PREGUNTAS EXAMEN JAVA REMOTO

GeoPagos

SONIA ABREGÚ

Que es y para que sirve el mecanismo de "synchronization". Ejemplo de diferentes aplicaciones (en método, en bloque)

La synchronization es un mecanismo utilizado en los entornos multihilos, el cual provee bloqueos para asegurar el acceso exclusivo a los recursos compartidos, permitiendo evitar cualquier corrupción en el estado o comportamiento inesperado de dichos recursos.

La synchronization en java solo es necesaria si los objetos compartidos son mutables (cambian su estado). En los casos donde los objetos sean inmutables o los hilos solo lean el valor de los recursos la synchronization no es necesaria.

- Ejemplo de synchronization en bloque

```
public class Singleton {
        private static volatile Singleton instance;

        public static Singleton getInstance() {
                if (instance == null) {
                        synchronized (Singleton.class) {
                                if (instance == null)
                                        instance = new Singleton();
                        }
                }
                return instance;
        }
}
```

- Ejemplo de synchronization en método

```
public class Counter {
        private static int count = 0;

        public static synchronized int getCount() {
                return this.count;
        }

        public static synchronized setCount(int count){
                this.count = count;
         }
}
```

¿Qué es una clase immutable?

Una clase immutable es una clase que no cambia de estado luego de ser instanciada, tanto atributos como clase son declaradas como final.

Un ejemplo podría ser una clase con atributos privados y ningún setter.

- Ejemplo de clase inmutable:

```
public final class Tipo {
        public final static Character A = 'A';
        public final static Character B = 'B';
        public final static Character C = 'C';
}
```

El uso más frecuente es para concurrencia, para poder asegurarse de que los threads están usando los mismos datos (el objeto en el mismo estado) y no se van a modificar.

Cómo esperaría la finalización del thread "t" para ejecutar el método "despuesDeThread()". Hacer la función o llamada de espera.

El método *join()* espera que el thread "t" finalice antes de continuar.

Ejemplo:

```
public class ExampleRunnable implements Runnable {
   public void run() {
      for (int i=0; i < 10; i++) {
         System.out.println(Thread.currentThread().getName() + " [" + i + "]");
      }
      System.out.println(Thread.currentThread().getName() + " {-- FIN --}");
   }
}

public class Main {
   public static void main(String args[]) throws InterruptedException {
      Thread t = new Thread(new ExampleRunnable());
      t.setName("Example_1");
      t.start(); // imprime del 0 al 9 + FIN
      t.join(); // el thread principal espera la finalización t para continuar
      despuesDeThread(); // se ejecuta después de la finalización del t
   }
}
```

Diferencia entre ArrayList y Vector

Sincronización

- Vector: es sincronizado, lo que significa que cualquier método que actúe sobre el contenido de Vector es threads safe pero esto lo hace menos performante.
- ArrayList: no es sincronizado, por lo que no es seguro en threads pero se puede hacer sincronizada como el Vector usando el método synchronizedList de la Java Collections Framework.

Crecimiento en capacidad

- Vector: cuando el numero de elementos insertados en el Vector sobrepasa la capacidad disponible el Vector dobla el tamaño de su array. En vector es posible especificar el incremento en espacio.
- ArrayList: calcula el incremento en espacio en forma automática, utiliza la fórmula (capacidad * 3) / 2 + 1.

¿Qué es un servlet?

Un servlet es un objeto java que admite peticiones a través del protocolo HTTP, reciben peticiones desde un navegador web, las procesan y devuelven una respuesta al navegador, normalmente en HTML.

¿Qué sucede al definir una variable como volatile?

Cuando se define una variable como volatile se está indicando que su valor será modificado por distintos threads. Los threads leerán el valor de la variable de la memoria principal y no lo guardaran localmente (como si hacen con el resto de las variables)

¿Qué es dependency of injection? ¿Para qué se usa?

La dependency of injection (DI) es un patrón de diseño que consiste en que las clases no sean las encargadas de instanciar los objetos declarados como atributos, sino que estos se inyectan mediante los métodos setters o mediante el constructor en el momento en que se crea la clase. Sin DI la clase necesitaría instanciar los objetos cada vez que se instancie ella misma.

DI permite desacoplamiento y reutilización de código. Al no ser necesario instanciar en una clase los objetos que necesita, sino que son inyectados, ante cualquier cambio en la clase inyectada no será necesario modificar nada en la clase que hace uso de ella.