Lezione n.2b
LPR-A-09
Threads: Callable & Future

6/10/2008

Vincenzo Gervasi

CALLABLE E FUTURE

- Un oggetto di tipo Runnable incapsula un'attività che viene eseguita in modo asincrono
- Una Runnable si può considerare un metodo asincrono, senza parametri e che non restituisce un valore di ritorno
- Per definire un task che restituisca un valore di ritorno occorre utilizzare le seguenti interfacce:
  - **Callable**: per definire un task che può restituire un risultato e sollevare eccezioni
  - **Future**: per rappresentare il risultato di una computazione asincrona. Definisce metodi per controllare se la computazione è terminata, per attendere la terminazione di una computazione (eventualmente per un tempo limitato),per cancellare una computazione, ...
- La classe **FutureTask** fornisce una implementazione della interfaccia **Future**.

L'INTERFACCIA CALLABLE

```
public interface Callable<V>
    { V call() throws Exception; }
```

L'interfaccia generica **Callable<V>**

- contiene il solo metodo call(), analogo al metodo run( ) della interfaccia Runnable, ma che può restituire un valore e sollevare eccezioni controllate
- per definire il codice del task, occorre implementare il metodo call()
- il parametro di tipo **<V>** indica il tipo del valore restituito
- Esempio: **Callable<Integer>** rappresenta una elaborazione asincrona che restituisce un valore di tipo Integer

CALLABLE: UN ESEMPIO

Definire un task T che calcoli una approssimazione di π, mediante la serie di Gregory-Leibniz (vedi lezione precedente). T restituisce il valore calcolato quando la differenza tra l'approssimazione ottenuta ed il valore di Math.PI risulta inferiore ad una soglia precision. T deve essere eseguito in un thread indipendente.

```
import java.util.concurrent.*;
public class Pigreco implements Callable<Double>{
    private Double precision;
    public Pigreco (Double precision) {this.precision = precision;};
    public Double call ( ){
        Double result = <calcolo dell'approssimazione di π>
            return result;
        }}}
```

L'INTERFACCIA FUTURE

- Per poter accedere al valore restituito dalla Callable, occorre costruire un oggetto di tipo **Future<V>**, che rappresenta il risultato della computazione
- Per costruire un oggetto di tipo **Future** se si gestiscono esplicitamente i threads:
  - si costruisce un oggetto della classe **FutureTask** (che implementa **Future** e **Runnable**) passando un oggetto di tipo **Callable** al costruttore
  - si passa l'oggetto **FutureTask** al costruttore del thread
- Se si usano i thread pools, si sottomette direttamente l'oggetto di tipo **Callable** al pool (con **submit()**) e si ottiene un oggetto di tipo **Future**

L'INTERFACCIA FUTURE

```
public interface Future <V>{
    V get( ) throws ...;
    V get (long timeout, TimeUnit) throws ...;
     void cancel (boolean mayInterrupt);
     boolean isCancelled( );
     boolean isDone( ); }
```

- **get( )** si blocca fino alla terminazione del task e restituisce il valore calcolato
- È possibile definire un tempo massimo di attesa della terminazione del task, dopo cui viene sollevata una **TimeoutException**
- E' possibile cancellare il task e verificare se la computazione è terminata oppure è stata cancellata

CALLABLE E FUTURE: UN ESEMPIO

```
import java.util.*;
import java.util.concurrent.*;
public class FutureCallable {
public static void main(String args[])
      double precision = ……..;
      Pigreco pg = new Pigreco(precision);
      FutureTask <Double> task= new FutureTask <Double>(pg);
      Thread t = new Thread(task);
      t.start();
```

CALLABLE E FUTURE: UN ESEMPIO

```
try{
 double ris = task.get(1000L, TimeUnit.MILLISECONDS);
 System.out.println("valore di isdone" + task.isDone());
 System.out.println(ris + "valore di pigreco");
 }
catch(ExecutionException e) { e.printStackTrace();}
catch(TimeoutException e)
    { e.printStackTrace();
    System.out.println("tempo scaduto");
    System.out.println("valore di isdone" + task.isDone());}
catch(InterruptedException e){   } }  }
```

THREAD POOLING CON CALLABLE

- E' possibile sottomettere un oggetto di tipo Callable<V> ad un thread pool mediante il metodo **submit( )**
- Il metodo restituisce direttamente un oggetto O di tipo Future<V>, per cui non è necessario costruire oggetti di tipo FutureTask
- E' possibile applicare all'oggetto O tutti i metodi visti nei lucidi precedenti

THREAD POOLING CON CALLABLE

```
import java.util.*;
import java.util.concurrent.*;
public class futurepools {
public static void main(String args[])
      ExecutorService pool = Executors.newCachedThreadPool ( );
      double precision = .....;
      pigreco pg = new pigreco(precision);
      Future <Double> result = pool.submit(pg);
      try{ double ris = result.get(1000L, TimeUnit.MILLISECONDS);
      System.out.println(ris+"valore di pigreco");}
      catch(.....){ }.....}}
```