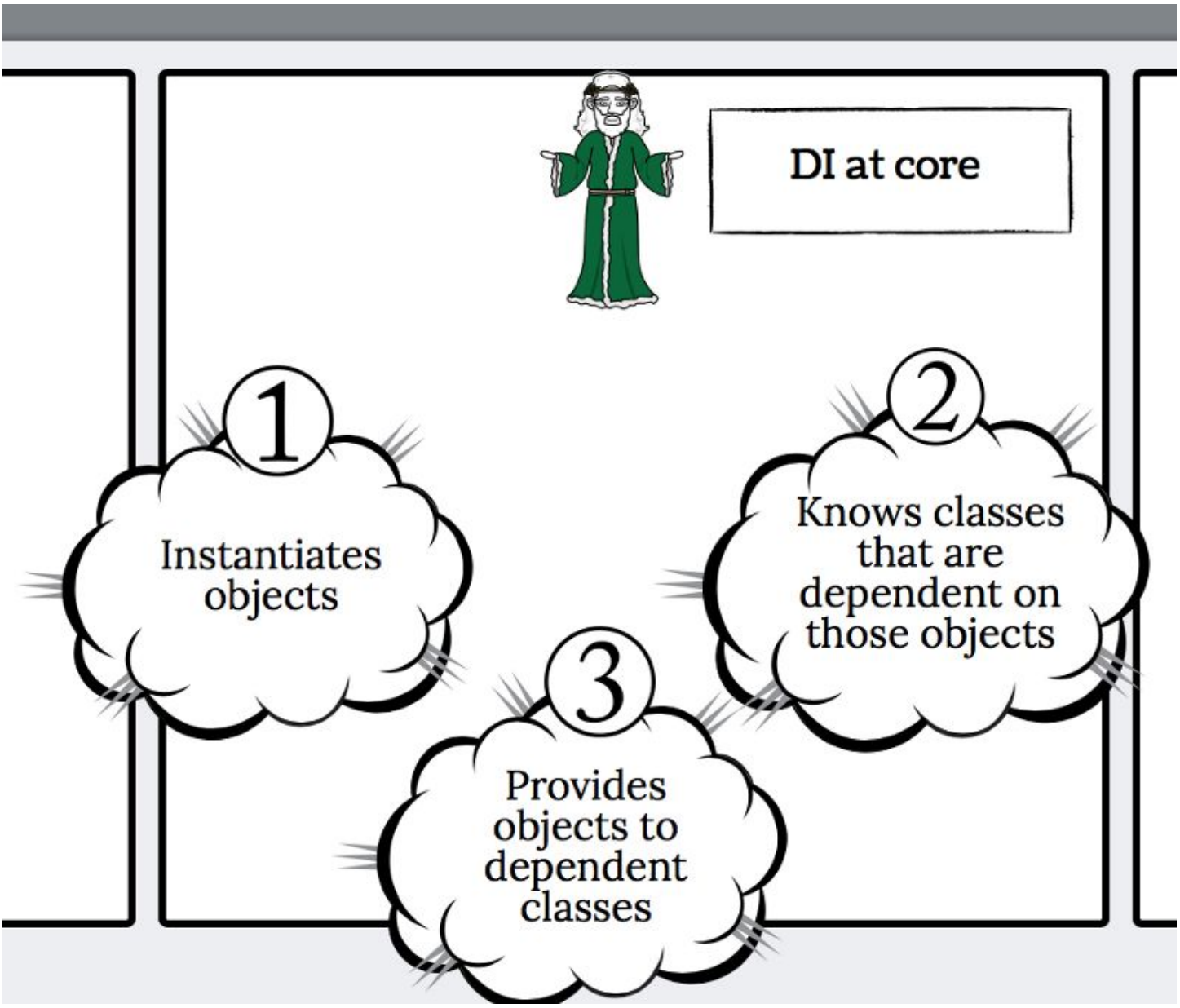
https://android.jlelse.eu/dagger-2-the-simplest-approach-3e23502c4cab

https://www.raywenderlich.com/146804/dependency-injection-dagger-2

Assume the rough overview that Di sets (provides) dependencies to views through setters. So,

1. If anything gets changed in dependencies, it will impact on Di class and not in view classes.
2. View classes will have setter methods so that Di can call those methods to set dependencies there.



