# SPRING CHEATCODEX

**……Only for Java Devii**

**ROAD RASH!!!**

**………chronologically formulated by  Jyotirmai Tiwari**

**@shadowcodgenstars ~~~~~~~~**

# SPRING CONCEPTS

1. Framework vs Module ?
2. Application framework VS web framework?
3. JavaEE vs spring ?
4. Is Spring a replacement of JavaEE or an extension of JavaEE ? Specify Reason
5. Boilerplate code vs Configurations ?
6. What are different ways to connect the two objects ?

   ***Inheritance:***
   ***Association: mapping{1-1,1-n,n-1,n-n}***
   ***: Aggregation vs composition:::***

   ***Interfaces:***
   ***IOC container Spring :***
7. Why is Inheritance a less preferred approach than association for object communication?
8. Inversion of control ?
9. Explain Bean Scope ? Types of Bean scope ?
10. Explain certain design patterns?

    ***Creational: Abstract Factory, Builder, Factory Method, Prototype, Singleton***

    ***Structural: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy***

    ***Behavioral: Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor***
11. Important Terms =>

    1. **Dispatcher Servlet** ?? handle http request and map it to the specific controller method.

    2. **Application Context**?? There can be many containers inside the spring application; these all containers can be segregated into two domains BeanFactory and ApplicationContext .

    3.**IOC Container** ?? controls up the formation and life cycle of object , by accepting

    configurations(metadata) and POJO Classes as input and create a Bean .

    A design pattern that enforces loose coupling upon objects.

    4. **POJO Classes** ?? a class that has no restriction imposed and only follows the JVM restrictions . No serialization needed and no configs required.

    5. **Bean** ?? a java POJO class with configs and database object mapping defined.

    6. **Dependency Injection** ?? the process of interrelating the objects to each other ,

u need to inject the dependency after initialisation of object | bean formation by IOC Container.

7. **Dependency** ?? dependency can be another object, class, service etc.

8. **Injector** ?? here it is IOC Container , that injects the dependency to its client.

12. Loose coupling vs Tight Coupling ?
13. Application context implementations ?

    ***ClassPathXmlApplicationContext***

    ***FileSystemXmlApplicationContext***

    ***AnnotationConfigApplicationContext***

    ***GenericWebApplicationContext***

14. Which out of the two techniques to use for Bean initialisation . BeanFactory or ApplicationContext??

    Beanfactory:older,memory-effective,does lazy-initialization of beans,security_specific application,small_app development
    Applicationcontext: extra features,more popular technique,large_application
15. How to perform dependency injection in Spring ? Techniques?

1. **Setter injection** :

```
  <bean id="" class="">
  <property name="" value="" />
  <property name="" ref="" />
  </bean>
```

2. **Constructor injection** :

```
<bean id="" class="">
  <constructor-arg value="" />
  <constructor-arg ref="" />
</bean>
```

3. **Java based** :use stereo-type annotation(@controller,@service,@componet etc) and @componentscan
 Method | factory injection: factory-method, factory-bean

```
<bean id=" " class=" "></bean>
```

```
<bean id=" " class=" " factory-method=" " factory-bean=" "></bean>
```

4. **Field injection** :
use @Autowired

"internally uses setter injection,constructor injection"
**FIELD INJECTION is not recommended way to inject dependency ?why**
Note::: @Autowired can only inject ***Object Type dependency***

16. Differentiate between setter and constructor injection?

17. In constructor injection in case of multiple constructor how to avoid ambiguity ?
**::: a constructor with maximum no of parameters is invoked first by default**
**:::use type,name,index inside constructor injection to avoid conflicts.**

18. Differentiate between setter injection and constructor injection?
**Constructor Injection =>**
   1. **Mandatory Dependency injection**
   2. **Safety and readability**
   3. **Object or bean created requires immutability.**

**Setter injection=>**
   1. **Optional Dependency Injection.**
   2. **You can make changes in bean created**
   3. **Memory effective**
   4. **Circular dependency injection**
   5. **Setter injection always overrides the constructor injection**
19. Why is setter injection memory effective?
20. Minimal setup required for spring application?
**Two classes, xml config,spring jars(core+bean:::IOC+DI)**
21. Is spring open source ? what's its need.
22. Spring Features ?lightwt,loosely coupled,modules,aop,template driven
23. Why is Spring Lightweight?
No Application Server required| like JAVAEE Server
Modular installation

24. Spring works upon which Design Principle?
25. Difference between spring and struts?

26. Modules in Spring ?
27. What is dependency?
28. What are mock objects?
29. What are factory methods?
**::: return the instance of class**
30. Define Bean scope? What is default bean scope ?
***:singleton ,prototype,request,session,application,websocket***
***Singleton: for the whole application single object created***
***Prototype: at every bean request .getBean() new object is created***
***Request: at every HTTP Request .getBean() creates new object***

