

The Spring Framework logo, a stylized green leaf inside a circle, is centered in the background.

Modern Java Component Design with Spring Framework 4.2

Juergen Hoeller
Spring Framework Lead
Pivotal

The State of the Art: Component Classes

```
@Service
@Lazy
public class MyBookAdminService implements BookAdminService {

    @Autowired
    public MyBookAdminService(AccountRepository repo) {
        ...
    }

    @Transactional
    public BookUpdate updateBook(Addendum addendum) {
        ...
    }
}
```

Composable Annotations

```
@Service  
@Scope("session")  
@Primary  
@Transactional(rollbackFor=Exception.class)  
@Retention(RetentionPolicy.RUNTIME)  
public @interface MyService {}
```

```
@MyService  
public class MyBookAdminService {  
    ...  
}
```

Composable Annotations with Overridable Attributes

```
@Scope(value="session")
@Retention(RetentionPolicy.RUNTIME)
public @interface MySessionScoped {

    ScopedProxyMode proxyMode() default ScopedProxyMode.NO;
}
```

```
@Transactional(rollbackFor=Exception.class)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyTransactional {

    boolean readOnly();
}
```

The State of the Art: Configuration Classes

```
@Configuration
@Profile("standalone")
@EnableTransactionManagement
public class MyBookAdminConfig {

    @Bean
    @Scope("session")
    public BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(bookAdminDataSource());
        return service;
    }

    ...
}
```

Configuration Classes with Base Classes

```
@Configuration
public class MyApplicationConfig extends MyBookAdminConfig {

    ...

}
```

```
public class MyBookAdminConfig {

    @Bean
    public BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(bookAdminDataSource());
        return service;
    }

}
```

Configuration Classes with Java 8 Default Methods

```
@Configuration
public class MyApplicationConfig implements MyBookAdminConfig {

    ...

}
```

```
public interface MyBookAdminConfig {
```

```
    @Bean
```

```
    default BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(bookAdminDataSource());
        return service;
    }
}
```

Ordered Configuration Classes

```
@Configuration
```

```
@Order(2)
```

```
public class MyApplicationConfig {
```

```
    @Bean
```

```
    public SpecialBookAdminService myBookAdminService() { ... }
```

```
}
```

```
@Configuration
```

```
@Order(1)
```

```
public class MyBookAdminConfig {
```

```
    @Bean
```

```
    public BookAdminService myBookAdminService() { ... }
```

```
}
```


Generics-based Injection Matching

@Bean

```
public MyRepository<Account> myAccountRepository() { ... }
```

@Bean

```
public MyRepository<Product> myProductRepository() { ... }
```

@Service

```
public class MyBookAdminService implements BookAdminService {
```

@Autowired

```
public MyBookAdminService(MyRepository<Account> repo) {  
    // specific match, even with other MyRepository beans around  
}
```

```
}
```

Ordered Collection Injection

```
@Bean @Order(2)
public MyRepository<Account> myAccountRepositoryX() { ... }
```

```
@Bean @Order(1)
public MyRepository<Account> myAccountRepositoryY() { ... }
```

```
@Service
public class MyBookAdminService implements BookAdminService {

    @Autowired
    public MyBookAdminService(List<MyRepository<Account>> repos) {
        // 'repos' List with two entries: repository Y first, then X
    }
}
```

Lazy Injection Points

```
@Bean @Lazy
public MyRepository<Account> myAccountRepository() {
    return new MyAccountRepositoryImpl();
}
```

```
@Service
public class MyBookAdminService implements BookAdminService {

    @Autowired
    public MyBookAdminService(@Lazy MyRepository<Account> repo) {
        // 'repo' will be a lazy-initializing proxy
    }
}
```

Component Declarations with JSR-250 & JSR-330

```
import javax.annotation.*;
```

```
import javax.inject.*;
```

```
@ManagedBean
```

```
public class MyBookAdminService implements BookAdminService {
```

```
    @Inject
```

```
    public MyBookAdminService(Provider<MyRepository<Account>> repo) {
```

```
        // 'repo' will be a lazy handle, allowing for .get() access
```

```
    }
```

```
    @PreDestroy
```

```
    public void shutdown() {
```

```
        ...
```

```
    }
```

```
}
```

Optional Injection Points on Java 8

```
import java.util.*;  
import javax.annotation.*;  
import javax.inject.*;
```

```
@ManagedBean
```

```
public class MyBookAdminService implements BookAdminService {
```

```
    @Inject
```

```
    public MyBookAdminService(Optional<MyRepository<Account>> repo) {  
        if (repo.isPresent()) { ... }  
    }
```

```
    @PreDestroy
```

```
    public void shutdown() {  
        ...  
    }
```

```
}
```