Source:
https://medium.com/@laaptu9/dependency-injection-in-android-part-iv-the-android-story-dagger-2-f89215735ecf

Sagar note:

We annotate our pojo or model class by module. Module is part of Di that cares about how dependency will be created.
In module, we can have providers who provides our dependencies. (Constructor or methods). We annotate such providers with sign @Provides.

Providers first gives dependencies to container and then dependents gets their dependencies from container.
So it is like: module/providers -> container (component) -> dependents.
Container means component.

In component, we can explicitly declare our modules as below:

```
@Component(modules = { DataModule.class })
```

Component is an interface. In component, there will be methods to be called by dependents. This way, dependents register themselves to use the dependencies specified by module through this component.
So, component can have some method like:

```
void inject(MainActivity target);
```

...and this method will be called by (in) *MainActivity* so that *MainActivity* will be able to use *DataModule* that will be provided by this *Component.*

..but how the *MainActivity* will get an object of *Component* so that it can call *inject* method?

Well, we do it in Application class.

```
public class MainApplication extends Application {
 private DataComponent dataComponent;

 @Override public void onCreate() {
   super.onCreate();
   /**
    * This part may be confusing
    * at first.
    * If you at first simply write this
    * line, the IDE would throw error as
    * these classes won't be built until
    * and unless you go to Build->RebuildProject.
    * Once you do that, go to
    * app/build/generated/source/apt/...
    * You will see these generated class
    * and it won't throw any error on IDE.
    * DataComponent= name of our component.
```

```
     * Dagger by default create the component as
     * DaggerDataComponent.
     * dataModule() method simply means your are
     * trying to use DataModule class as defined
     * in the @Component(modules = { DataModule.class }).
     * So if there is another module named HelloModule
     * and being used by the DataComponent, there
     * will be method named
     * helloModule().
    */
    dataComponent = DaggerDataComponent.builder()
        .dataModule(new DataModule(this)).build();
    /**
     * @Component(modules = { DataModule.class }),
     * simply creates a
     * setter in DaggerDataComponent class as
     * dataModule(DataModule dataModule). Look
     * upon the generated class. If you don't set
     * the DataModule using that method,
     * it will be null i.e. if you won't
     * do as above statement, DataComponent will be created
     * but with null dataModule. So annotating and
     * setting the module is both need for component
     * to function properly
     */
  }

  public DataComponent getDataComponent() {
    return dataComponent;
  }
}
```

Source:
https://medium.com/@laaptu9/dependency-injection-in-android-part-iv-the-android-story-dagger-2-f89215735ecf

After we have instantiated the component, we need to use it in the dependent classes as well in the following manner

```
public class MainActivity extends AppCompatActivity {

 @Inject DataService dataService;

 @Override protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ((MainApplication)getApplication())
       .getDataComponent().inject(this);
 }
}
```

After doing this, the dependent class gets it dependent object as variable. If in future, DataService is to be changed, then the only place it needs to be changed is in the DataModule class or to say module class

Source:
https://medium.com/@laaptu9/dependency-injection-in-android-part-iv-the-android-story-dagger-2-f89215735ecf

Project source:
https://github.com/androidlife/get-a-fix-of-dependency/tree/with_di

One thing that need to be understood in above example is: DataService is an interface.
It is according to Separation concept. That means, the *MainActivity* will interact for *User Data* through *Interface.*
In other words, *User Data* is interacting with View - *MainActivity* through interface.

https://medium.com/@harivigneshjayapalan/dagger-2-for-android-beginners-dagger-2-part-i-f2de5564ab25

Purpose: Reusability, test and maintenance

Dagger will only look for methods annotated with @Provides

A class should get it's dependency from configuration class and should never create new one.

@Inject means dependencies to provide.

```
@Inject
public War(Starks starks, Boltons bolton){
    this.starks = starks;
    this.boltons = bolton;
}
```

https://google.github.io/dagger/users-guide

Classes that lack @Inject annotations cannot be constructed by Dagger.

If your class has @Inject-annotated fields but no @Inject-annotated constructor, Dagger will inject those fields if requested, but will not create new instances. Add a no-argument constructor with the @Inject annotation to indicate that Dagger may create instances as well.