***Session: at every HTTP session .getBean() creates new object***
***Application: for each servletcontext a new object is created***
***Websocket : for each websocket a new object is created***

***!!!End { Spring_Basics }***

## Let's explore the SHOw!!!

# Spring AOP

1. What is the difference between OOPS Design Pattern and AOP ?
2. What is the need of AOP?

   **Multiple code repeatition,increase in modularity, memory effective**

3. What are cross-cuts in AOP?

   **Cross-cuts are the centralized part of a system ,required for all components ,like : logging info, security check ,cache used, transactions,db authorisation is needed under all the components .**

4. join points?

   **A part of code where injection of advice is needed**

5. Advice ?

   Action taken at particular join-point ,

   @beforeadvice, @afteradvice, @throwadvice, @afterreturn , @around

6. Aspect?

   Have a connection with both join-points and advice ,its combo!!!

7. Weaving: The process of linking aspects to the execution of an application.
   a. Compile-Time Weaving: Modifications are applied at compile time.
   b. Load-Time Weaving: The aspect is applied when the class is loaded, often by a special class loader.
   c. Run-Time Weaving: Changes are made during the execution of the code.
8. The three weaving mechanisms can be further categorized into four strategies:

    a. Singleton weaving: The aspect is a singleton and is woven into the client at most once.
    b. Per-instance weaving: The aspect is woven into each object before it is returned.
    c. Single-time weaving: The aspect is woven into the client the first time it is instantiated.
    d. Combination-of-above weaving: A combination of the above strategies is used to achieve weaving.
9. Decoration: Uses a proxy or wrapper to intercept method calls and apply cross-cutting concerns.
    a. Dynamic Proxy: Java's `java.lang.reflect.Proxy` is often used.
    b. CGLIB: A code generation library for high-performing and customized proxies.
10. Code visuals:::

    **public class** BeforeAdvisor **implements** MethodBeforeAdvice{

    **public class** AfterAdvisor **implements** AfterReturningAdvice{

    **public class** AroundAdvisor **implements** MethodInterceptor{

    **public class** ThrowsAdvisor **implements** ThrowsAdvice{

11. Example of AOP concept in spring ?

```
public class OnlineStore {
    private String productName;

    public String getProductName() {
        return productName;
    }
    public void setProductName(String productName) {
        this.productName = productName;
    }
}
```

```
@Aspect
public class LogAspect {

    @Before("execution(* OnlineStore.getProductName())")
    public void logMethodName(JoinPoint joinPoint) {
        System.out.println("Method invoked: " +
joinPoint.getSignature());
    }

    @AfterReturning(pointcut = "execution(*
OnlineStore.getProductName())", returning = "result")
    public void logReturnValue(JoinPoint joinPoint, Object result) {
        System.out.println("Returned: " + result);
    }

}
```

```
@Configuration
@EnableAspectJAutoProxy
public class AppConfig {

    @Bean
    public OnlineStore myProduct() {
        return new MyProduct();
    }

    @Bean
    public LogAspect logAspect() {
        return new LogAspect();
    }

}
```

///////……………..AOP BYE…BIle……………………

# SPRING MVC

1. Why use spring mvc ?

  1. **Web framework**
  2. **Model-view-controller**
  3. **Spring mvc use inbuilt dispatcher servlet**
  4. **It uses light-weight servlet container to develop and deploy your application.**

2. Can by using spring we cannot make web applications?

**No you can do so by including dispatcherservlet JARS dependencies explicitly in xml file**

**3.** Important Terms :=>

1. Frontcontroller :
2. Dispatcherservlet:
3. contextLoaderListener: **starts up and shuts down Spring's root *WebApplicationContext,extract the config from web.xml and before hand load up all beans***

   ***Combines and automates the bean creation process + application context initialisation to the servletcontext initialisation and its lifecycle, so automatically at servlet loading applicationcontext related work is handled.***

4. ***Applicationcontext vs servletcontext:***
5. Controller 👍
6. Services 👍
7. Model:
8. Model interface 👍
9. ModelAndView interface 🙂
10. View:
11. ViewResolver 👍
12. HttpSession
13. HttpRequest
14. HttpResponse
15. getAttribute()| setAttribute()

