1. **What is polymorphism?**

we can create functions or reference variables which behaves differently in different programmatic context.

1. **Why Strings are immutable?**
2. **To increate performance - String pool is nothing but a collection of some strings mostly unique**
3. **Provide security considerations**

**3.Why Char array is preferred over String for storing password?**

1. **Can we make String blank after using it? No, we cannot. We know that once a String is created, we cannot manipulate it e.g. you cannot change its content. Strings are final and immutable.**
2. **But char arrays are mutable, their content can be overwritten after use it. So your application shall use char [] to store password text, and after using the password, replace array content with a blank.**

**4.String Immutable?**

An [**immutable object**](https://howtodoinjava.com/java/related-concepts/how-to-make-a-java-class-immutable/) is **an object which state is guaranteed to stay identical over its entire lifetime**.

Normally **immutability in java is achieved through** following steps :

1. Don’t provide mutator methods for any field
2. Make all fields final and private
3. Don’t allow subclasses by declaring the class final itself
4. Return deep cloned objects with copied content for all mutable fields in class
5. **Difference between @Autowire and @Inject**

|  |  |  |
| --- | --- | --- |
| **@Inject** | **@Autowire** | **@Resource** |
| **AutowiredAnnotationBeanPostProcesser – inject dependencies** | **AutowiredAnnotationBeanPostProcesser – inject dependencies** | **CommonAnnotationBeanPostProcessor** |
| **standard annotation** | **spring specific annotation** | **standard annotation** |
| **by type** | **by type** | **by name -> by type -> qulifier** |

1. **Spring boot starter projects -> convenient dependency descriptors**
2. **Setting up data while startup of spring application?**

|  |  |  |
| --- | --- | --- |
| **Implements CommandLineRunner** | As mentioned in the [documentation](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/CommandLineRunner.html), multiple CommandLineRunner beans can be defined within the same application context and can be ordered using the @Ordered interface or @Order annotation. | Run() callback function |
| **Constructor Injection** |  |  |
| **By Implementing InitializingBean** afterPropertiesSet method |  |  |
| **@postConstruct** |  |  |
| **@ApplicationListener**  onApplicationEvent method override |  |  |
| **@EventListener** |  |  |
| **@Bean initMethod attribute** | @Bean(initMethod="latha") |  |
|  |  |  |

1. **SingleTon design pattern**

|  |  |
| --- | --- |
| Reflection | [enums](https://www.geeksforgeeks.org/enum-in-java/) are used because java ensures internally that enum value is instantiated only once  public enum GFG  {    INSTANCE;  } |
| Serializable  By implementing serializable readResolve() | class Singleton implements Serializable  {      ……        // implement readResolve method      protected Object readResolve()      {          return instance;      }  } |
| Cloneable | @Override    protected Object clone() throws CloneNotSupportedException    {      return super.clone();    } |
|  |  |
|  |  |

1. **Why we need application configuration?**

Configuration for applications vary from one environment to another

* You would want to connect to a different database or queues
* You would want to connect with different services
* You would want to configure less logging in production
* You might want to have different custom configuration

1. **How to read properties from application.properties?**

@Value("${welcome.message}") private String welcomeMessage; - Pick up the value for welcome.message from application configuration(application.properties) and assign it to this member variable.

1. **What is spring container and Spring configuration?**

Spring Configuration annotation indicates that the class has @Bean definition methods. So Spring container can process the class and generate Spring Beans to be used in the application

The interface org.springframework.context.ApplicationContext represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the aforementioned beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata. The configuration metadata is represented in XML, Java annotations, or Java code.

1. **Difference between prototype and singleton design patern?**

|  |  |
| --- | --- |
| **Single ton** | **prototype** |
| **Rest or state less objects** | **Statefull objects** |
| **Eg : DAO** | **Eg: Service** |
| dependencies are resolved at instantiation time | **Client needs to take care resource clearance**  **By using post bean processor** |

Trick1 - dependencies are resolved at instantiation time. This means that if you dependency inject a prototype-scoped bean into a singleton-scoped bean, a brand new prototype bean will be instantiated and then dependency injected into the singleton bean... but that is all.

Trick2 - The scopes that are described in the following paragraphs are only available if you are using a web-aware Spring ApplicationContextimplementation (such as XmlWebApplicationContext). If you try using these next scopes with regular Spring IoC containers such as theXmlBeanFactory or ClassPathXmlApplicationContext, you will get an IllegalStateException complaining about an unknown bean scope.

Trick3 - "When a bean is a singleton, only one shared instance of the bean will be managed, and all requests for beans with an id or ids matching that bean definition will result in that one specific bean instance being returned by the Spring container.

To put it another way, when you define a bean definition and it is scoped as a singleton, then the Spring IoC container will create exactly one instance of the object defined by that bean definition. This single instance will be stored in a cache of such singleton beans, and all subsequent requests and references for that named bean will result in the cached object being returned."

1. **How spring handling cyclic bean dependency?**

**Scenario : 1 constructor injection**

**When a class depends on b class and b depends on a, it will create circular dependency and stops container creation**

**Solution: 1. @Lazy annotation for dependency**

**2.Setter injection**

**3. @PostConstruct**

**4. ApplicationContestAware and InitializingBean**

1. **Handle multiple IOC containers?**
2. **Difference between BeanFactory and ApplicationContext?**

| **Feature** | **BeanFactory** | **ApplicationContext** |
| --- | --- | --- |
| Bean instantiation/wiring | Yes | Yes |
| Automatic BeanPostProcessor registration | No | Yes |
| Automatic BeanFactoryPostProcessor registration | No | Yes |
| Convenient MessageSource access (for i18n) | No | Yes |
| ApplicationEvent publication | No | Yes |

1. **What is single ton?**

In summation, Spring guarantees exactly one shared bean instance for the given id per IoC container

Reference: https://coderanch.com/t/563755/frameworks/Spring-Multiple-IoC-Containers-Singleton