	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql

51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9cIHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

MCQ on Spring and Hibernate

Q1. Robert is working on a web application, where Spring is being used. He wants to apply prototype scope in the bean named as 'Person'. Which one of the following is the correct option where the 'Person' bean works within the prototype scope :

```
(A) <bean id="person" class="com.test.Person"  scope="Prototype"/>
```

```
(B) <bean id="person" class="com.test.Person"  scope="prototype"/>
```

```
(C) <bean id="person" class="com.test.Person"  PrototypeScope="true"/>
```

```
(D) <bean id="person" class="com.test.Person"  Scope="prototype"/>
```

A1. B is correct. Option 'A' and 'D' has syntactical errors. Option 'C' has PrototypeScope which is not a valid attribute. Hence B is correct.

Q2. Spring framework offers us multiple ways to define the scope of a bean. Which one of the following is an incorrect way of defining a 'request' scope?

```
(A) @RequestScope
    public class Product {
           // some properties & methods
    }
```

```
(B) <bean id="product" class="com.test.Product" Scope="request"/>
```

```
(C) @Scope("request")
    public class Product {
           // some properties & methods
    }
```

```
(D) <bean id="product" class="com.test.Product" scope="request"/>
```

A2. B is correct. Option B has incorrect attribute 'Scope'. It should be 'scope'. All Other options are correct.

Q3. Select the option that is not a correct approach to create bean life cycle methods in Spring framework?

```
    (A) By using Annotations

    (B) By implementing Interfaces provided by Spring framework

    (C) By extending classes provided by Spring framework

    (D) By using XML configuration
```

A3. C is correct. Spring doesn't provide any class to extend in order to create life cycle methods, but it provides interfaces.

Q4. Which interface out of the following options will you use to perform destruction of beans in the context of the spring life cycle methods?

```
    (A) InitializingBean

    (B) PostConstruct

    (C) DisposableBean

    (D) PreDestroy
```

A4. C is correct. Spring allows your bean to perform destroy callback method by implementing the DisposableBean interface. Option B & D are the annotations, not the interface.

Q5. Who is capable of maintaining a registry of different beans and their dependencies in a Spring based web application?

```
(A) BeanFactory class
```

```
(B) BeanFactory interface
```

```
(C) beanFactory method
```

```
(D) Client Application
```

A5. B is correct. A BeanFactory is the interface which is capable of maintaining a registry of different beans and their dependencies.

Q6. Select the incorrect statement about BeanFactory in Spring Framework?

```
(A) BeanFactory holds bean definitions

(B) BeanFactory instantiates beans whenever asked by the client application

(C) BeanFactory is capable of creating associations between dependent objects whil

(D) BeanFactory supports the Annotation-based dependency Injection
```

A6. D is correct. BeanFactory does not support the Annotation-based dependency injection, whereas ApplicationContext, a superset of BeanFactory does.

Q7. Four annotations given below, are used in Spring Boot based application. Which one is the annotation of Spring Boot that is an alternative to Spring's standard @Configuration annotation?

```
(A) @EnableAutoConfiguration
```

```
(B) @SpringBootConfiguration
```

```
(C) @ConfigurationProperties


(D) @ConfigurationPropertiesScan
```

A7. B is correct. @SpringBootConfiguration is an alternative to Spring's standard @Configuration annotation.

Q8. What is the purpose of @SpringBootConfiguration annotation in a Spring Boot web application?

```
(A) enables registration of extra beans in the context


(B) scans on the package where the application is located


(C) enables Spring Boot's auto-configuration mechanism


(D) disables additional configuration classes from the application
```

A8. A is correct. @SpringBootConfiguration enables registration of extra beans in the context or the import of additional configuration classes.

Q9. Johnson is a developer. He is working in a Hibernate based application. He defines a class, called 'InvoiceId'. The fields 'category' and 'name' will represent a unique InvoiceId as below:

```
@Embeddable
public class InvoiceId implements Serializable {


    private String category;


    private String name;
    // getters and setters // equals() and hashCose()
}
```

Which of the following option represents that the 'Invoice' entity has a composite key?

```
(A) @Entity
    public class Invoice {
       @Id
       @Embedded
       private InvoiceId id;
       private String description;
       private Double amount;
       //standard getters and setters
    }
```

```
(B) @Entity
    public class Invoice {
       @EmbeddedId
       private InvoiceId id;
       private String description;
       private Double amount;
       //standard getters and setters
    }
```

```
(C) @Entity
    public class Invoice {
       @Id
       private InvoiceId id;
       private String description;
       private Double amount;
       //standard getters and setters
    }
```

```
(D) @Entity
    public class Invoice {
       @EmbeddableId
       private InvoiceId id;
       private String description;
       private Double amount;
```

```
            //standard getters and setters
        }
```

A9. B is correct. @EmbeddedId is the correct annotation to represent that an entity has a composite key.

