Course Schedule

- Angular Intro
- Components
- Templates and Directives
- Data Binding and Pipes
- Components Revisited
- Nested Components
- **Services and Dependency Injection**
- Http Service
- Navigation and Routing
- Routing Techniques
- Angular Modules

- **What are Services?**
- **Working with Services**
- **Create Simple Service**
- **Using Services**
- **Injecting Services**
- **What is DI?**
- **Non Dependency Injection**
- **Class with DI**
- **Dependency Injection**
- **@Inject**
- **@Injectable**
- **Registering the Service**
- **Registering a Provider**
- **Injecting the Service**
- **Service Checklist**

What are Services?

- **Functions or objects available to Angular application :**
  - Reusable business logic components independent of views
  - Wired together using dependency injection (DI)
- **Angular has about built-in services which can be injected :**
  - Many applications also want to create their own
- **Class with a focused purpose used for features that :**
  - Are independent from any particular component
  - Provide shared data or logic across components
  - Encapsulate external interaction
- **Angular services are :**
  - Lazily instantiated :
    - Angular only instantiates a service when an application component depends on it
  - Singletons :
    - Each component gets reference to single instance generated by service factory
- **To use Angular service add dependency for the component**

Working with Services

- **Applications often require services :**
  - Such as a product data service or a logging service.
- **Implement functionality independent of particular component :**
  - Share data or logic across components
  - Sncapsulate external interactions, such as data access
- **By shifting responsibilities from component to a service :**
  - Code is easier to test, debug and reuse
- **Component can create instance of service class and use it :**
  - Instance is local to the component, so we can't share data or other resources
- **By registering service with Angular singleton is created**

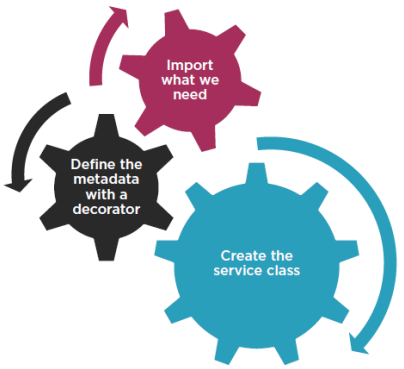
Create Simple Service

- **Create file** `simpleservice.service.ts` **:**

```
import { Injectable } from '@angular/core';

@Injectable()
export class SimpleService {
    // This is where methods and properties go, for example:
    serviceMethod() {
        return 'Hello from Service';
    }
}
```

- **To create a service :**
  - Create service class, define metadata with decorator, and import what we need

```
import { Injectable } from '@angular/core';
import { IProduct } from './product';
@Injectable()
export class ProductService {
     getProducts(): IProduct[] {
          return;
     }
}
```



Using Services

- **Service can be imported directly in components :**

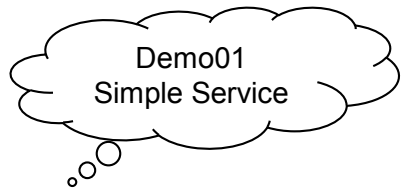
```
import { Component } from '@angular/core';
import { SimpleService } from './simpleservice.service';
```

- **Alternatively can import it to the `app.module.ts` file :**
  - Give all of your components access to that service
- **Add it to providers array in Component decorator metadata :**

