Mises en œuvre d'Active Object

Version Oracle (Java)

- La base : Executor
  - Executor est une interface pour lancer des tâches
  - ExecutorService enrichit Executor pour permettre un meilleur contrôle
  - ScheduledExecutorService rajoute le temps

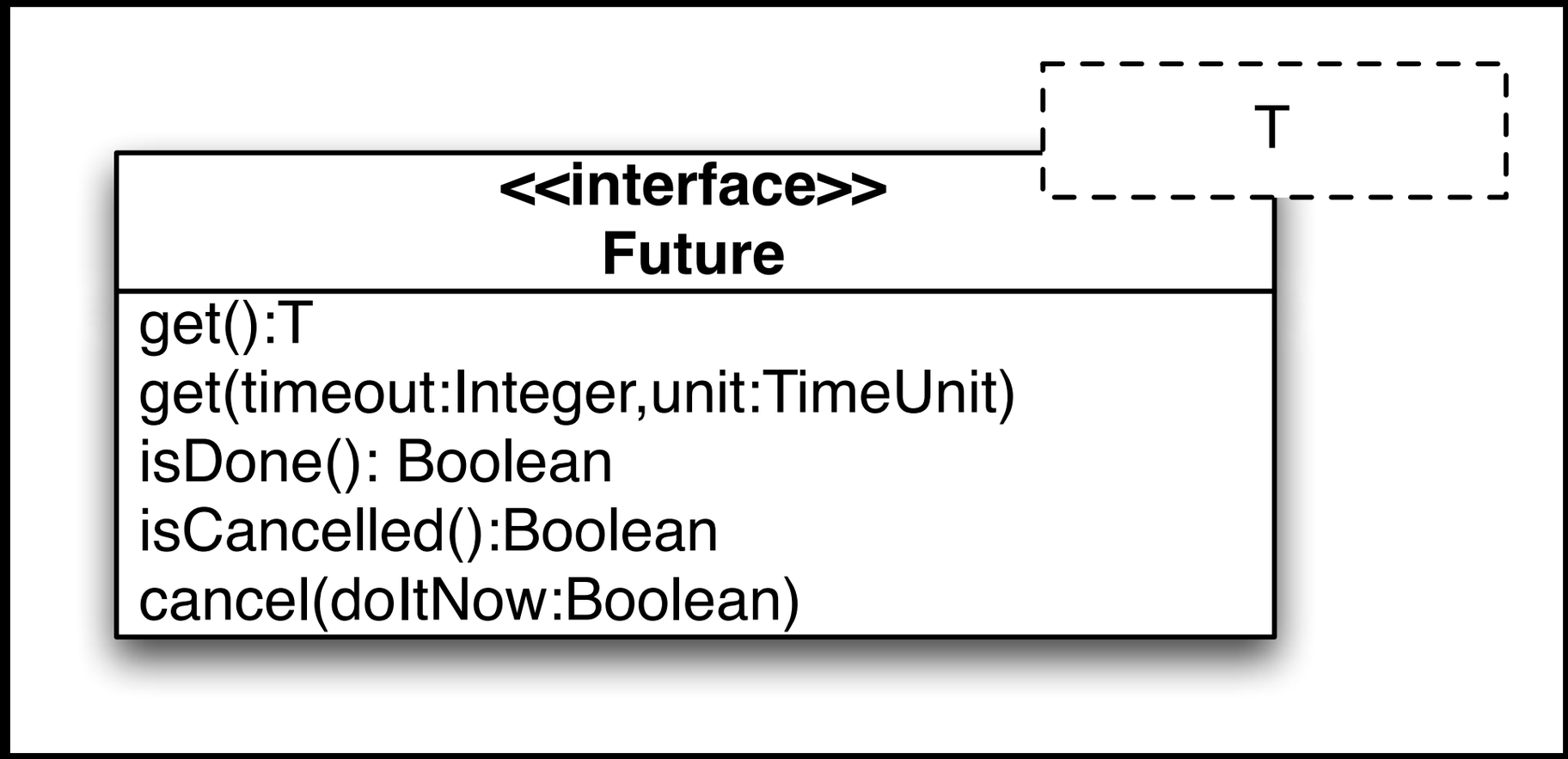
Lancement d'une tâche

- Executor e = ...; // Voir plus loin
- Runnable r = ...; // Idem
- e.execute(r)
  - pas de création explicite de thread
  - le Runnable est passé à une file d'exécution
  - les détails dépendent de la mise en œuvre de Executor

ExecutorService

- Extension de Executor
- Permet de mieux contrôler l'exécution (abandon)
- Permet de démarrer des Callable qui gèrent la notion de Future au sens Active Object

Future


T
<<interface>>
Future
get():T
get(timeout:Integer,unit:TimeUnit)
isDone(): Boolean
isCancelled():Boolean
cancel(doItNow:Boolean)

Mise en œuvre de Future

- La classe FutureTask fournit une mise en œuvre de départ
- Elle dispose
  - d'une opération set pour la valeur
  - d'une opération setException pour signaler une exception d'exécution

Callable

- Une interface simple
- correspond à Method Invocation pour Active Object

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

Mise en œuvre de Callable

- Typiquement par une classe anonyme

```
Callable<String> monCallable =
    new Callable<String>() {
            public String call() {
            return ("Hello world!");
    }
```

Exemple ExecutorService

```
interface ExecutorService {
        Future<T> submit(Callable<T> tache);
        List<Future<T>>
        invokeAll(Collection<? extends
        Callable<T>);
}
```

Exemple d'appel d'ExecutorService

```
ExecutorService exec =
Executors.newFixedThreadPools(10);
Future<String> resultat =
exec.submit(monCallable);
// Autre calcul... le temps passe
String valeurRetour = resultat.get();
```

Autres interfaces Executor

- interface Executor
- version simplifiée
- opération execute(…) qui prend un Runnable au lieu d'un Callable (no Future)

Autres interfaces Executor (2)

- interface ScheduledExecutorService
  - permet d'invoquer un Callable après un certain délai
  - permet de faire un exécution périodique
- ScheduledFuture<V> schedule(Callable<V> c, long delay, TimeUnit unit)
  - execSvc.schedule(c, 10000, TimeUnit.MILLISECONDS)

Création des Executors

- Fabrique Executors : class Executors
  - static ExecutorService newCachedThreadPool()
    - gère dynamiquement un pool de threads (avec recyclage et élimination après délai)
  - static ExecutorService newFixedThreadPool(int nbThreads)

Création des Executors (2)

- static ExecutorService newSingleThreadExecutor()
- création d'une file « séquentielle »

Et la synchronisation ?

- Le parallélisme ne dispense pas de synchronisation
- Emploi de structures spécifiques
  - BlockingQueue
  - Lock

Différences entre mise en œuvre Sun et le PC Active Object

- Pas de Proxy
- Juste une gestion de pool de threads