useFactory

```
export const ENV = new InjectionToken<string>('Environment')
  ...
export function dataServiceFactory(environment: string): DataService {
  ...
}
```

```
{
  provide: DataService,
  useFactory: dataServiceFactory,
  deps: [ENV]
},
```

- useFactory permet de définir un callback pour instancier notre dépendance
- En utilisant deps, on peut spécifier des paramètres pour notre callback. deps est constitué de tokens qui sont résolus et passés dans l'ordre au callback
- On peut utiliser une factory pour fournir des implémentation différentes pour un même service, selon un paramètre (penser au pattern factory)

- Il est possible de définir un provider au niveau de l'environment injector lors de la déclaration d'un token

```
export const HELLO_TOKEN = new InjectionToken<string>('bonjour', {
  providedIn: 'root',
  factory: () ⇒ 'Bonjour'
})
```

- factory est un callback, qui est appelée pour instancier la dépendance
- Il n'est pas possible de spécifier des paramètres à factory

Modification de la recherche

- Du coté du consumer, il est possible de modifier la recherche de provider

- @Optional() : Retourne null plutot qu'une exception si la dépendance n'est pas résolue

```
@Optional() public firstCounterService: CounterService,
```

```
firstCounterService = inject(CounterService, {optional: true} )
```

- inject() renvoie un union type CounterService | null

Modification de la recherche

- Du coté du consumer, il est possible de modifier la recherche de provider

- @SkipSelf : Saute l'élément courant

```
@SkipSelf() public firstCounterService: CounterService,
```

```
firstCounterService = inject(CounterService, {skipSelf: true} )
```

Modification de la recherche

- Du coté du consumer, il est possible de modifier la recherche de provider

- @Self : Arrête la recherche à l'élément courant

```
@Self() public firstCounterService: CounterService,
```

```
firstCounterService = inject(CounterService, {self: true} )
```

Modification de la recherche

- Du coté du consumer, il est possible de modifier la recherche de provider

- @Host : Arrête la recherche à l'élément hôte (pour les directives et contenu projeté)

```
@Host() public firstCounterService: CounterService,
```

```
firstCounterService = inject(CounterService, {host: true} )
```

Exercices

- Sur le composant fils, partager le premier compteur avec les autres composants, et avoir le deuxième indépendant

- Inverser la résolution de dépendances pour premier compteur du composant fils :
  - Un provider est cherché dans les composants pères
  - Si aucun provider n'est trouvé utiliser un provider par défaut dans le composant fils

- Reprendre le composant répertoire et ajouter une méthode qui affiche le chemin complet de chaque composant