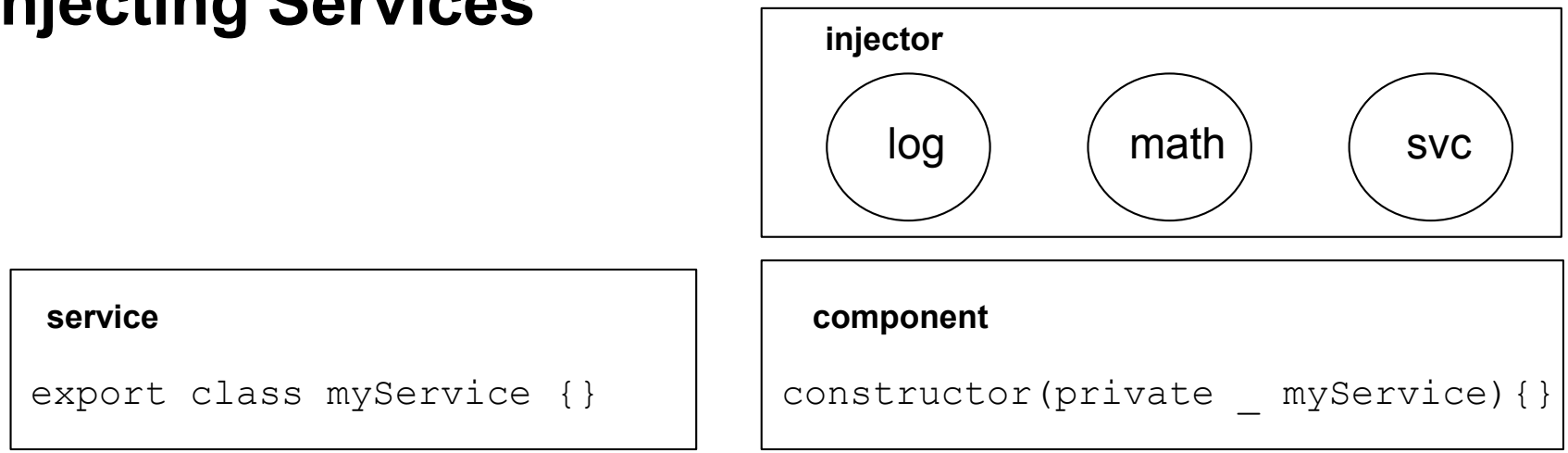
```
@Component({
    selector: 'my-app',
    template: '<h1>{{ title }}</h1>',
    providers: [SimpleService]
})
```

- **In constructor arguments of the component class :**
  - Include it through dependency injection

```
constructor(private _simpleService: SimpleService) { }
```



Injecting Services



- **We register services with Angular's built-in injector :**
  - Maintains a container of created service instances
  - Creates and manages singleton of each registered service
- **If component needs a service :**
  - Component class defines service as a dependency
  - Injector then injects service class instance when component class is instantiated
  - Process is called dependency injection
- **Since Angular manages the single instance :**
  - Any data or logic in that instance is shared by all classes that use it

