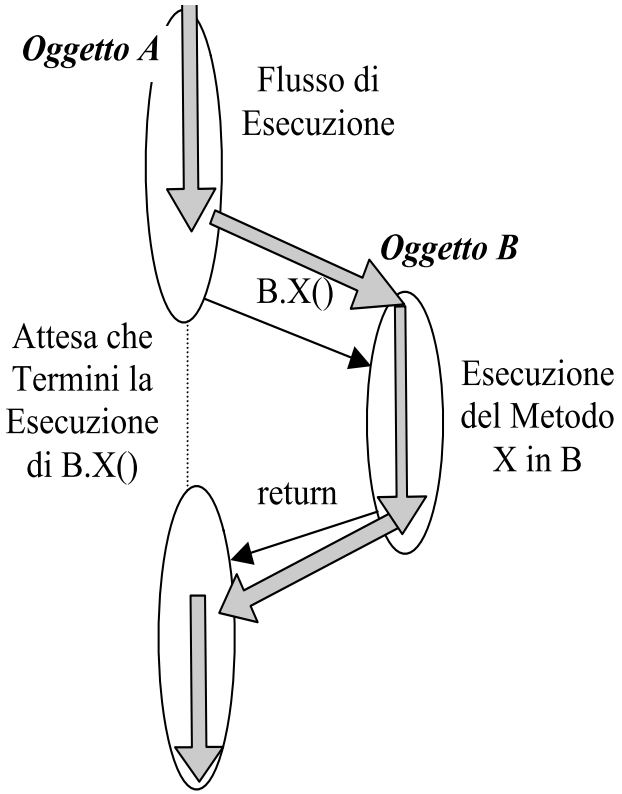
Thread in Java

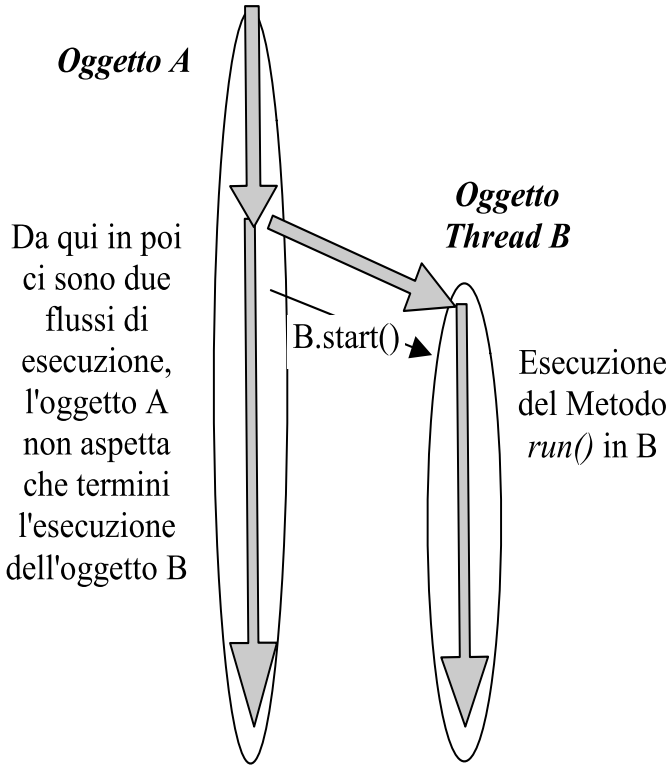
- **Thread** = flusso di esecuzione indipendente nel codice del programma
- Come si può realizzare il concetto di Thread in Java?
- Seguendo la filosofia OO: sono oggetti particolari ai quali si richiede un servizio (chiamato start()) corrispondente al lancio di una attività, di un thread
- MA: non si aspetta che il servizio termini, esso procede in concorrenza a chi lo ha richiesto

Richiesta di servizio

- Normale richiesta di servizio



- Richiesta di servizio start() a un thread



La classe Thread

- La classe Thread è la base per la gestione dei thread in Java
- Essa comprende metodi per attivare, fermare, sospendere, riprendere, attendere thread, per controllarne la priorità, lo scheduling, etc
- Creazione di un thread → normale creazione di un oggetto di classe Thread o derivate

Definire ed eseguire un thread

- Il corpo del thread è costituito dal metodo run()
- Quindi, per definire una nuova classe di thread si deve:
  - definire una propria sottoclasse di Thread che ridefinisca opportunamente il metodo run()
- Per creare un nuovo oggetto thread si dovrà quindi:
  - creare una istanza della sottoclasse di Thread
- Per far partire il nuovo thread (cioè l'esecuzione del suo metodo run()) basta poi invocare su di esso il metodo start()

