SCJP – Capitulo 10

Hebras

Agenda

- Definiendo, instanciando e iniciando Hilos
- Código sincronizado
- Interacción entre Hilos

Que son los Hilos (Thread)

- Una instancia de java.lang.Thread
- Un Hilo de ejecución

- Cuando se trabaja con Hilos, muy pocas cosas son garantizadas
- Cada Hilo tiene su propio stack de ejecución

Definiendo un Hilo

```
public class UnHilo extends Thread {
    public void run(){
        //trabajo que debe hacer el Hilo
    }
}

public class UnEjecutable implements Runnable{
    public void run(){
        //trabajo que debe hacer el Hilo
    }
}
```

Creado un Hilo

- Si se eligió la estrategia de extender la clase Thread, se puede iniciar directamente.

```
UnHilo unHilo = new UnHilo();
unHilo.start();
```

- Si se eligió la estrategia de implementar Runnable, se necesita un objeto Thread,para iniciar el Hijo.

```
UnEjecutable unEjecutable = new UnEjecutable();
Thread unHilo = new Thread( unEjecutable );
unHilo.start();
```

El método start() inicia el Hilo en un proceso aparte (internamente ejecuta el método ***run()***)

Otros constructores de Hilos

- Thread()
- Thread(Runnable target)
- Thread(Runnable target, String name)
- Thread(String name)

Iniciando un Hilo

- Al ejecutar el método *start()*, sucede lo siguiente:
  - Una nueva hilo de ejecución es creado (es su propio stack)
  - El Hilo se mueve desde el estado "**nuevo**" al estado "**ejecutable**"
  - Cuando el Hilo tiene el chance para ejecutarse, es ejecutado el método ***run()***

*Nota: si se ejecuta el método **run()** directamente, no se iniciará un nuevo Hilo*

Métodos de Thread

- sleep(long milis) throws InterruptedException
  - Hace “dormir” el Hilo por la cantidad de milisegundos enviada por parámetro, una vez que finaliza el tiempo, vuelve al estado “ejecutable” (también se puede interrumpir su sueño, invocando al método interrupt())
- join() throws InterruptedException
  - El Hilo “A” espera a que Hilo “B” finalice y así seguir la ejecución (del Hilo “A”) (este método esta sobrecargado para recibir el tiempo en milisegundos que debe esperar)

Métodos de Thread

setPriority(int nuevaPrioridad)

- Las prioridades de los Hilos van de 1 a 10 (1 es menor), por defecto la prioridad es 5. Este método debe ser invocado antes del método *start()*, también se pueden usar las constantes:
  - Thread.MIN_PRIORITY = 1
  - Thread.NORM_PRIORITY = 5
  - Thread.MAX_PRIORITY = 10

- yield()
  - Sugiere la ejecución de otro Hilo que se encuentre en el estado "ejecutable" (el Hilo a elegir será de acuerdo a su prioridad)

Estados de un Thread

- **Nuevo** : Se ha creado la instancia de Thread, pero aun no se llama al método *start()*
- **Ejecutable** : Es elegible para ser ejecutado (normalmente después de *start()*)
- **Ejecutándose** : Actualmente se esta ejecutando
- **Esperando / bloqueado / durmiendo** : El Hilo esta vivo, pero no es elegible para ejecutar
- **Muerto** : ha finalizado el método *run()*

Sincronizando los Objetos

- synchronized: Permite asegurar que una porción de código sólo podrá ser accedido por un Hilo a la vez.
  - Puede ser utilizado como modificador de método
  - Puede ser utilizado para sincronizar un cierto sector de un método (o sincronizar externamente a otro objeto)

```
synchronized ( InstanciaDeObjeto){
    //Codigo sincronizado
}
```

Sincronizando los Objetos

```
public class Cuenta {
    private int saldo;
    public synchronized void depositar(int monto){
        //todo el código del método esta sincronizado
        saldo += monto;
    }
}
```

En ambos clases se esta sincronizando el método depositar

```
public class Cuenta {
    private int saldo;
    public void depositar(int monto){
        //si aquí hubiera código, no estaría sincronizado
        synchronized (this){
            saldo += monto;
        }
    }
}
```

En esta clase particularmente, sólo una porción del código esta sincronizado

Sincronizando los Objetos

```
public class Cuenta {
    private int saldo;
    public void depositar(int monto){
        saldo += monto;
    }
}
public class ClienteCuenta extends Thread {
    private Cuenta cuenta;
    public ClienteCuenta (Cuenta cuenta){
        this.cuenta = cuenta;
    }
    public void run(){
        synchronized (cuenta){
            cuenta.depositar(150);
        }
    }
}
```

La clase cuenta no tiene sus métodos sincronizados…

La clase ClienteCuenta, se asegura de sincronizar externamente la instancia del objeto cuenta, antes de realizar algún tipo de operación sobre ella

Interacción entre Hilos

- La interacción entre Hilos se puede lograr con los métodos *wait()*, *notify()* y *notifyAll()* (de la clase Object)
- Estos métodos se deben invocar desde un contexto sincronizado
- No se puede ejecutar alguno de estos métodos desde otro objeto, a menos que obtenga el bloque mediante:
  - synchronized (otroObjeto) { … }

Interacción entre Hilos

- wait() : bloquea el hilo actual hasta que se realice una llamada a notify() o notifyAll()
  - wait() tiene versiones sobrecargadas que reciben por parámetro el numero de milisegundos a esperar
- notify() : Notifica a un sólo Hilo que están esperando
- notifyAll() : Notifica a todos los hilos que se encuentran esperando