More to read on @Inject:

https://stackoverflow.com/questions/43287645/dagger-2-injecting-constructors?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

@Component: Who provides dependency/ies

We define the interface, annotate it with @Component and we declare the method there that will return our dependency.

```
1  @Component
2  interface BattleComponent {
3      War getWar();
4  }
```

Somewhere:

```
BattleComponent component = DaggerBattleComponent.create();
War war = component.getWar();
```

@Provide annotation is used for the method or when @Inject cannot be used.

https://google.github.io/dagger/users-guide

- Interfaces can’t be constructed.
- Third-party classes can’t be annotated.
- Configurable objects must be configured!

@Module annotation is used for the class that provides dependency. If anything in class has @Provide annotation, then @Module annotation is must for that class.

https://github.com/codepath/android_guides/wiki/Dependency-Injection-with-Dagger-2

http://www.vogella.com/tutorials/Dagger/article.html

If we use @Provide for the method that has parameter and if that parameter itself a dependency and if the constructor of that parameter has @Inject annotation, then dagger can provide that dependency on it's own for the @Provide method.

If the provider has parameters but constructor of that parameter has no @Inject annotation, then there must be another module and a provider which should provide that parameters. In such case, @Component can have multiple modules as below:

```
@Component(modules =
{CoffeeProviderOther.class,IngredientsProvider.class})
public interface CoffeeComponentOther {
    void provideCoffee(RestaurantB restaurantB);
}
```

Where:

```
@Module
public class CoffeeProviderOther {
    @Provides
    public CoffeeHelper coffeeHelper(int quantity, Coffee.Flavor
flavor) {
        return new CoffeeHelper(quantity, flavor);
    }

}
```

And because provider of *CoffeeProviderOther* module is dependent on other module and it's provider that can supply required parameters:

```
@Module
public class IngredientsProvider {
    @Provides
    public int quantities() {
        return 10;
    }

    @Provides
    public Coffee.Flavor getFlavor() {
        return Coffee.Flavor.Latte;
    }
}
```

Hence:

```
@Component(modules =
{CoffeeProviderOther.class,IngredientsProvider.class})
public interface CoffeeComponentOther {
    void provideCoffee(RestaurantB restaurantB);
}
```

Source:
https://medium.com/@laaptu9/part-4-simple-ways-to-stab-with-dagger-2-module-dependencies-and-named-providers-a3e27f8d3421

However, if constructor of parameter has no @Inject annotation, we will have to tell the dagger how it can get that parameter and that is the answer of:
When and why do we use @Component (modules = *comma separated modules class*)?

https://medium.com/@laaptu9/part-2-simple-ways-to-stab-with-dagger-2-module-component-and-field-injection-e85cbef8678b

@Module and used @Inject on our classes

```
public class RestaurantA {

    @Inject
    public CoffeeHelper coffeeHelper;
}
public class RestaurantB {

    @Inject
    public CoffeeHelper coffeeHelper;
}
public class HotelB {

    @Inject
    public CoffeeHelper coffeeHelper;
}
```

This annotation = Hey Dagger I need CoffeeHelper object, so please provide me an instance of this

I am the coffeeHelper and I am going to be used as dependency right here. But I don't know how am I initialized!