Esempio di thread

Scaricare Esercitazione8-es1 dal server SVN

Esempio di thread 2

Scaricare Esercitazione8-es2 dal server SVN

L'interfaccia Runnable

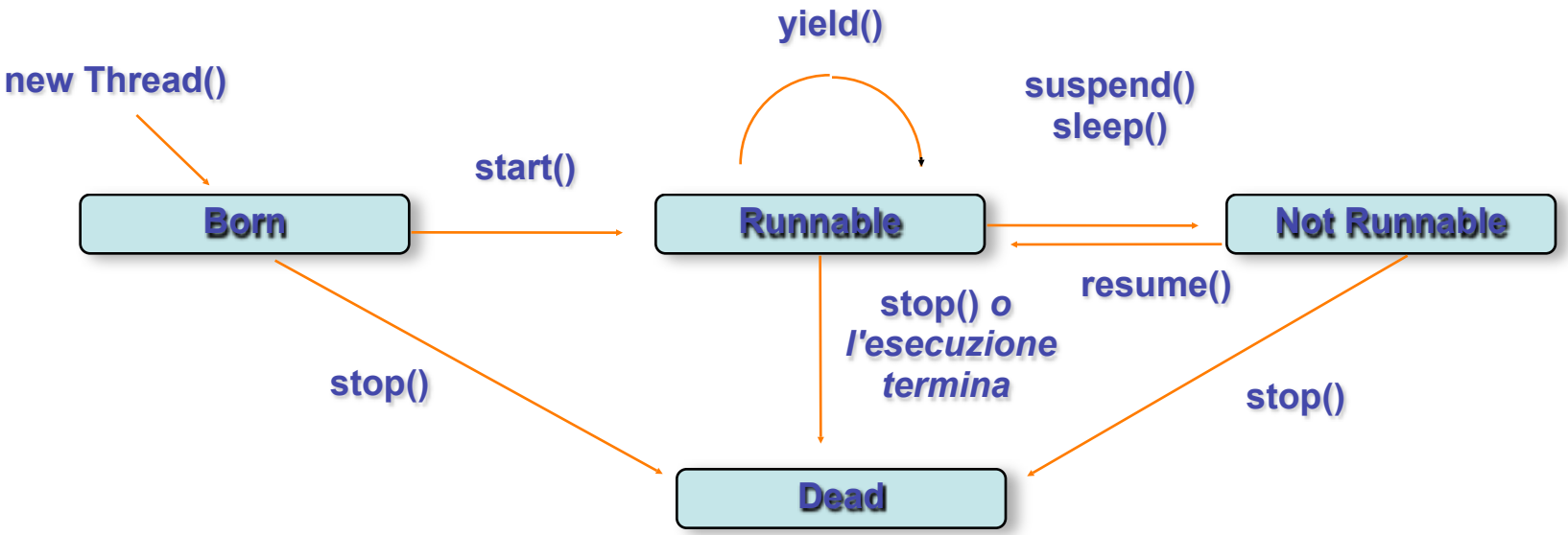
- Il metodo run() dei Thread è anche dichiarato nella interfaccia Runnable. Quindi, come metodo alternativo per definire dei thread si può:
  - definire una propria classe che implementi l'interfaccia Runnable
  - in questa classe, implementare il metodo run()

- Questa via è più flessibile (rispetto all'ereditare dalla classe Thread), in quanto consente di definire classi di thread che non siano per forza sottoclassi di Thread ma che siano sottoclassi di altri classi

Creare thread con Runnable

- Per creare e usare un thread bisogna:
  - prima creare una istanza della propria classe (che implementa l'interfaccia Runnable)
  - poi creare una istanza di Thread a partire da tale oggetto

Stato di un thread



Metodi

Metodi di Thread

- public void start()
- public void run()
- public void stop()
- public String getName()
- public void setName(String)
- public void sleep(long)
- public void suspend()
- public void resume()
- public void yield()

Interfaccia di Runnable

- public void run()

start() e stop()

- start() inizia l'esecuzione dopo la necessaria inizializzazione
  - invoca run() (o meglio la JVM invoca run() in un altro thread Java, facendo partire di fatto la vita del thread)
- stop() termina l'esecuzione del thread (lancia una eccezione ThreadDeath() alla vittima!!) se si ha il privilegio per farlo.

getName() e setName()

- Gestione del nome del thread
- Ogni Thread ha sempre associato un nome (non necessariamente univoco!), corrispondente a una stringa, che o viene assegnato di default dal sistema o viene assegnato dall'utente in fase di inizializzazione!
- Per ottenere il nome del thread corrente:

```
Thread.currentThread().getName();
```