4. How is the Frontcontroller able to map with a specific controller ?

<servlet-mapping></> + scan-component(“package_name”) + @controller

5. Minimal code for constructing mvc projects ?

**Load the spring JARS files and add dependencies ,create controller,web.xml(servlet-mapping), xml file,server jars**

**6. Annotations :::-->**

1. **@Controller**
2. **@ControllerAdvice**
3. **@RestController : @Controller + @ResponseBody**
4. **@RequestMapping(“/”):** map the url to the method

   ***@RequestMapping(“/home”)***

5. **HttpServletRequest :** get the request parameters
6. **@RequestParam :** is used to read info from form, replacement of HttpServletRequest, request_parameters then can be further used to make query to db

   **@RequestParam(“username”) String uname;**

7. **@ModelAttribute(“”) :** the database object is converted to a web-view object.

   the method return value is converted to web-based object and merged as model attribute and send to view

   **@ModelAttribute("reservation") Reservation res**

8. **@ResponseBody** :it returns the rest response to client |web browser.

9. **@RequestBody** :web browser sends the object as a part of http request , that is decomposed to XML | JSON form and saved as java object.
10. **@GetMapping,@PostMapping,@PutMapping,@PatchMapping,@DeleteMapping**
11. **@PathVariable("")** : extract the URI path variables info and save it internally.

    **@RequestMapping("/main/{uid}/{utype}")**

    **Public String process(@PathVariable("uid") int user_id, @PathVariable("utype") String user_type){}**

12. **@RequestParam CommonsMultipartFile file ??**
13. **HttpSession**
14. **@SessionAttributes :**storing the model attribute in the user's session**.**

15. **@Autowired: field or method injection, dependency to be injected must be object type**
16. **@EnableWebMvc:** <mvc: annotation-driven> in an XML configuration

**7. What is Model,ModelAndView,ModelMap**

**8. What is a form backing object?**

**9. @Qualifier vs @Autowired ?**

**10. When to use @Qualifier with @Autowired ?**

**11. @Required Annotation ? Why is it used in Setter injection ?**

**12. When and why to use MultiPartResolver ?**

**13. What are spring interceptors?**

**14. How can we handle exceptions in spring? @ExceptionHandler()?**

**15. ResponseEntity<Object>  ?**

16. **@ResponseStatus ,@ControllerAdvice,@ExceptionHandler ?**

# SPRING BOOT

1. **Why use spring boot ?**

**Rapid application development , boilerplate code, default configs|minimal config ,no xml configs ,use starters**

**Spring + starters + embedded server(tomcat)- xml config=spring boot**

2. **SpringApplication.run(ClassName.class, args) what it does ??**

   **Bundle up your module as a single class and convert to a single jar and then use starter parent dependency to attach it with its default dispatcher servlet .**

   `@EnableAutoConfiguration , @SpringBootApplication`

3. **Autoconfiguration ? How does it happen in detail?**

   **Configuration or metadata for bean creation is automatically loaded as per jars and maven dependencies.**

4. **Bootstrap the application?**
5. **Latest version of spring , and which versions of java it supports ?**
6. **New features been integrated to springboot ?**

   **@ConfigurationProperties: for setter di injection.**

   **Spring.main.lazy-initialization**

   **Spring.application.admin.enabled: to enable admin related default features as provided under new spring.**

Rsocket di : for stream based communication between client-server or in microservices communication.

7. How is the spring boot able to run web applications without servlet mapping?

   A starter dependency starter-web need to be injected inside the pom.xml

   For a proof in application.properties

   ```
   server.servlet.context-path=/project_javalearner
   ```

   ```
   spring.mvc.servlet.path=/servlet_frontcontroller
   ```

   ```
   U can checkout at the localhost/servlet_frontcontroller for
   demonstration.
   ```

8. What are build tools ? Why are they used ?

   A build tool, or software utility, is essential in contemporary software development since it automates the transformation of source code into an executable and deployable program.It concatenates your pojos , configs, db interface and inject dependency , providing you with dependency manager.

   Converts the whole project folder→.class->.JAR,.war,.ear.apk -> EXECUTE N RUN on JVM CONTAINER

   Maven and Gradle are used in boot .

   GRADLE: use when no xml,java annotations or Groovy

   MAVEN: for xml based config, fast build formation

9. Can I use other build tools than maven and gradle ?

   Yes , you can use Ant. , n others in market

10. What does spring-parent contain?

    A dependency management system ,boilerplate code +default config ,dependency tree structure ,default maven-configs.

11. Where do we define the application configs if needed?

    Use application.properties or application.ymls file

12. Spring starters a dependency descriptor ? How?
13. Illuminate some points about the maven repository ?
14. Dependency tree: how to trace in ? cmd ?
15. Starter and the providers ?