Declarative Formatting with Java 8 Date-Time

```
import java.time.*;
import javax.validation.constraints.*;
import org.springframework.format.annotation.*;

public class Customer {

    // @DateTimeFormat(iso=ISO.DATE)
    private LocalDate birthDate;

    @DateTimeFormat(pattern="M/d/yy h:mm")
    @NotNull @Past
    private LocalDateTime lastContact;

    ...
}
```

Declarative Formatting with JSR-354 Money & Currency

```
import javax.money.*;
import org.springframework.format.annotation.*;

public class Product {

    private MonetaryAmount basePrice;

    @NumberFormat(pattern="¤¤ #000.000#")
    private MonetaryAmount netPrice;

    private CurrencyUnit originalCurrency;

    ...
}
```

Declarative Scheduling (with two Java 8 twists)

```
@Async
public Future<Integer> sendEmailNotifications() {
    return new AsyncResult<Integer>(...);
}
```

```
@Async
public CompletableFuture<Integer> sendEmailNotifications() {
    return CompletableFuture.completedFuture(...);
}
```

```
@Scheduled(cron="0 0 12 * * ?")
@Scheduled(cron="0 0 18 * * ?")
public void performTempFileCleanup() {
    ...
}
```


Annotated MVC Controllers

```
@RestController
@CrossOrigin
public class MyRestController {

    @RequestMapping(value="/books/{id}", method=GET)
    public Book findBook(@PathVariable long id) {
        return this.bookAdminService.findBook(id);
    }

    @RequestMapping("/books/new")
    public void newBook(@Valid Book book) {
        this.bookAdminService.storeBook(book);
    }
}
```

STOMP on WebSocket

```
@Controller
```

```
public class MyStompController {
```

```
    @SubscribeMapping("/positions")
```

```
    public List<PortfolioPosition> getPortfolios(Principal user) {
```

```
        ...
```

```
    }
```

```
    @MessageMapping("/trade")
```

```
    public void executeTrade(Trade trade, Principal user) {
```

```
        ...
```

```
    }
```

```
}
```

Annotated JMS Endpoints

```
@JmsListener(destination="order")  
public OrderStatus processOrder(Order order) {  
    ...  
}
```

```
@JmsListener(id="orderListener", containerFactory="myJmsFactory",  
    destination="order", selector="type='sell'", concurrency="2-10")  
@SendTo("orderStatus")  
public OrderStatus processOrder(Order order, @Header String type) {  
    ...  
}
```

Annotated Event Listeners

```
@EventListener
public void processEvent(MyApplicationEvent event) {
    ...
}
```

```
@EventListener
public void processEvent(String payload) {
    ...
}
```

```
@EventListener(condition="#payload.startsWith('OK')")
public void processEvent(String payload) {
    ...
}
```

Declarative Cache Interaction

```
@CacheConfig("books")
public class BookRepository {

    @Cacheable
    public Book findById(String id) {
    }

    @CachePut(key="#book.id")
    public void updateBook(Book book) {
    }

    @CacheEvict
    public void delete(String id) {
    }
}
```

JCache (JSR-107) Support

```
import javax.cache.annotation.*;

@CacheDefaults(cacheName="books")
public class BookRepository {

    @CacheResult
    public Book findById(String id) {
    }

    @CachePut
    public void updateBook(String id, @CacheValue Book book) {
    }

    @CacheRemove
    public void delete(String id) {
    }
}
```

Java 8 Lambdas with Spring APIs

```
JdbcTemplate jt = new JdbcTemplate(dataSource);
```

```
jt.query("SELECT name, age FROM person WHERE dep = ?",  
        ps -> ps.setString(1, "Sales"),  
        (rs, rowNum) -> new Person(rs.getString(1), rs.getInt(2)));
```

```
jt.query("SELECT name, age FROM person WHERE dep = ?",  
        ps -> {  
            ps.setString(1, "Sales");  
        },  
        (rs, rowNum) -> {  
            return new Person(rs.getString(1), rs.getInt(2));  
        }  
    );
```

Learn More. Stay Connected.



- **Current: Spring Framework 4.1.7**
(June 2015)
- **Coming up: Spring Framework 4.2**
(July 2015)
- **Coming next: Spring Framework 4.3**
(May 2016)
- **Check out Spring Boot!**
<http://projects.spring.io/spring-boot/>

Twitter: twitter.com/springcentral

YouTube: spring.io/video

LinkedIn: spring.io/linkedin

Google Plus: spring.io/gplus