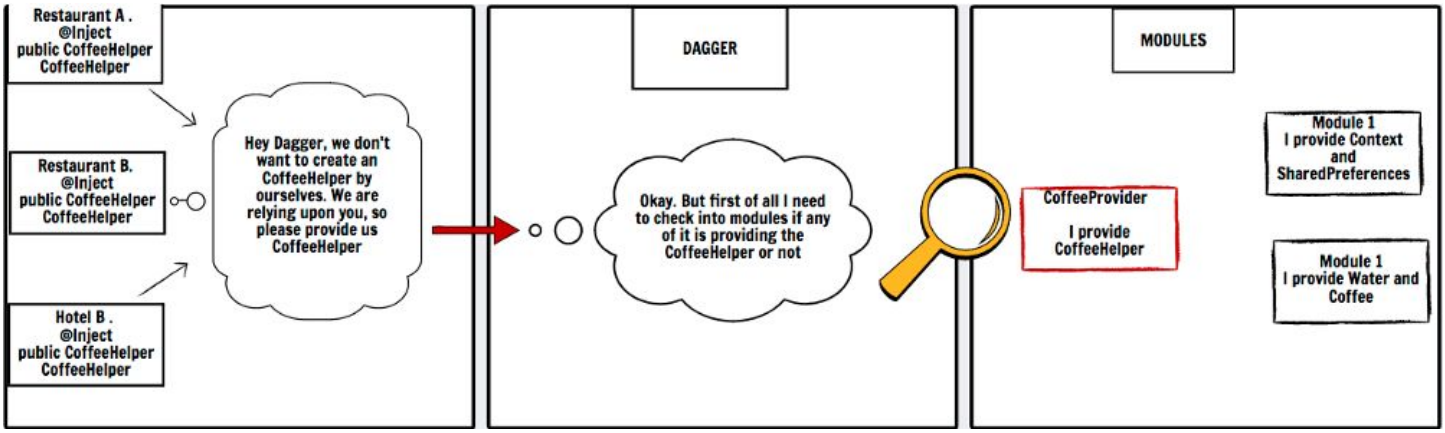
```
@Module
public class CoffeeProvider {

    @Provides
    CoffeeHelper getCoffeeHelper() {
        return new CoffeeHelper();
    }
}
```

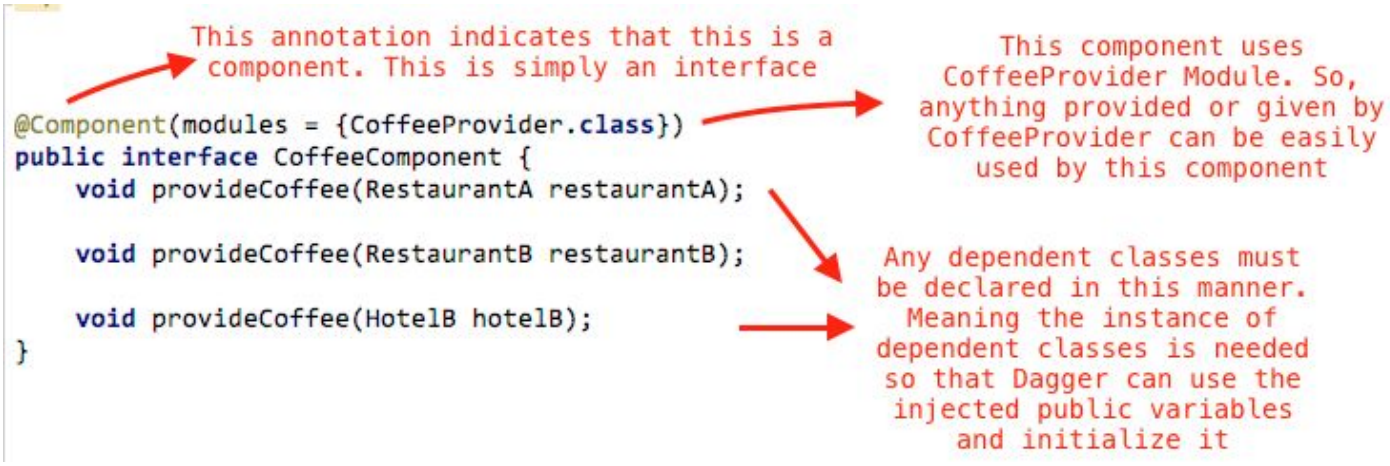
This annotation indicates that this is a module. An application can have any number of modules.