    Spring-starter-parent: dependency manager, default maven config ,autoconfiguration,version manager

    Spring-starter-web: tomcat,mvc features,web-flux

    Spring-data-jpa: jpa specification, mapping the java object to JDBC,ORM objects,boilerplate code to (load driver, connection start,query_sys , transaction starter) +JPQL(a java_based styloo to query the db)

16. Spring Data Repository ?

    **CrudRepository**

    **PagingAndSortingRepository**

    **JPARepository**

17. Reactive web application development?web-flux ?
18. DB related Annotations

    @RestController : return JSON RESPONSE or XML RESPONSE,no @ResponseBody needed in additional

    @RestController=@Controller+@ResponseBody

**@controller : return HTTP RESPONSE,listens to http request and return view name that has been resolved by viewresolver and then returned as user_view.**

**@Entity :**

**@Columns:**

**@Table() :**

**@Id : @GeneratedValue(strategy = GenerationType.AUTO) : generated type AUTO,IDENTITY,TABLE,SEQUENCE**

19. **Important Terms :::=>>>**

    **JPQL :**

    **EntityManagerFactory:**

    **EntityManager: EntityManager API for processing queries and transactions on the objects against the database. It uses a platform-independent object-oriented query language JPQL (Java Persistence Query Language).**

    **Entity :**

    **Persistence object :**

    **Transient object : not to map in as db property**

20. Define the flow of how object *JAVA is saved as object*DBO ??

object_java->java persistence API(persistence)—> entitymanagerfactory->entitymanager{manage entity,transaction,JPQL CONVERSION TO db query}->entity->DB_STORAGE

21. ORM VS JPA VS JDBC/ODBC VS Hibernate ?
22. Differentiate between Hibernate and JPA ?
23. How to connect MYSQL TO SPRING BOOT ?
24. HOW to enable in-memory database, and configure it to Spring ?
25. In-memory database ,why to use ? some use cases when we should use them ?

    Fast access-time, storing the data which is most frequently demanded by user and the rate of change of that object is minimal.eg: use for testing , intermediate service data saving ,tracking information

26. When we have an in-memory data_store system ,what's the need of cache? Caching system?

    redis||Cache :

    1. data stored on RAM
    2. most fast
    3. no configs needed(url,password,driver)
    4. It is in mid of application and db.(temporary storage region)
    5. Small in storage size

    H2 DB:

    1. data storage is by default in-memory(main memory|RAM) but can be configured to store upon disk
    2. (needs configs, query system)
    3. Its a in memory-DB system

4. Storage is large as per cache

28. Explain the following Annotations @EnableCaching ,@CacheConfig

29. SPRING ANNOTATIONS : a metadata , easiest way to inject metadata

- Following are the additional spring boot annotation
- @EnableAutoConfiguration: It enables the Spring Boot auto-configuration mechanism.
- @ComponentScan: It scans the package where the application is located.
- @Configuration: It allows us to register extra beans in the context or import additional configuration classes.

@SpringBootApplication: it's the combination of above three mentioned annotations.

@Required : used in setter injection, for mandatory parameter initialisation for bean creation.

@Autowired: used to inject one class as dependency inside another class, perform tight coupling of objects.

**@Component : it used to define that this class is a component ,so it needs to be auto initialized by the ioc container at boot time . This reveals the following class methods will be recognised as Java Beans .**

**@configuration : it specifies that this class defined @beans will be used as an alternate of xml-configs.**

**@ComponentScan : it need to present in main class that initiates the runner, this annotation searches for all @component present in the project|package passed as a parameter value .**

**@Controller :**

**@RequestMapping :**

**@Service :**

**@Repository :**

**@Trasient :**

**@Value:**

**@MappedCollection :**

```
@MappedCollection(idColumn = "CUSTOMER_NAME")

    Set<MyCustomerEntity> customerEntities;
```

**@Query :custom query**

**@OneToMany:**

```
  @OneToMany(mappedBy="product")
```

```
    private Set<Item> items;
```

**@ManyToOne:**

```
@ManyToOne

    @JoinColumn(name="product_id", nullable=false)

    private Product product;
```

@Transactional() :

@Lock() : + types of lock supported

30. RowMapper vs ResultSetExtractor ?

31. ApplicationListners and Event Handling in spring ?

32. Lifecycle of Event Management | handling in spring ?

33. Service registry ?

34.sessionFactory ?

—--------------->>>>>UPDATIONS still UNDER MENTAL HEALING

<<<<<<<<<<<<<<<<PREP-THOUGHT:

**“CRITICIZE YOUR IDEA BEFORE U DRAW CONCLUSION “**

**!!!!......@@@@@@.....#JYotIrMai tIwArI**

**…..published under SHADOW _CODGEN_ STARS :){;}**