Let me introduce you to the core component of the laravel that is service container.

What is Service container

Service container is a powerful tool for managing class dependencies and performing dependency injectio n. Its sole responsibility is to manage the dependencies in your laravel project.

What is dependency injection

Dependency Injection means that class dependencies are “injected” into the class via the constructor or, i n some cases, “setter” methods. Mostly constructor is used for injecting the dependency.

Service container is like a container where we define how the dependency should be resolved. We have t o register the dependencies into the service container during the initialization of the framework and the be st place to do it is the service provider.

Number of Ways of Registering the Dependency

Now as we have understood what is service container, what is dependency injection, lets now focus on th e number of ways to register the dependency in service container.

We have to register the dependency in register method of service provider

```
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use App\Services\Contracts\ExampleContract;
use App\Services\NormalDependency;
use App\Services\ViaInterfaceDependency;
use App\Services\SingleTonDependency;



class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->bind(NormalDependency::class,function($app){
            return new NormalDependency();
        });

        $this->app->singleton(SingleTonDependency::class,function($app){
            return new SingleTonDependency();
        });

        $this->app->bind(ExampleContract::class,function($app){
            return new ViaInterfaceDependency();
        });
```

```
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        //
    }
}
```

For Demo purpose I have created NormalDependency, SingleTonDependency, ViaInterfaceDependency class and ExampleContract interface (Name of the class or interface are up to you)

```
$this->app->bind(NormalDependency::class,function($app){
            return new NormalDependency();
        });
```

The above bind method registers the NormalDependency class into the service container. Whenever this dependency is resolved from the container, each time a new instance of this class will be provided.

```
$this->app->singleton(SingleTonDependency::class,function($app){
            return new SingleTonDependency();
        });
```

The above singleton method register SingleTonDependency class into the service container. Here Whene ver this dependency is resolved from the container, each time the same instance of this class will be provi ded. A new instance is provided only for the first time resolution of this class, every next resolution will pro vide the same instance of the class.

```
$this->app->bind(ExampleContract::class,function($app){
            return new ViaInterfaceDependency();
        });
```

The above is not different from the earlier discussed bind method, it's just the different technique to resolv e the class using interface. This comes to help while resolving the dependency when we tell to service co ntainer that give me such class that implements a particular interface. Above same example would also w ork with singleton method.

Number of Ways of Resolving the Dependency

Now we have understood how to register the dependencies to the container, now let's understand how to resolve them

Laravel supports automatic dependency injection in the constructor of the class limited to controllers, eve nt listeners, middleware and to the handle method of the queued jobs. For automatic dependency injectio n, the dependencies have to be type hinted

If you have a scenario where you want to resolve dependency other then automatic injection then you hav e to do them manually using the resolve helper method, the resolve method expects dependency name a s a parameter.

Let's see example

```
namespace App\Http\Controllers;

use App\Services\NormalDependency;
use App\Services\SingleTonDependency;
use Illuminate\Http\Request;

class ResolveDependenciesController extends Controller
{
    private $normalDependency;
    private $manualDependency;

    public function __construct(NormalDependency $normalDependency)
    {
        $this->normalDependency = $normalDependency;
    }


    public function resolveDependencyManually()
    {
        $this->manualDependency = resolve(SingleTonDependency::class);
    }
}
```

Above is the demo example of the controller to understand the resolving of dependency from the service container. In the constructor for automatic injection, I have type hinted NormalDependency and in the res olveDependencyManually method I have used resolve helper method for resolving SingleTonDependency class manually from the service container.

One Important note for when to bind dependency manually in service container

When your dependency depends on interface that is give me such class that implements this interface
Or your dependency class need extra configuration to be instanciated like passing additional parameters t o the constructor of the dependency class
If your dependency is simple like we have in this example like NormalDependency such type of dependen cy does not required to be binded manually in service provider because they are resolved automatically b y service container using reflection.