Demo02
SharedService

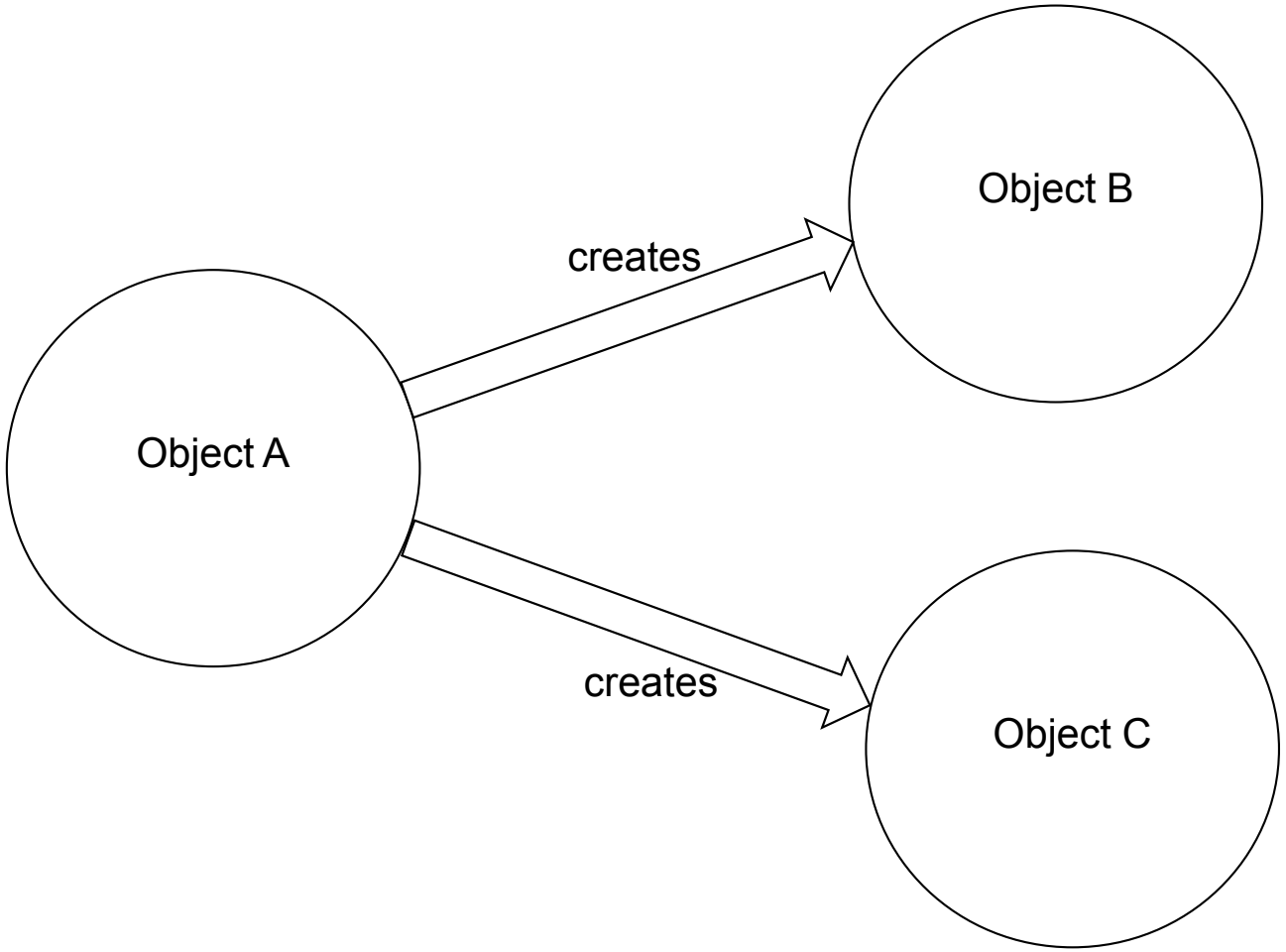
What is DI?

- **Following class does not use dependency injection :**

```
class Hamburger {
  private bun: Bun;
  private patty: Patty;
  private toppings: Toppings;
  constructor() {
    this.bun = new Bun('withSesameSeeds');
    this.patty = new Patty('beef');
    this.toppings = new Toppings(['lettuce', 'pickle', 'tomato']);
  }
}
```

- **Class assumes :**
  - Hamburger consists of a Bun, Patty and Toppings
  - Class is also responsible for making the Bun, Patty and Toppings
- **This is a bad thing :**
  - What if a vegetarian burger were needed?
  - Or a gluten free burger
- **Should we create another class for that??**

