Pregunta 2 Java

Pregunta 2:

Mi primer nombre es José por lo que escogeré a Java.

- a)
  - 1a) Java posee soporte nativo para el desarrollo de aplicaciones y programas que soporten/ejecuten código u acciones de manera concurrente. Entre las opciones para implementar concurrencia tenemos a la clase java.lang.Thread la cual puede ser extendida por una clase personalizada para poder hacer uso de los hilos. A su vez existe una interfaz llamada java.lang.Runnable que se encarga de abstraer el comportamiento y uso de la clase java.lang.Thread. Finalmente Java facilita la concurrencia con el paquete java.util.concurrent que es utilizado para implementar programas mas complicados con concurrencia.

```
// clase abstracta con constructor implicito

public class ClassWithThreads extends Threads {
  ...
}

----------------------------------------------------------------

public class ClassWithInterface implements Runnable {
  ...
}
```

2a) La base del control y creación de las tareas concurrentes en Java es mediante instancias de la clase Threads. Para poder crear Threads se debe extender la clase Threads o implementar la interface Runnable mencionadas anteriormente en alguna otra clase .

Para poder crear un Thread se utiliza la palabra “new” seguido de Thread y se le debe pasar una instancia de una clase que soporte threads y que adicionalmente tenga implementado el método run( ) . El método run( ) es aquel que es ejecutado cada vez que un hilo es inicializado, este método puede variar según las necesidades de la clase que extiende a Thread y por

ello es necesario que toda clase que extiende a Thread implemente su propio método run( ).

Sin embargo para que un hilo inicialice su ejecución no basta con solo crear una instancia del mismo, adicionalmente se debe llamar al método start( ) que básicamente se encarga de decirle al hilo que inicio su labor y llame al método run( ).

Adicionalmente existen otras propiedades interesante de los hilos que permiten conocer información de ellos y aun mas importante manejar su comportamiento entre ellos tenemos :

- getName : Retorna el nombre del hilo.
- getPriority : Retorna la prioridad del hilo.
- isAlive : Determina si un hilo sigue siendo ejecutado.
- join : Espera que un hilo termine su ejecución.
- sleep : Suspende al hilo por un periodo de tiempo.

```
public class MyClass extends Thread {
     public void run(){
     System.out.println("MyClass running");
   }
}

------------------------------------------------------------
public class MyClass implements Runnable {
  public void run(){
  System.out.println("MyClass running");
     }
}
```

Sobre el manejo de memoria compartida para los hilos en Java tenemos que existen dos :

- El heap que es el lugar donde esta la memoria compartida a la cual todos los hilos de un proceso tiene acceso.
- El stack que es la memoria privada para cada uno de los hilos de un proceso.

Finalmente para la comunicación de mensajes en los hilos de Java los usuarios/programadores deben definir/implementar sus propios canales de

comunicación, pero estos suelen ser implementados como objetos que son compartidos entre los hilos.

3a) La sincronización siempre ha sido un tema complicado para los programadores y en ese sentido Java no es la excepción. Si bien hoy en día existen clases como BlockingQueue y Executors que alivian en gran medida la complejidad de realizar sincronización de hilos en Java para los programadores, aun existen casos en los que es necesario “meter bien las manos en la masa” y utilizar los métodos nativos para sincronizar a los hilos, en ese sentido Java provee de los métodos wait( ) , notify( ) y notifyAll( ).

wait( ) pone en suspensión un hilo hasta que otro hilo llame al método notify( ) en el mismo bloque.

notify( ) se encarga de despertar a un hilo que ha llamado a wait( ) en el mismo bloque.

notifyAll( ) se encarga de despertar a todos los hilos que han llamado a wait en el mismo bloque.