Q10. Suppose you are working on a Hibernate-based application. You have two classes as below. You want to create a table into the database using hibernate ORM concept.

```
    public class Employee {
        private int id;
        private Name name;
        private double salary;
        // getters & setters
    }

    public class Name {
        private String firstName;
        private String lastName;
        //getters & setters
    }
```

You want to create a table named 'employee' with 4 columns: Id, firstName, lastName, salary. Which option is correct to create the aforesaid table. Assume that table & column names are not case-sensitive.

```
    (A) public class Employee {
            @Id
            private int id;
            private Name name;
            private double salary;
            // getters & setters
        }

        @Embeddable
        public class Name {
```

```
       private String firstName;
       private String lastName;
       // getters & setters
    }
```

(B)
```
    @Entity
    public class Employee {
       @Id
       private int id;
       @Embedded
       private Name name;
       private double salary;
       // getters & setters
    }

    @Embedded
    public class Name {
       private String firstName;
       private String lastName;
       // getters & setters
    }
```

(C)
```
    @Entity
    public class Employee {
       @Id
       private int id;
       @Embedded
       private Name name;
       private double salary;
       // getters & setters
    }

    @Entity
    public class Name {
       private String firstName;
       private String lastName;
       // getters & setters
    }
```

(D)
```
    @Entity
    public class Employee {
       @Id
       private int id;
       @Embeddable
       private Name name;
       private double salary;
       // getters & setters
   }

    @Entity
    public class Name {
       private String firstName;
       private String lastName;
       // getters & setters
    }
```

A10. C is correct. In Option A , @Entity is missing in Employee class. In Option B, @Embedded is disallowed at class level and In Option D, @Embeddable is disallowed at field level.

Q11. Mary is working in a Hibernate based application as a web-application developer. She has two entities in her application: Student & Address. She is implementing a relation where One student can have multiple addresses and one address can accommodate multiple students in a bidirectional relationship. Mary doesn't want hibernate to create one additional table. Which of the following annotation and its attribute will she use to fulfill the given requirement?

```
(A) @ManyToMany and mappedBy

(B) @ManyToOne and mappedBy

(C) @OneToMany and MappedBy

(D) @OneToOne and MappedBy
```

A11. C is correct. As per the problem statement, only @OneToMany or @MantToOne can be used. Hence, Options A & D are incorrect. MappedBy attribute is disallowed in @ManyToOne, that makes Option B also incorrect. Hence, option C is correct. If you wish to learn entity relationship concept in detail, kindly visit a separate article on Entity Relationship In JPA/Hibernate/ORM.

Q12. In the context of Access types in Hibernate, if @Id is located at the getter method of an entity, what does it mean?

```
(A) Entity access behavior is Field Access

(B) Entity access behavior is Property Access

(C) It represents both field & property access behavior

(D) From the given information, we can't determine the Access behavior
```

A12. B is correct. If we apply @Id on the getter method of an entity/class, it means property access behavior is enabled. If we apply @Id on field, it means field access behavior is enabled.

Q13. Suppose you are working on a web application that uses Hibernate as an ORM tool. At which level/(s) @Access is allowed to be used in an Entity?

```
(A) Field Level

(B) Method Level, Field Level

(C) Method Level

(D) Field Level, Method Level, Class level
```

A13. D is correct. @Access can be used at all three levels : Field, Method & Class.

Q14. In JPA, which one of the following annotation converts the date and time values from Java object to compatible database type and retrieves back to the application.

```
(A) @Timestamp

(B) @Time

(C) @Temporal
```

`(D) @Date`

A14. C is correct. @Temporal annotation converts the date and time values from Java object to compatible database type and retrieves back to the application.

Q15. Jacob is working on a web application where JPA is being used in data layer. He wrote a POJO class and wants this class to work as an entity to implement the ORM concept. What are the minimum required annotations he must use at the class to create a table in the database?

`(A) @Id, @Entity`

`(B) @Table, @Entity, @Id`

`(C) @Column, @Entity, @Table`

`(D) @Id, @GeneratedValue`

A15. A is correct. In order to create a table in the database, @Entity & @Id annotations are mandatory, otherwise JPA will throw an exception.

