Spring Annotation

@Component
Auto generate a bean ID for decorated class
Components scan will be able to search this Bean ID.
By default, the Bean ID will be non-qualified and de-capitalize class name
Redundant way to write this:

```
@Component("car")
public class Car {...}
```

@Autowired
With Autowired decorated, when we need this object, Spring will search the xml file
for Bean

```
public class Samsung {
@Autowired
// will check the AppConfig
MobileCPU cpu;
```

@Configuration
The decorated class is for annotation configuration. Usually name as AppConfig. We
are expecting this class will contribute to generate the Bean.

```
@Configuration
public class AppConfig {

@Bean
// Declare this is a Bean
public Samsung getPhone(){
return new Samsung();
}

@Bean
// notice the getter function doesn't matter. the bean id is auto generated based on
class name
public MobileCPU getCPU(){
```

```
    return new Intel();
  }
}
```

@Bean

decorated this class is a Bean. When you use AnnotationConfigApplicationContext instead of xml-based, this is equivalent with the Bean definition.

@ComponentScan

Means all components are pre-defined. Spring will scan everything with @Component annotation under package "org.example5"

```
@Configuration
@ComponentScan(basePackages = "org.example5")
public class AppConfig {
// Removing everything, we will get Bean-not-found error
// Adding @Component to Samsung and Intel
}
```

@Primary

When there are multiple Beans represent the same Class for Injection, The Bean with @Primary will be chosen.

```
@Component
@Primary
public class Intel implements MobileCPU {
public void process() { System.out.println("Intel CPU example5"); }
}
->
Samsung Class example 5
Intel CPU example5
```

@Qualifier("")

Specific which Bean you want to choose as source of injection.

```
@Component
public class Samsung {
@Autowired
@Qualifier("amd")
MobileCPU cpu;
 ...
```

```
->
Samsung Class example 5
AMD CPU example5
```