# Migration for missing `@Injectable()` decorators and incomplete provider definitions

## What does this schematic do?

1. This schematic adds an `@Injectable()` decorator to classes which are provided in the application but are not decorated.
2. The schematic updates providers which follow the `{provide: SomeToken}` pattern to explicitly specify `useValue: undefined`.

**Example for missing `@Injectable()`**

*Before migration:*

```
export class MyService {...}
export class MyOtherService {...}
export class MyThirdClass {...}
export class MyFourthClass {...}
export class MyFifthClass {...}

@NgModule({
  providers: [
    MyService,
    {provide: SOME_TOKEN, useClass: MyOtherService},
    // The following classes do not need to be decorated because they
    // are never instantiated and just serve as DI tokens.
    {provide: MyThirdClass, useValue: ...},
    {provide: MyFourthClass, useFactory: ...},
    {provide: MyFifthClass, useExisting: ...},
  ]
})
```

*After migration:*

```
@Injectable()
export class MyService {...}
@Injectable()
export class MyOtherService {...}
export class MyThirdClass {...}
export class MyFourthClass {...}
export class MySixthClass {...}

...
```

Note that `MyThirdClass`, `MyFourthClass` and `MyFifthClass` do not need to be decorated with `@Injectable()` because they are never instantiated, but just used as a DI token.

**Example for provider needing `useValue: undefined`**

This example shows a provider following the `{provide: X}` pattern. The provider needs to be migrated to a more explicit definition where `useValue: undefined` is specified.

*Before migration*:

```
{provide: MyToken}
```

*After migration*:

```
{provide: MyToken, useValue: undefined}
```

## Why is adding `@Injectable()` necessary?

In our docs, we've always recommended adding `@Injectable()` decorators to any class that is provided or injected in your application. However, older versions of Angular did allow injection of a class without the decorator in certain cases, such as AOT mode. This means if you accidentally omitted the decorator, your application may have continued to work despite missing `@Injectable()` decorators in some places. This is problematic for future versions of Angular. Eventually, we plan to strictly require the decorator because doing so enables further optimization of both the compiler and the runtime. This schematic adds any `@Injectable()` decorators that may be missing to future-proof your app.

## Why is adding `useValue: undefined` necessary?

Consider the following pattern:

```
@NgModule({
  providers: [{provide: MyService}]
})
```

Providers using this pattern will behave as if they provide `MyService` as DI token with the value of `undefined`. This is not the case in Ivy where such providers will be interpreted as if `useClass: MyService` is specified. This means that these providers will behave differently when updating to version 9 and above. To ensure that the provider behaves the same as before, the DI value should be explicitly set to `undefined`.

## When should I be adding `@Injectable()` decorators to classes?

Any class that is provided must have an `@Injectable()` decorator. The decorator is necessary for the framework to properly create an instance of that class through DI.

However, classes which are already decorated with `@Pipe`, `@Component` or `@Directive` do not need both decorators. The existing class decorator already instructs the compiler to generate the needed information.

## Should I update my library?

Yes, if your library has any classes that are meant to be injected, they should be updated with the `@Injectable()` decorator. In a future version of Angular, a missing `@Injectable()` decorator will always throw an error.

Additionally, providers in your library that follow the described `{provide: X}` pattern should be updated to specify an explicit value. Without explicit value, these providers can behave differently based on the Angular version in applications consuming your library.