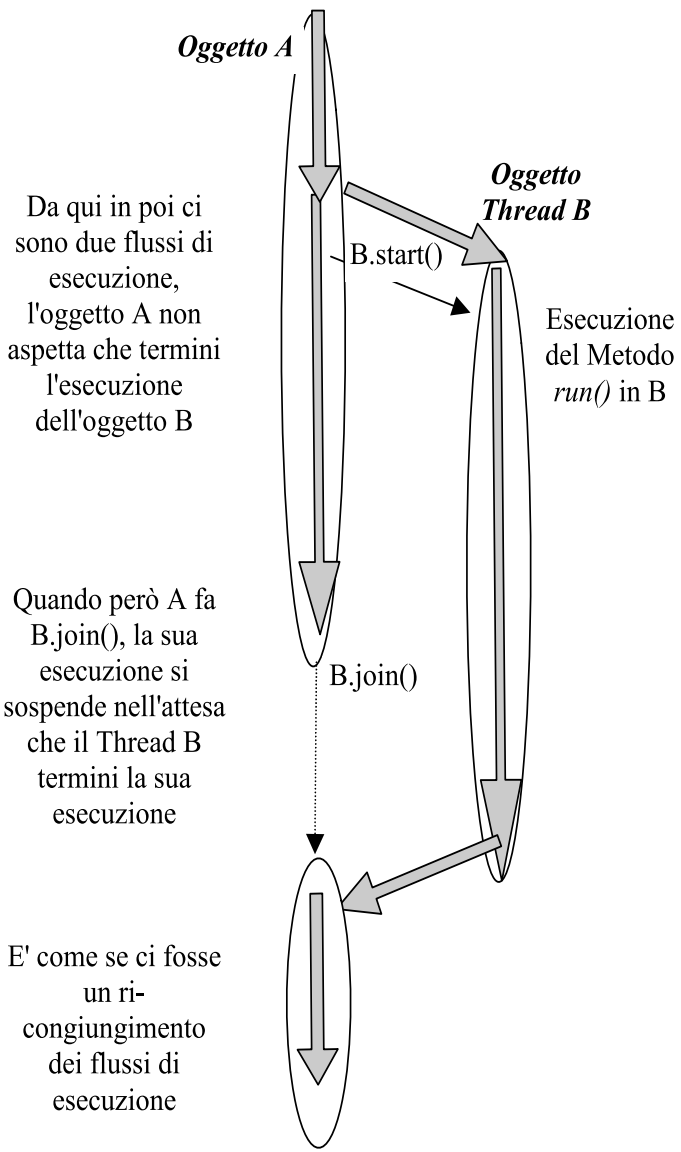
sleep(), suspend() e resume()

- I primi due metodi portano il thread nello stato not runnable

- sleep() per un periodo di tempo prefissato
- suspend() sospende un thread
- resume() risveglia un thread sospeso da suspend(), portandolo nello stato runnable

Join()

- Si può decidere di aspettare che termini l'esecuzione di un thread
- Per fare questo, bisogna invocare l'operazione join() sul thread che si intende aspettare



Esempio di join()

- Il programma principale genera un thread e gli passa un riferimento ad un oggetto StringBuffer
- Il thread scrive nello StringBuffer la data corrente
- Il programma principale aspetta che il thread finisca e poi stampa il valore dello StringBuffer

Esempio di join

Scaricare Esercitazione8-es3 dal server SVN

Note sull'esempio

- L'attesa di join() potrebbe essere interrotta da un'eccezione
  - necessità di catturare l'eccezione tramite il costrutto try/catch (si veda più avanti)
  - necessità analoghe per il metodo sleep()
- Se non avessimo usato join(), come sarebbe stato il risultato dell'esecuzione?

Costrutto synchronized

- In Java si può garantire l'accesso in mutua esclusione a una istanza, proteggendo il metodo o la sezione di codice critica tramite la keyword synchronized:
  - a livello di metodo
  - a livello di blocco di codice

Metodi

1. Metodo sincronizzato:

```
public synchronized void conta() {...}
```

- Solo un thread alla volta può eseguire questo metodo sullo stesso oggetto

2. Blocco di codice sincronizzato:

```
synchronized(object) {...}
```

- Solo un thread alla volta può eseguire la parte di codice protetta sull'oggetto object (che può essere this)

Wait e notify

- Java mette a disposizione due metodi (di Object) per la sospensione e il risveglio dei thread
- wait() (→ wait in pseudo pascal)
  - Sospende il thread che lo invoca su una coda associata all'oggetto sul quale il metodo è invocato
- notify() (→ signal in pseudo pascal) e notifyAll()
  - Risveglia il primo thread (o tutti se notifyAll()) sospeso sulla coda dell'oggetto su cui viene invocato il metodo
- **NB: wait() e notify() possono essere invocati soltanto all'interno di un metodo o blocco synchronized**

Esempio con wait e() notify()

Scaricare Esercitazione8-es4 dal server SVN