Java SE: Multithreading

- Multitasking
  - Multiprocessing vs Multithreading
  - Thread in Java
  - Sincronizzazione
  - Comunicazione tra thread
- Progetto di riferimento
  - https://github.com/egalli64/jse *(modulo 10)*

Multitasking

- Multiprocessing: esecuzione di più processi concorrenti:
  - Efficienza e miglior bilanciamento nell'uso delle risorse
  - Il sistema operativo assicura che siano indipendenti e isolati
  - Possono comunicare tra loro usando socket, memoria condivisa, file, …
- Multithreading: un singolo processo può avere più thread (flussi di esecuzione)
  - Un thread può essere pensato come un processo "leggero"
    - Ognuno ha il proprio program counter, stack di esecuzione, variabili locali
    - Ma condividono le risorse del processo, tra cui lo heap (con relative variabili)
  - È più economica la creazione, lo switch di contesto, la comunicazione
  - Ma è nostra responsabilità assicurare la correttezza dell'esecuzione
    - Sincronizzazione, accesso a variabili condivise

Runnable e Thread

- Il codice eseguito da un thread è definito nel metodo run()
  - Interfaccia funzionale Runnable
- La classe Thread:
  - Implementa Runnable
  - Offre una serie di metodi per la gestione del thread associato
    - Ad es. getName() ritorna il suo nome
- Esecuzione di un nuovo thread
  - Si crea un oggetto Thread passando al costruttore un runnable
    - Approcci alternativi: estensione o aggregazione di Thread + ridefinizione di run()
  - Si invoca il metodo Thread.start()

Main thread

- Un processo entra in esecuzione sul suo main thread
- Accesso al thread corrente via currentThread()
  - Metodo statico che ritorna un reference al thread corrente
- È qui che (di solito) vengono creati gli altri thread
- E (spesso) si resta in attesa della loro terminazione
  - Il metodo join() ritorna quando il thread associato termina

Sincronizzazione

- Race condition: più thread competono per l’accesso alla risorsa
  - Occorre serializzale l’esecuzione per evitare risultati scorretti
- In un oggetto con metodi “synchronized”, uno solo tra quei metodi può essere eseguito in un dato momento.
  - La sincronizzazione avviene su “this”
  - Eventuali altri thread devono attendere il proprio turno
  - I metodi non sincronizzati possono comunque essere acceduti senza limitazioni
- Un blocco può essere sincronizzazione su un qualunque oggetto
  - In questo modo è possibile ottenere una serializzazione più mirata

Comunicazione tra thread

- Basata sul concetto di monitor definito da Tony Hoare (~1970)
- In un blocco sincronizzato possono essere invocati questi metodi di Object
  - wait(): il thread corrente viene messo in attesa
  - notify() / notifyAll(): rimette in attività uno / tutti i thread in pausa nello stesso monitor. Un (o “il”) thread torna in esecuzione
    - Si preferisce usare notifyAll(), per non rischiare che la notifica non arrivi al thread corretto
- È possibile che un thread in pausa torni in esecuzione anche senza un notify. Conviene quindi controllare sempre la condizione che ha portato al wait prima di riprendere l’esecuzione