Parte 1

**1. ¿Qué es Java y cuál es su importancia en el desarrollo de software?**

Java es un lenguaje de programación con el que podemos desarrollar cualquier aplicación, como videojuegos, aplicaciones web, aplicaciones móviles, aplicaciones de escritorio, etc.

Java a tomado relevancia debido a que nos permite crear múltiples aplicaciones y para diferentes plataformas (Windows, Linux, Mac), así como también cuenta con una gran cantidad de herramientas que nos permiten agilizar el desarrollo de software.

**2. ¿Qué es una clase en Java y cuál es su relación con los objetos?**

Una clase es una descripción de un conjunto de objetos similares, mientras que un objeto es una instancia de una clase.

**3. ¿Qué es un método en Java y cuál es su propósito?**

Un método es un conjunto de instrucciones dentro de una clase para realizar una tarea determinada.

**4. Explica qué es la herencia en Java y proporciona un ejemplo.**

La herencia es un mecanismo por el cual una clase permite heredar las características (atributos y métodos) de otra clase.

La implementación de la herencia se hace mediante la palabra reservada extends.

**Ejemplo:**

public class SeleccionFutbol

{

protected int id;

protected String Nombre;

protected String Apellidos;

protected int Edad;

// constructor, getter y setter

public void Concentrarse() {

...

}

public void Viajar() {

...

}

}

public class Futbolista extends SeleccionFutbol

{

private int dorsal;

private String demarcacion;

public Futbolista() {

super();

}

// getter y setter

public void jugarPartido() {

...

}

public void entrenar() {

...

}

}

public class Entrenador extends SeleccionFutbol

{

private String idFederacion;

public Entrenador() {

super();

}

// getter y setter

public void dirigirPartido() {

...

}

public void dirigirEntreno() {

...

}

}

**5. ¿Qué es una interfaz en Java y cómo se diferencia de una clase abstracta?**

Una interfaz Java tiene un conjunto de métodos abstractos y propiedades constantes que permiten activar la herencia múltiple. Los métodos son siempre públicos y tan solo se declaran, no se implementan.

Una interfaz es totalmente declarativa, mientras que la clase abstracta puede implementar métodos que serán heredados por otras clases. Además, las clases abstractas solo pueden heredarse una vez, mientras que las interfaces permiten la herencia múltiple.

**6. Describe qué es Spring Boot y cuál es su objetivo principal en el desarrollo de aplicaciones.**

Spring Boot es un framework para realizar aplicaciones backend, ya sea APIs Rest, microservicios o una aplicación del lado del servidor.

Y su objetivo de principal es agilizar el desarrollo de aplicaciones, ya que permite crear aplicaciones autónomas, no dependen de un servidor externo y la configuración que realizamos es mínima.

**7. ¿Cuál es la diferencia entre una aplicación Spring Boot y una aplicación Java tradicional?**

Las aplicaciones Spring Boot son autónomas, configuración automática y son fáciles de mantener con el tiempo debido a que están desacopladas, mientras que las aplicaciones Java tradicionales son monolíticas, es decir, contienen todo lo necesario para su ejecución, pero eso perjudica la agilidad de los desarrolladores a la hora de implementar cambios: para una sola modificación, deben rediseñar todo el sistema, probarlo y volver a implementarlo.

**8. ¿Qué es la inyección de dependencias en Spring Boot y por qué es importante?**

La inyección de dependencias es un patrón de diseño que tiene como objetivo tomar la responsabilidad de crear las instancias de las clases que otro objeto necesita y suministrárselo para que esta clase los pueda utilizar.

**9. ¿Qué es un Bean en Spring y cómo se configura en una aplicación Spring Boot?**

Un bean es un componente de software reutilizable que proporciona servicios, utilidades y funcionalidades a otros componentes de una aplicación.

Para configurar un bean en Spring Boot con anotar un método con @Bean.

**10. ¿Cuál es la función principal de Spring MVC y cómo se relaciona con Spring Boot?**

Spring MVC es un marco web basado en modelo, vista y controlador ampliamente utilizado para desarrollar aplicaciones web, debemos crear la configuración manualmente

Spring Boot utiliza internamente Spring MVC, pero la configuración es automática.

Parte 2

**¿Qué es Docker y cuál es su propósito en el desarrollo de aplicaciones?**

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente.

El propósito de Docker es reducir los recursos de infraestructura de las aplicaciones, así como el tiempo de desarrollo. Cualquier aplicación se puede ejecutar en cualquier equipo una vez que haya sido Dockerizada.

**¿Cuál es la diferencia entre una imagen y un contenedor en Docker?**

Una imagen de Docker contiene el código que se ejecutará, incluida cualquier definición para cualquier biblioteca o dependencia que el código necesite, mientras que un contenedor es una instancia en ejecución de una imagen de Docker.

**Explica qué es un Dockerfile y cuál es su función en la creación de imágenes.**

Un Dockerfile es un archivo que contiene una serie de instrucciones necesarias para crear una imagen de Docker.

**¿Qué es un registro de Docker (Docker registry) y cuál es su importancia en el ecosistema de Docker?**

Docker Registry es una forma habitual de almacenar y distribuir imágenes de Docker.

Docker Registry nos ayuda a mejorar el control de acceso y la seguridad de las imágenes de Docker almacenadas en su repositorio.

**¿Qué es Kubernetes y cuál es su objetivo principal en la administración de contenedores?**

Kubernetes es una plataforma de código abierto que se utiliza para administrar contenedores de Linux en entornos de nube privada, pública e híbrida.

Kubernetes automatiza las tareas operativas de la administración de contenedores e incluye comandos integrados para implementar aplicaciones, actualizarlas, escalarlas a fin de que se ajusten a tus necesidades, supervisarlas y mucho más. Kubernetes nos facilita la administración de las aplicaciones.

**¿Cuál es la diferencia entre un pod y un contenedor en Kubernetes?**

La diferencia es que un POD es un grupo de contenedores (como contenedores Docker) con almacenamiento y red compartidos.

**Describe qué es un servicio en Kubernetes y cómo se utiliza para exponer aplicaciones.**

Un servicio de Kubernetes es una desvinculación lógica para un grupo de PODs implementados en un clúster que comparten una misma función.

**¿Qué es un nodo (node) en el contexto de Kubernetes y cuál es su función?**

Un nodo es una máquina virtual o física, dependiendo del tipo de clúster. Cada nodo está gestionado por el componente máster y contiene los servicios necesarios para ejecutar pods.

Un nodo tiene la función de ejecutar PODs.

**Explica qué es un manifiesto de Kubernetes (Kubernetes manifest) y cómo se utiliza para definir la configuración de una aplicación.**

Es un conjunto de instrucciones en un archivo yaml o json que especifican cómo desplegar una aplicación al nodo o nodos en un cluster de Kubernetes.

**¿Cuál es la importancia de los volúmenes persistentes en Kubernetes y para qué se utilizan?**

Un volumen persistente ofrece un almacenamiento persistente que permite que sus datos permanezcan intactos, independientemente de si se terminan los contenedores a los que está conectado el almacenamiento.