Concurrency

`NOTE`

- Usiamo JAVA8 come versione.
  Quando viene cercato su internet un metodo o una classe, fare attenzione che la docuentazione si riferisca alla versione giusta.

Astrazioni di alto livello

In una versione semplice di quello che abbiamo visto in Example01, il livello di astrazione in cui separavamo `Notifier` e `Waiter` non era molto alto.
Programmare vicino alla macchina, vicino al SO, ha vantaggio se quello che andiamo a fare è critico, causa quantità risorse di calcolo elevate, non lo è tuttavia sempre.

JAVA permette di affrontare problemi di concorrenza fornendo **astrazioni** di *alto livello*, perché non possiamo sempre pensare a risolvere problemi di sincronizzazione. Ci serve qualcosa abbastanza generale da poter usare spesso, che sfrutti bene il sistema a disposizione e che permetta al nostro software di acquisire nuove funzionalità.

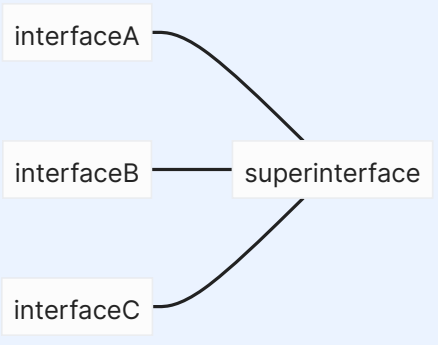
Blocking queue

Esiste una coda per accodare thread.
Inseriamo elementi o togliamo elementi dalla nostra coda, in modo FIFO.

- *creazione*, usando la classe, manipolandola con l'interfaccia;
- *distruzione* della coda;
- *is empty*;
- *is full*, ritorna di solito sempre `TRUE` (lunghezza arbitraria, lunghezza limitata);
- *enqueue* per mettere in coda;
- *dequeue* per togliere dalla coda.

La coda bloccante ha in più che se piena si metterà in attesa, aspettando finché lo spazio non si crea, oppure se vuota la dequeue si bloccherà.

Nota sulle interfacce e super-interfacce



Una super-interfaccia necessita che tutti i metodi delle interfacce che la compongono, venghino implementati. Per esempio: `BlockingQueue<E>` è interfaccia della super-interfaccia `Queue<E>`.

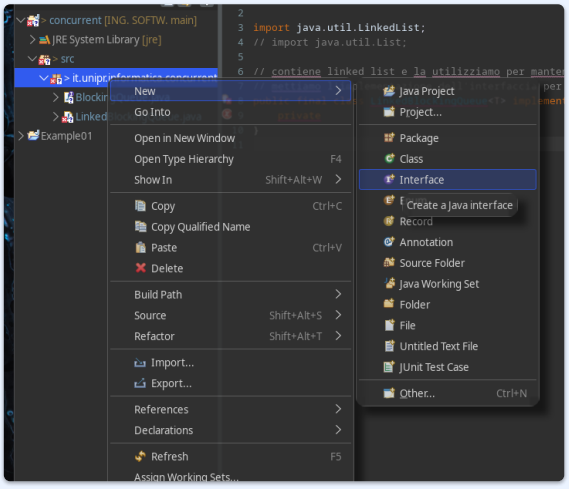
`BlockingQueue<E>`

[BlockingQueue documentation Oracle]

La nostra `BlockingQueue` ha argomento `<E>` e ha delle super-interfacce (che non ci interessano). Dei metodi che esistono, guardiamo solo:

- `void put(T e)`, per aggiungere aspettando nel caso non ci sia spazio
  - lancia quindi `InterruptedException` se non c'è;
- `public T take()` rimuove la testa della coda e se non la contiene, aspetta
  - lancia quindi `InterruptedException` se non c'è;
- `int remainingCapacity()` ritorna quanto spazio ci rimane nella coda se definito (altrimenti `MAX_VALUE`);
- `boolean isEmpty()` verifica in modo sincrono se la coda è vuota;
- `void clear()` per liberare tutte le reference che l'oggetto ha, senza garanzia che gli oggetti vengano liberati per davvero.

Creazione `interface` in Eclipse




```java
package it.unipr.informatica.Concurrency;

public interface BlockingQueue<T> {
        // aggiungo
        public void put(T e) throws InterruptedException;
        // rimuovo
        public T take() throws InterruptedException;
        // elementi disponibili di coda limitata superiormente
        public int remainingCapacity();
        // verifichaimo la coda vuota o meno
        public boolean isEmpty();
        // eliminiamo reference alla coda
```

```
        public void clear();
}
```

`put()`

```
@Override
public void put(T object) {
        synchronized (queue) {
                queue.addLast(object);

        if (queue.size() == 1)
                queue.notify();
        }
}
```

La `notifyAll()` si potrebbe usare anziché `notify()`, ma siccome facciamo un controllo per verificare se la coda ha qualcuno già al suo interno, non è necessaria.

`take()`

```
@Override
public T take() throws InterruptedException {
        synchronized (queue) {
                while (queue.size() == 0)
                        queue.wait();

                T object = queue.removeFirst();

                if (queue.size() > 0)
                        queue.notify();

                return object;
        }
}
```

`remainingCapacity()`

```
@Override
public int remainingCapacity() {
        return Integer.MAX_VALUE;
}
```

`isEmpty()`

```
@Override
public boolean isEmpty() {
        synchronized (queue) {
                return queue.isEmpty();
        }
}
```

`clear()`

```
@Override
public void clear() {
        synchronized (queue) {
                queue.clear();
        }
}
```

La distruzione non avviene veramente, dovremmo mettere a `NULL` il riferimento alla `queue`, ma noi lo lasciamo nel caso serva ancora ad altri oggetti.