Q16. Consider the following bean definition :

```
<bean id="test" class="com.dev.test.TestBean" />
```

Which of the below bean scope is applied in this bean?

`(A) Session`

`(B) Request`

```
(C) Prototype
```

```
(D) Singleton
```

A16. D is correct. A bean has singleton scope by default, whether we declare it in the bean definition or not.

Q17. Robin is working in a Spring based web application. He declares the scope of a bean by using annotations. Select the option which is incorrect in declaring the scope.

```
(A) @Component
    @Scope("request")
    public class Product {
     //some methods and properties
    }
```

```
(B) @Component
    @Scope("session")
    public class Product {
     //some methods and properties
    }
```

```
(C) @Component
    @Sessionscope
    public class Product {
     //some methods and properties
    }
```

```
(D) @Component
    @SessionScope
    public class Product {
     //some methods and properties
    }
```

A17. C is correct. Option C has incorrect annotation. It should be '@SessionScope' in place of '@Sessionscope'. All other options are correct.

Q18. Suppose you are working on a Spring based Web Application. Generally, there are three ways to implement the core features of Spring framework in your application:

1) Using Annotation 2) Using XML configuration 3) Implementing Interfaces provided by Spring framework

Which option is correct if you want to define Bean Lifecycle methods in your application:

```
(A) Only Option (1)

(B) Option (1) & Option (2)

(C) Option (2) & Option (3)

(D) All three
```

A18. D is correct. There are three ways to define bean life cycle methods: Annotation or XML configuration or Spring Interfaces.

Q19. Which option is incorrect about @Component annotation?

```
(A) It scans our application for classes annotated with @Component

(B) It instantiates classes whenever required

(C) It supports Spring's auto-detection mechanism

(D) It doesn't inject any specified dependencies
```

A19. D is correct. @Component annotation also injects any specified dependencies.

Q20. Melissa works in an application where Spring is being used. She wrote a custom function in a class that handles bean life cycle disposal. By mistake, She also implemented the DisposableBean interface and overrides the default method provided by the Spring container in the same class. What will happen if she runs the application?

```
(A) Only default method will be called

(B) Only custom method will be called

(C) First default method and then custom method will be called

(D) First custom method and then default method will be called
```

A20. C is correct. Default methods provided by Spring always take precedence in a bean life cycle.

Q21. There are two classes 'Product' and 'ProductCategory' in a Spring based Project. You want to inject ProductCategory into Product class by means of constructor injection. Which of the below option is correct to implement it?

```
(A) @Component
    public class Product {
        private int id;
        @Autowired
        private ProductCategory category;
        public Product(ProductCategory category) {
           this.category = category;
        }
        public ProductCategory getCategory() {
           return category;
        }
        public void setCategory(ProductCategory category) {
           this.category = category;
        }
    }

(B) @Component
    public class Product {
       private int id;
```

```
        private ProductCategory category;
        public Product(ProductCategory category) {
            this.category = category;
        }
        public ProductCategory getCategory() {
            return category;
        }
        @Autowired
        public void setCategory(ProductCategory category) {
            this.category = category;
        }
    }
```

(C)
```
    @Component
    public class Product {
        private int id;
        private ProductCategory category;
        @Autowired
        public Product(ProductCategory category) {
            this.category = category;
        }
        public ProductCategory getCategory() {
            return category;
        }
        public void setCategory(ProductCategory category) {
            this.category = category;
        }
    }
```

(D)
```
    @Component
    public class Product {
        private int id;
        private ProductCategory category;
        public Product(ProductCategory category) {
            this.category = category;
        }
        @Autowired
        public ProductCategory getCategory() {
            return category;
        }
        public void setCategory(ProductCategory category) {
            this.category = category;
            }
        }
```

A21. C is correct. @Autowired annotation will be applied on constructor as the question is talking about constructor injection.

Q22. Suppose you are working on a Student library system which is developed in the Spring framework. There are four classes as shown below.

```
@Component
public class Student {
   private int id;
   private Address address;
}
@Component
public interface Address {
  // some fields & Methods
}
@Component
public class PermanentAddress implements Address {
  // some fields & Methods
}
@Component
public class MailingAddress implements Address {
  // some fields & Methods
}
```

You want to inject dependency of PermanentAddress class into Student class. Which option represents the correct use of dependency injection?

```
(A) @Component
    public class Student {
        private int id;
        @Autowired
        private Address address;
     }
```

```
(B) @Component
    public class Student {
       private int id;
       @Autowired
       @Qualifier("PermanentAddress")
       private Address address;
    }
```

```
(C) @Component
    public class Student {
       private int id;
       @Autowired
       @Qualifier
       private Address address;
    }
```

```
(D) @Component
    public class Student {
       private int id;
       @Autowired
       @Qualifier("permanentAddress")
       private Address address;
    }
```

A22. D is correct. Spring container by default considers bean name same as the class name, but with the first letter in lower case. In order to get more idea on @Qualifier, visit the article on @Qualifier Annotation.

