¿Cómo funciona synchronized en Java?

Mucha gente que usa Java no tiene claro cómo funciona realmente la palabra clave synchronized, es realmente muy fácil si se tienen en cuenta algunas cosas.

En primer lugar, usar synchronized en un método de **instancia** es lo mismo que poner un bloque de synchronized(this){} que contenga todo el código del método. Es decir,

```
public synchronized void metodo() {
  // codigo del metodo aca
}
```

es lo mismo que

```
public void metodo() {
  synchronized(this) {
    // codigo del metodo aca
  }
}
```

Si el método es de **clase** entonces es lo mismo pero el bloque de synchronized se aplica a la clase. Ejemplo, si el método está en la clase MiClase entonces esto

```
public static synchronized void metodo() {
  // codigo del metodo aca
}
```

es lo mismo que

```
public static void metodo() {
  synchronized(MiClase.class) {
    // codigo del metodo aca
  }
}
```

Habiendo dicho esto, **el problema se reduce a entender cómo funciona el synchronized de un bloque** porque, como vimos, el synchronized en métodos de instancia o de clase son simplemente casos especiales de esto.

El bloque synchronized lleva entre paréntesis la referencia a **un objeto**. Cada vez que un thread intenta acceder a un bloque sincronizado le pregunta a **ese** objeto si no hay algún otro thread que ya tenga el *lock* para **ese** objecto. En otras palabras, le pregunta si no hay otro thread ejecutando algun bloque sincronizado con **ese** objeto (y recalco que es **ese** objeto porque en eso radica la clave para entender el funcionamiento)

Si el lock está tomado por otro thread, entonces el thread actual es suspendido y puesto en espera hasta que el lock se libere. Si el lock está libre, entonces el thread actual toma el lock del objeto y entra a ejecutar el bloque. Al tomar el lock, cuando venga el proximo thread a intentar ejecutar un bloque sincronizado con **ese** objeto, será puesto en espera. ¿Cuándo se libera el lock? Se libera cuando el thread que lo tiene tomado sale del bloque por cualquier razón: termina la ejecución del bloque normalmente, ejecuta un return o lanza una excepción.

Es importante notar una vez más que el **lock es sobre un objeto en particular**. Si hay dos bloques synchronized que hacen referencia a distintos objetos (por más que ambos utilicen el mismo nombre de variable), la ejecución de estos bloques **no** será mutuamente excluyente.

Conclusiones:

De todo esto se puede concluir que un método synchronized de instancia puede ser ejecutado al mismo tiempo que uno de clase.
Además, dos threads pueden ejecutar el mismo método de instancia al mismo tiempo aunque este sea synchronized si la invocación se hace a dos objetos diferentes.

Por último, todos los métodos synchronized de instancia son mutuamente excluyentes entre sí. Es decir que, dado un objeto compartido por más de un thread, sólo uno de ellos puede acceder en un determinado momento a uno de esos métodos de instancia. Los otros threads deberán esperar aunque hayan querido acceder a otro de los métodos (porque todos están synchronizados con el mismo this).