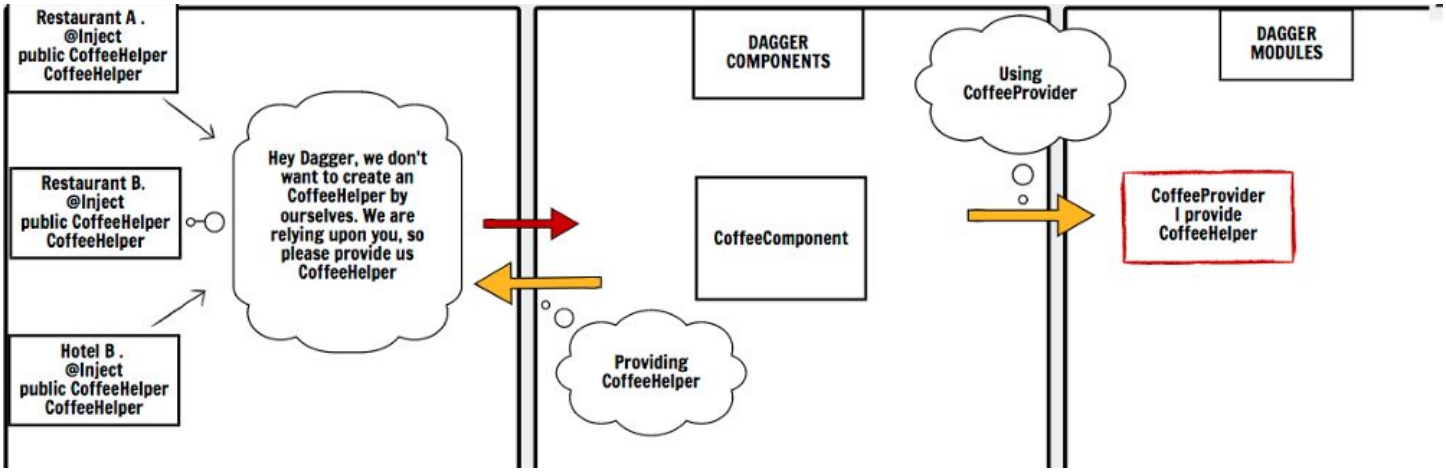
This annotation indicates that this method will provide a way to instantiate an object. In this case, it is providing an instance of CoffeeHelper.

I will provide the instance of coffeeHelper to whom so ever it may concerned!

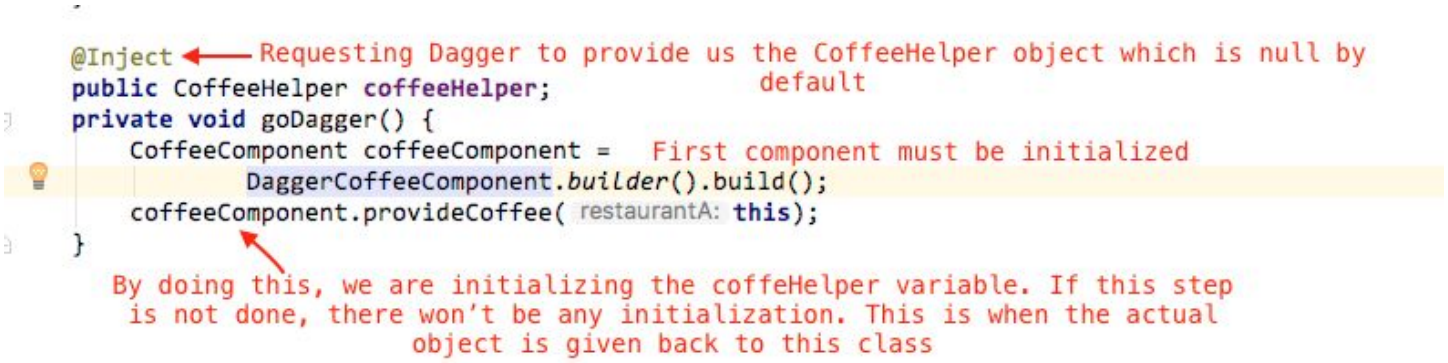


But We can't use the module directly. To use the modules we need ***Component.***





And then we do in each client (who requires dependency, the dependent)



https://blog.mindorks.com/introduction-to-dagger-2-using-dependency-injection-in-android-part-2-b55857911bcd

@Qualifier: To distinguish between or among same type of object but with different instances. Context from Application Vs Context from Activity or say, Application context Vs Activity context.

@Scope: Enables to create global and local singletons. (Source: https://android.jlelse.eu/dagger-2-part-i-basic-principles-graph-dependencies-scopes-3dfd032ccd82)

https://mirekstanek.online/

http://frogermcs.github.io/dependency-injection-with-dagger-2-custom-scopes/

@Subcomponent:
https://proandroiddev.com/dagger-2-component-relationships-custom-scopes-8d7e05e70a37