Q23. Which option is true for the role of BeanFactory in Spring Framework:

```
(A) BeanFactory provides the configuration framework and basic functionality
```

```
(B) BeanFactory provides access to messages in i18n-style
```

```
(C) BeanFactory provides access to resources, such as URLs and files
```

```
(D) BeanFactory provides event propagation to beans implementing the ApplicationLi
```

A23. A is correct. Options B, C, D are the roles of ApplicationContext, not the BeanFactory.

Q24. Select the option which is true about BeanFactory & ApplicationContext in the Spring Framework:

```
(A) Any description of ApplicationContext capabilities and behavior should be cons

(B) An ApplicationContext is a complete superset of a BeanFactory

(C) BeanFactory builds on top of the ApplicationContext.

(D) When building most applications in a J2EE-environment, the best option is to u
```

A24. B is correct. According to Spring Framework documentation, option B is correct. For more details, kindly refer the Spring Official Documentation.

Q25. @SpringBootApplication annotation is a combination of three annotations. Which one of the given option is not the part of a combination of three annotations?

```
(A) @SpringBootConfiguration

(B) @EnableAutoConfiguration

(C) @Configuration

(D) @ComponentScan
```

A25. A is correct. @SpringBootConfiguration is not the part of this combination, but the @Configuration is. For more details, visit @SpringBootApplication annotation.

Q26. Consider the below code where the 'Product' entity has 'Category' as @EmbeddedId and other fields related to a product. A 'Category' tells Hibernate that the 'Product' entity has a composite key.

```
@Entity
public class Product implements Serializable {
    @EmbeddedId
    private Category id;
    private String name;
    //getters & setters
}
```

What will be the structure of the Person entity in the context of the above composite key?

```
(A) @Embedded
    public class Category implements Serializable {
       private String name;
       private String description;
       //getters & setters
    }
```

```
(B) @Component
    public class Category implements Serializable {
       private String name;
       private String description;
       //getters & setters
    }
```

```
(C) @Entity
    public class Category implements Serializable {
       private String name;
       private String description;
       //getters & setters
    }
```

```
(D) @Embeddable
```

```
public class Category implements Serializable {
   private String name;
   private String description;
   //getters & setters
}
```

A26. D is correct. We represent a composite primary key in Hibernate by using the @Embeddable annotation on a class.

Q27. In a Hibernate based application, in order to interact with the database, we make use of various methods provided by EntityManager API. Which statement is false in the context of these methods?

```
(A) We can make use of the persist() method in order to have an object associated w

(B) We can use the findReference() method, if we just need the reference to the ent

(C) We can use the detach() method to detach an entity from the persistence context

(D) We can use the find() method to retrieve an object from the database.
```

A27. B is correct. If we just need the reference to the entity, we can use the getReference() method, not the findReference().

Q28. In order to access entity attributes, we implement access strategies in Hibernate. Which type/(s) of Access strategy/(ies) is allowed in Hibernate?

```
(A) field-based access

(B) property-based access

(C) class-based access
```

```
(D) field-based access & property-based access
```

A28. D is correct. There are two types of access strategies allowed in Hibernate : field-based access and property-based access.

Q29. Sophia is using Spring Data JPA for the data layer implementations in her web application. She wrote a custom method in the repository. She is using 'update' query to implement the functionality of the method. Which one of the following annotation she should use on that method?

```
(A) @Updating

(B) @Updated

(C) @Modifying

(D) @Modified
```

A29. C is correct. Only @Modifying annotation is available. All others are incorrect.

Q30. Which option is incorrect about Spring Framework?

```
(A) It is a lightweight framework.

(B) Spring applications are loosely coupled because of dependency injection.

(C) It offers only Java-based annotations for configuration options.

(D) It provides declarative support for caching, validation, transaction, and form
```

A30. C is correct. Apart from Java-based annotations, it also offers XML based configuration options. Hence option 'C' is the right answer.