Non-Dependency Injection



Class with DI

- **Hamburger class could be rewritten as follows :**

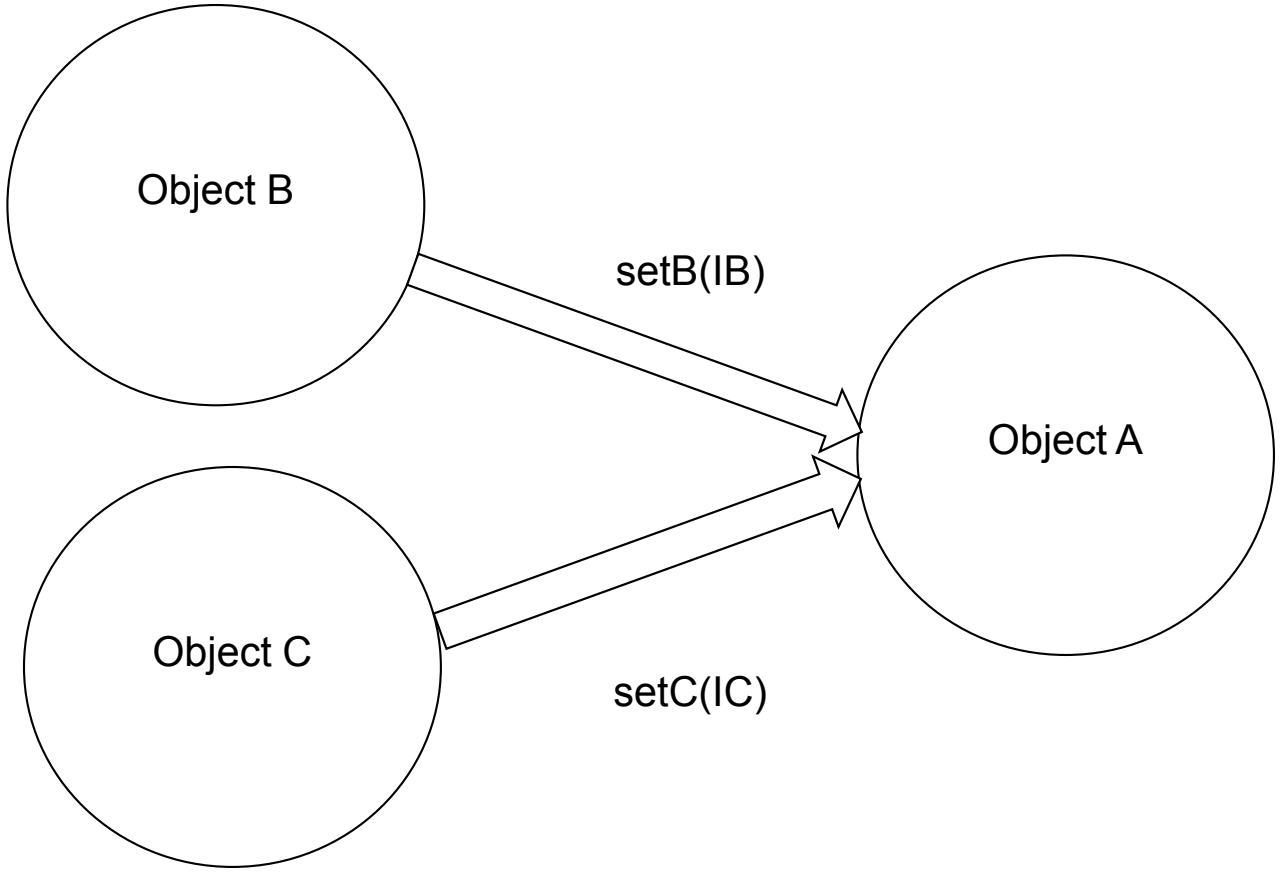
```
class Hamburger {
  private bun: Bun;
  private patty: Patty;
  private toppings: Toppings;

  constructor(bun: Bun, patty: Patty, toppings: Toppings) {
    this.bun = bun;
    this.patty = patty;
    this.toppings = toppings;
  }
}
```

- **Now when Hamburger is instantiated :**
  - Does not need to know anything about its Bun, Patty, or Toppings
  - Construction of these elements has been moved out of the class
- **TypeScript allows it to be written in shorthand like so :**

```
class Hamburger {
  constructor(private bun: Bun, private patty: Patty,
    private toppings: Toppings) {}
}
```

Dependency Injection

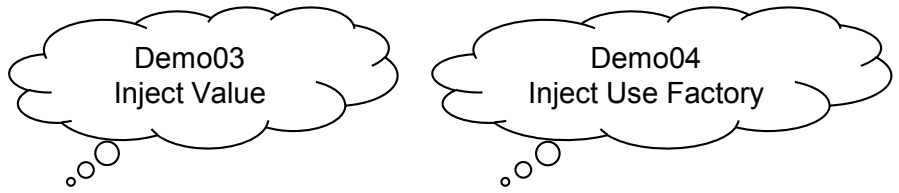
- **Coding pattern in which :**
  - Class receives the instances of objects it needs from an external source
  - Rather than creating them itself



