Sharing modules

Creating shared modules allows you to organize and streamline your code. You can put commonly used directives, pipes, and components into one module and then import just that module wherever you need it in other parts of your application.

Consider the following module from an imaginary app:

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { CustomerComponent } from './customer.component';
import { NewItemDirective } from './new-item.directive';
import { OrdersPipe } from './orders.pipe';

@NgModule({
 imports:      [ CommonModule ],
 declarations: [ CustomerComponent, NewItemDirective, OrdersPipe ],
 exports:      [ CustomerComponent, NewItemDirective, OrdersPipe,
                 CommonModule, FormsModule ]
})
export class SharedModule { }
```

Note the following:

- It imports the `CommonModule` because the module's component needs common directives.
- It declares and exports the utility pipe, directive, and component classes.
- It re-exports the `CommonModule` and `FormsModule`.

By re-exporting `CommonModule` and `FormsModule`, any other module that imports this `SharedModule`, gets access to directives like `NgIf` and `NgFor` from `CommonModule` and can bind to component properties with `[(ngModel)]`, a directive in the `FormsModule`.

Even though the components declared by `SharedModule` might not bind with `[(ngModel)]` and there may be no need for `SharedModule` to import `FormsModule`, `SharedModule` can still export `FormsModule` without listing it among its `imports`. This way, you can give other modules access to `FormsModule` without having to import it directly into the `@NgModule` decorator.

More on NgModules

You may also be interested in the following:

- [Providers](#).
- [Types of Feature Modules](#).