Q31. What is the main component of the Spring framework?

```
(A) Servlet API
```

```
(B) Hibernate ORM
```

```
(C) Spring Core Container
```

```
(D) Spring Data JPA
```

A31. C is correct. The Spring Core Container is the main component of the Spring framework, which is responsible for managing the application's beans and their dependencies.

Q32. What is the purpose of the BeanFactory interface in Spring?

```
(A) To manage the lifecycle of beans
```

```
(B) To provide a simple way to retrieve bean instances
```

```
(C) To define beans and their dependencies
```

```
(D) To handle transaction management
```

A32. B is correct. The BeanFactory in Spring is an interface that provides a simple way to retrieve bean instances.

Q33. What is the main difference between ApplicationContext and BeanFactory in Spring?

```
(A) BeanFactory is a sub-interface of ApplicationContext
```

```
(B) ApplicationContext provides additional functionalities than BeanFactory

(C) BeanFactory and ApplicationContext both are same

(D) ApplicationContext and BeanFactory are interchangeable
```

A33. B is correct. The ApplicationContext interface in Spring is a sub-interface of BeanFactory and provides additional functionalities like event publishing, internationalization, and application-level contexts.

Q34. What is the purpose of the @Autowired annotation in Spring?

```
(A) To define a bean in the Spring application context

(B) To inject dependencies into a bean

(C) To define the scope of a bean

(D) To define the lifecycle of a bean
```

A34. B is correct. The @Autowired annotation in Spring is used to inject dependencies into a bean, either through constructor injection, setter injection, or field injection.

Q35. What is Inversion of Control in Spring?

```
(A) A design pattern that allows for greater control over an application's objects

(B) A technique for delegating control of the application to a central authority

(C) A method for controlling an application's flow of control from the inside out
```

```
(D) The term 'Inversion of Control' is not used in Spring
```

A35. B is correct. Inversion of Control (IoC) is a technique for delegating control of the application to a central authority, such as the Spring IoC container. The container creates and manages the objects and their dependencies, allowing the objects to focus on their specific responsibilities.

Q36. Which of the following caching strategies in Hibernate stores data in a cache that can be accessible across different sessions?

```
(A) Query cache

(B) First-level cache

(C) Second-level cache

(D) Collection cache
```

A36. C is correct. The first-level cache is related to a single session and is used to cache data within that session only. The second-level cache in Hibernate caches data across different sessions. It stores the cached data in a shared cache region. It allows multiple sessions to access the same data without hitting the database. The query cache caches the results of queries, and the collection cache caches the persistent collections.

Q37. You are working in a Spring Boot application that needs to connect to a relational database. You want to configure the data source properties, such as URL, username, and password, in the application.properties file. Which of the following is the correct way to configure the data source properties in the application.properties file for an H2 in-memory database?

```
(A)
jdbc.url=jdbc:h2:mem:testdb
jdbc.username=admin
jdbc.password=admin123


(B)
```

```
db.datasource.url=jdbc:h2:mem:testdb
db.datasource.username=admin
db.datasource.password=admin123
```

```
(C)
spring.db.datasource.url=jdbc:h2:mem:testdb
spring.db.datasource.username=admin
spring.db.datasource.password=admin123
```

```
(D)
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=admin
spring.datasource.password=admin123
```

A37. D is correct . In a Spring Boot application, you typically configure data source properties in the application.properties or application.yml file. Spring Boot uses the spring.datasource prefix to identify data source-related properties.

Q38. In a spring application there are four classes some code segments are defined in the given options select the one which would contain the business logic.

```
(A)
public class ProductController {
   @PostMapping("/product")
   public Product createProduct(@RequestBody Product product) {
      return productService.saveProduct(product);
   }
}
```

```
(B)
public class ProductService {
   public Product saveProduct(Product product) {
      if (product.getPrice() > 0) {
        return productRepository.save(product);
      } else {
         throw new IllegalArgumentException("Price must be greater than zero");
      }
   }
}
```

```
(C)
public interface ProductRepository extends JpaRepository<Product, Long> {
  // Data access methods
}


(D)
public class ProductValidator {
   public boolean isValid(Product product) {
     // Validation logic
     return product.getPrice() > 0;
   }
}
```

A38. B is correct. The class ProductService, as shown in option B, is the most appropriate place for business logic in a Spring application. In this case, ProductService handles the core logic for validating and saving a product. The ProductController class in option A is responsible for handling incoming HTTP requests, while the ProductRepository interface in option C manages data persistence. The ProductValidator class in option D might be used for specific validation or utility functions but does not typically contain business logic.

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project