChannel[] channels = synth.getChannels();

            // Suona la nota
            channels[0].noteOn(note, 100);
            Thread.sleep(500); // durata nota
            channels[0].noteOff(note);

            synth.close();

            // Log fine task
            System.out.println("Thread " + Thread.currentThread().getName() +
                               " ha terminato " + noteLabel);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Spiegazione passo passo

1 Thread Pool

```
static ExecutorService executor = Executors.newFixedThreadPool(4);
```

- Crea un **pool di 4 thread riutilizzabili**.
 - Quando clicchi un tasto, il task viene messo in **coda** se tutti i thread sono occupati.
 - Permette di **gestire più click contemporanei senza creare nuovi thread ogni volta**.
-

2 GUI e etichetta

```
JLabel label = new JLabel("Premi un tasto", SwingConstants.CENTER);
```

- Mostra la **nota premuta** in tempo reale.
 - Aiuta l'utente a capire quale tasto è stato attivato.
-

3 Pulsanti dei tasti

```
int[] notes = {60, 62, 64, 65, 67, 69, 71};  
String[] labels = {"C", "D", "E", "F", "G", "A", "B"};
```

- `notes` → codici MIDI delle note.
 - `labels` → testo visibile sui pulsanti.
-

4 Gestione click

```
button.addActionListener(e -> {  
    label.setText("Nota premuta: " + noteLabel);  
    executor.execute(new PianoTask(note, noteLabel));  
});
```

- Quando clicchi il pulsante:
 1. L'etichetta GUI mostra la nota.
 2. Viene creato un **Runnable PianoTask**.
 3. Viene inviato all'**ExecutorService**.
-

5 Runnable PianoTask

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

- Ogni task:
 1. Stampa in console **il thread che sta eseguendo la nota**.
 2. Apre il **sintetizzatore MIDI**.
 3. Suona la nota per 500ms.
 4. Chiude il sintetizzatore.
 5. Stampa in console **fine task**.
-

6 Log dei thread

Esempio output:

```
GUI: Click su C
Thread pool-1-thread-1 inizia a suonare C
GUI: Click su E
Thread pool-1-thread-2 inizia a suonare E
Thread pool-1-thread-1 ha terminato C
Thread pool-1-thread-2 ha terminato E
```

- Ti permette di vedere **chi esegue cosa** e quando un thread diventa disponibile.
 - Ottimo per capire il funzionamento del **Thread Pool**.
-

7 Vantaggi

- GUI **reattiva**, anche se premi più tasti insieme.
- Thread Pool evita **sovraccarico di risorse**.
- Log in console **traccia l'esecuzione dei task**.