

Enregistrement des ServiceTracker

- Pour chaque service requis par un composant (bundle) B
 - On enregistre dans la classe d'activation de B, le ServiceTracker correspondant
 - On ouvre le tracker pour qu'il commence à travailler (à traquer les services)

```

14
15 public class Activateur implements BundleActivator, ServiceTrackerCustomizer<HelloWorld, HelloWorld> {
16
17     private BundleContext context;
18     private HelloWorld service;
19     private ServiceReference<HelloWorld> ref;
20     private ServiceTracker<HelloWorld, HelloWorld> tracker;
21
22     public void start(BundleContext bundleContext) throws Exception {
23         context = bundleContext;
24         ref = null;
25         service = null;
26         tracker = new ServiceTracker<>(context, HelloWorld.class, this);
27         tracker.open();
28         chercherService();
29     }
30
31     public void stop(BundleContext bundleContext) throws Exception {
32         if (service != null) {
33             service.goodbye();
34             service = null;
35             context.ungetService(ref);
36             ref = null;
37             tracker.close();
38             tracker = null;
39         }
40         context = null;
41     }
42 }
    
```

Annotations:

- Contexte dans lequel traquer les services (points to `context` in line 26)
- Type des services à traquer ou filtre sur les services à traquer (points to `HelloWorld.class` in line 26)
- Gestionnaire en cas de service trouvé (points to `this` in line 26)
- objet gérant la réception des services traqués (points to `tracker` in line 26)

Dans cet exemple nous avons utilisé la classe `Activateur` comme classe qui implémente l'interface `ServiceTrackerCustomizer<S,T>`

On peut aussi hériter de la classe `ServiceTracker` et faire la même chose.

```

public class Activateur2 extends ServiceTracker<HelloWorld, HelloWorld>
implements BundleActivator{
    
```

```

        public Activateur2(BundleContext context, Class<HelloWorld>
        clazz, ServiceTrackerCustomizer<HelloWorld, HelloWorld> customizer) {
            super(context, clazz, customizer);
        }

        // etc.
    }
    
```

Remarque : la méthode `chercherService()` dans `start` n'est pas nécessaire. La méthode `open` le fait suite à l'envoi d'événements

Définition d'un ServiceTrackerCustomizer

- Un ServiceTrackerCustomizer est appelé lorsque le ServiceTracker associé est tenu au courant d'un événement sur un service qu'il traque

```
6 import org.osgi.framework.BundleContext;
7 import org.osgi.framework.InvalidSyntaxException;
8 import org.osgi.framework.ServiceReference;
9 import org.osgi.util.tracker.ServiceTrackerCustomizer;
10
11 import exemple.contrat.HelloWorld;
12
13 public class HelloWorldTraqueurService implements ServiceTrackerCustomizer<HelloWorld, HelloWorld> {
14     private ServiceReference<HelloWorld> ref;
15     private BundleContext context;
16     private HelloWorld service;
17
18     public HelloWorldTraqueurService(BundleContext bc, ServiceReference<HelloWorld> sr, HelloWorld s) {
19         context = bc; ref = sr; service = s;
20     }
21
22     public HelloWorld addingService(ServiceReference<HelloWorld> sr) {}
23
24     public void modifiedService(ServiceReference<HelloWorld> sr, HelloWorld service) {}
25
26     public void removedService(ServiceReference<HelloWorld> sr, HelloWorld service) {}
27
28     private void chercherService() {}
29 }
```

On peut aussi définir une nouvelle classe pour implémenter l'interface ServiceTrackerCustomizer<S,T>

On peut aussi hériter de la classe ServiceTracker et faire la même chose.

Dans l'activateur, il faut créer l'objet instance de cette classe avant de créer le tracker :

```
public void start(BundleContext bundleContext) throws Exception {
    Activateur2.context = bundleContext;
    ref=null;
    service=null;
    tracker = new ServiceTracker<>(context, HelloWorld.class, new
    HelloWorldTraqueurService(context, ref, service));
    tracker.open();
}
```