DI using Java Config

Dependency Injection using Java Configuration

When using Dependency Injection using Java Configuration you need to explicitly define all your beans and you need to use @Autowire on @Bean method level to inject dependencies.

```
@Configuration
public class ApplicationConfiguration {
    @Bean
    @Autowired
    public SpringBean1 springBean1(SpringBean2 springBean2, SpringBean3 springBean3) {
        return new SpringBean1(springBean2, springBean3);
    }

    @Bean
    public SpringBean2 springBean2() {
        return new SpringBean2();
    }

    @Bean
    public SpringBean3 springBean3() {
        return new SpringBean3();
    }
}
```

Dependency Injection using Annotations

- Create classes annotated with @Component annotations

```
@Component
public class SpringBean1
```

```
@Component
public class SpringBean2
```

```
@Component
public class SpringBean3
```

- Define dependencies when required

```
@Autowired
private SpringBean2 springBean2;
@Autowired
private SpringBean3 springBean3;
```

- Create Configuration with Component Scanning Enabled

```
@ComponentScan
public class ApplicationConfiguration {
}
```

- Component Scanning

Process in which Spring is scanning Classpath in search for classes annotated with stereotypes annotations (@Component, @Repository, @Service, @Controller, …) and based on those creates beans definitions.

- Simple component scanning within Configuration package and all subpackages

```
@ComponentScan
public class ApplicationConfiguration {
}
```

- Advanced Component Scanning Rules

```
@ComponentScan(
        basePackages = "com.spring.professional.exam.tutorial.module01.question10.annotations.beans",
        //basePackageClasses = SpringBean1.class,
        includeFilters = @ComponentScan.Filter(type = FilterType.REGEX, pattern = ".*Bean.*"),
        excludeFilters = @ComponentScan.Filter(type = FilterType.REGEX, pattern = ".*Bean1.*")
)
public class ApplicationConfigurationAdvanced {
}
```

- Stereotypes – Definition

Stereotypes are annotations applied to classes to describe role which will be performed by this class. Spring discovered classes annotated by stereotypes and creates bean definitions based on those types.

- Types of stereotypes
  - Component – generic component in the system, root stereotype, candidate for autoscanning
  - Service – class will contain business logic
  - Repository – class is a data repository (used for data access objects, persistence)
  - Controller – class is a controller, usually a web controller (used with @RequestMapping)

- Meta-Annotations

Meta-annotations are annotations that can be used to create new annotations.

- Example of Meta-Annotation

@RestController annotation is using @Controller and @ResponseBody to define its behavior

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {
    @AliasFor(
        annotation = Controller.class
    )
    String value() default "";
}
```

- Scopes of Spring Beans

Scope	Description
Singleton	Single Bean per Spring Container - Default
Prototype	New Instance each time Bean is Requested
Request	New Instance per each HTTP Request
Session	New Instance per each HTTP Session
Application	One Instance per each ServletContext
Websocket	One Instance per each WebSocket