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"/>
```

Explaination: 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 not the correct 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"/>
```

Explaination: 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
```

Explaination: 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 life cycle methods?

(A) InitializingBean

(B) PostConstruct

(C) DisposableBean

(D) PreDestroy

Explaination: 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

Explaination: 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 while instantiating them

(D) BeanFactory supports the Annotation-based dependency Injection

Explaination: 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

Explaination: 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

Explaination: 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
    }
```

Explaination: 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
    }
```

Explaination: 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

Explaination: 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.

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 behaviour

Explaination: 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

Explaination: 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

Explaination: 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

Explaination: 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

Explaination: 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
}
```

Explaination: 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 a core feature of Spring framework in your application. Below are the three ways:

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 project:

(A) Only Option (1)

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

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

(D) All three

Explaination: 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

Explaination: 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

Explaination: 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;
        }
    }
```

Explaination: 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;
    }
```

Explaination: 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 ApplicationListener interface

Explaination: 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 considered to apply to BeanFactory as well.

(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 use the BeanFactory, since it offers all the features of the ApplicationContext.

Explaination: B is correct. According to Spring Framework documentation, option B is correct. Refer spring 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

Explaination: 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
}
```

Explaination: 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 with the EntityManager.

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

(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.

Explaination: 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

Explaination: 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

Explaination: 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 formatting.

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