

Threads

Sumario

- 1 Creación de threads
 - ◆ 1.1 Herdar da clase `java.lang.Thread`
 - ◆ 1.2 Implementación da interface `java.lang.Runnable`
- 2 Instanciación de threads
- 3 Os estados dun thread
- 4 Acceso concorrente e sincronización de threads

Creación de threads

Para crear un thread necesitamos que a clase teña o método `run()`. Isto pódese conseguir de dúas formas, herdando da clase `Thread` ou implementando o interface `Runnable`.

Herdar da clase `java.lang.Thread`

O xeito máis simple de definir código para que se execute nun thread independente é o seguinte:

1. Derivar da clase `Thread`.
2. Sobrescribir o método `run()` que, de feito, na clase `Thread` está baleiro (ademais, o método `run()` pode estar sobrecargado)

Isto faise da seguinte forma:

```
class MeuThread extends Thread {
    public void run() {
        System.out.println("Traballo importante executándose en MeuThread");
    }
}
```

A principal limitación desta solución é que se herdamos de `Thread` non poderemos herdar de ningunha outra clase.

Implementación da interface `java.lang.Runnable`

Mediante a implementación de `Runnable` podemos herdar de calquera outra clase, xa que é unha interface:

```
class ThreadRunnable implements Runnable {
    public void run() {
        System.out.println("Traballo importante executándose en ThreadRunnable");
    }
}
```

Independentemente do mecanismo que se use hai que instanciar un thread dende o código para poder executalo.

Instanciación de threads

Se derivamos da clase `Thread` a instanciación é inmediata:

```
MeuThread t = new MeuThread();
```

Se implementamos a interface `Runnable` tamén é moi simple:

```
ThreadRunnable r = new ThreadRunnable();
Thread t = new Thread(r);
```

O seguinte código de exemplo amosa como crear un thread e poñelo a durmir:

```
class Contador extends Thread {
    public void run() {
        for(int i = 1; i<=100; ++i) {
```

```

        System.out.print(i + " ");
        if(i % 10 == 0)
            System.out.println("Ola, meu!");
        try {
            Thread.sleep(1000);
        } catch(InterruptedException e) {}
    }
}
public static void main(String [] args) {
    new Contador().start();
}
}

```

Nun thread só se pode chamar ao método `start()` unha vez. Se se chama máis dunha vez lanzarase unha `RuntimeException`. Cando se instancia un `Thread` non será tal até que se invoque ao método `start()`. Cando un obxecto de tipo `thread` existe pero non foi iniciado co método `start` non se considera que estea "vivo". Existe un método da clase `Thread` chamado `isAlive` que devolve `true` se o `thread` foi arrancado con `start()`. Executar o método `start()` non garante que se execute inmediatamente o método `run()`, simplemente, o `thread` pasará ao estado preparado. **Non se debe chamar ao método `run()` directamente**, xa que se executará o código deste método sen que interveña o planificador. É o método `start()` o que rexistra o `thread` no planificador de `threads`.

Os estados dun thread

Unha vez que un `thread` se inicia (**start**) entrará sempre no estado de executable (**runnable**). O método `start` chama ao método `run()`.

O planificador de `threads` pode cambiar os estados dun `thread` entre executándose (**running**) e executable (`runnable`).

Só pode haber un `thread` executándose ao mesmo tempo, aínda que pode haber moitos en estado executable.

A orde en que foron iniciados os `threads` non determina como se executarán.

Polo tanto, os estados dun `thread` poden ser:

- **En execución** (`running`). O `thread` está activo e ten asignada a CPU.
- **Preparado** (`runnable`). O `thread` está activo pero non ten asignada a CPU.
- **Bloqueado**. O `thread` está en espera de que outro `thread` elimini o bloqueo, momento no que pasará ao estado preparado.

Acceso concorrente e sincronización de threads

Os métodos sincronizados úsanse para evitar que máis dun `thread` poda acceder a un método dun obxecto que teña código crítico. Por crítico entendemos calquer código que necesite unha execución exclusiva para evitar posibles inconsistencias provocadas pola execución de dous ou máis `threads` (por exemplo, os movementos que poden facer dúas persoas ao mesmo tempo sobre unha conta bancaria). Só se poden sincronizar métodos, non variables.

A palabra clave **synchronized** pódese usar como modificador dun método ou para indicar o inicio dun bloque de código sincronizado. Para sincronizar un bloque de código (noutras palabras, un anaco de código menor que un método), hai que especificar un argumento que sexa o obxecto que se quere bloquear coa sincronización. Por exemplo:

```

class SyncTest {
public void faiAlgo() {
    System.out.println("non sincronizado");
    synchronized(this) {
        System.out.println("sincronizado");
    }
}
}
}

```

Só un `thread` pode acceder a código sincronizado dunha instancia particular. Con todo, outros moitos `threads` poden acceder ao código non sincronizado do mesmo obxecto.

Cando un `thread` pasa a estado durmido (`sleep`) o código que teña bloqueado seguirá así até que desperte.