@Inject

- `@Inject()` **is a manual mechanism :**
  - Letting Angular know that a parameter must be injected
- **With TypeScript** `@Inject` **is only needed for injecting primitives :**
  - TypeScript's types let Angular know what to do in most cases
- **In example code random number gets injected :**

```
@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent],
  providers: [{ provide: 'Random', useValue: Math.random()},],
  bootstrap: [AppComponent],
})
…
@Component({
  selector: 'app-root', template: `Random: {{ value }}`})
export class AppComponent {
  value: number;
  constructor(@Inject('Random') r) {
    this.value = r;
  }}
```



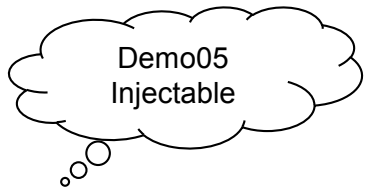
@Injectable

- **Lets Angular know class can be used with dependency injector :**
  - Not strictly required if class has other Angular decorators
  - Important is that any class that is going to be injected with Angular is decorated
- **Best practice is to decorate injectables with `@Injectable()` :**

```
@Injectable()
export class ChatWidget {
  constructor(
    public authService: AuthService,
    public authWidget: AuthWidget,
    public chatSocket: ChatSocket) { }
}
```

- **Other classes of `@Injectable` as well :**

```
@Injectable()
   export class AuthService {}
@Injectable()
export class ChatSocket {
  encryption = true;
}
```



Registering the Service

- **We should register service with an Angular injector :**
  - Injector provides the service instance to any class that defines it as a dependency
- **To register a service, we must register a provider :**
  - Provider is code that can create or return a service, typically the service class itself
- **Defining a provider :**
  - As part of the component or Angular module metadata
- **If registering a provider in a component's metadata :**
  - Angular injector can inject this service into the component and any of its children
  - Take care to register provider at appropriate level of application component tree
- **If we register a provider in an Angular module :**
  - Service is registered with the Angular injector at the application's root
  - Makes the service available everywhere in the application
- **After registration include service through dependency injection :**

```
constructor(private _someService: SomeService) { }
```

Injecting the Service

product-list.component.ts

```
...
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent {
 private _productService;
 constructor(productService: ProductService) {
   _productService = productService;
 }

}
```

app.component.ts

```
...
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-app',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <pm-products></pm-products>
    </div>
    `,
  providers: [ProductService]
})
export class AppComponent { }
```

- **Shortcut form**

product-list.component.ts

```
...
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent {

 constructor(private _productService: ProductService) {
 }

}
```

Service Checklist

- **Service class :**
  - Clear name, Use PascalCasing, Append "Service" to the name, export keyword
- **Service decorator :**
  - Use Injectable ,Prefix with @; Suffix with ()
- **Import what we need**
- **Registering a Service in a Component :**
  - Select the appropriate level in the hierarchy :
    - Root component if service is used throughout the application
    - Specific component if only that component uses the service
    - Otherwise, common ancestor
  - Component metadata :
    - Set the providers property, Pass in an array
- **Dependency Injection :**
  - Specify the service as a dependency
  - Use a constructor parameter
  - Service is injected when component is instantiated

Summary : Services and Dependency Injection

- **Services are functions or objects available to Angular apps :**
  - Contain reusable business logic componentes independent of views
- **Service are wired together using dependency injection (DI) :**
  - Coding pattern in which class receives instances rather than creating them itself
- **Angular has about built-in services which can be injected :**
  - Many applications also want to create their own
- `@Inject()` **decorator is a manual mechanism :**
  - Letting Angular know that a parameter must be injected
- `@Injectable()` **decorator :**
  - Lets Angular know class can be used with dependency injector :
- **If registering provider in component's metadata :**
  - Angular injector can inject this service into the component and any of its children
- **If we registering provider in an Angular module :**
  - Service registered at application's root
  - Available everywhere in application