```
public class Clock {

    private long time;

    public Clock(){
       this.time = System.currentTimeMillis();
    }

    public void reset(Clock r){
       r.time = System.currentTimeMillis();
    }

    public static void sinc(Clock r1, Clock r2){
(1)    reset(r1);
(3)    r2 = r1;
    }

    public static void main(String[] args) {
       Clock clock1 = new Clock();
       Clock clock2 = null;
(2)    sinc(clock1, clock2);
       System.out.println(clock1.equals(clock2));
(4)    clock2.reset();
       System.out.println(clock2.equals(clock1));
    }

}
```

(1) El método `reset()` está definido para un entorno de instancia y no puede ser utilizado en un método estático.
SOLUCIÓN: definir el método reset como static, para que funcione en un entorno de clase

(2) Los parámetros son pasados por valor, luego de ejecutar el método `sinc()`, clock2 seguiría valiendo NULL
SOLUCIÓN: El método sinc podría devolver una referencia a un objeto Clock “sincronizado”, sin embargo en este caso sería una referencia a clock1 (por la utilización del operador de asignación “=”), el método sinc en un entorno de clase debería crear un nuevo objeto Clock, igualar el valor del atributo time de ambos relojes y devolver la referencia al nuevo objeto.
También podría crearse un método sinc que funcione en un entorno de instancia, el cual recibe un Clock como parámetro e iguala el valor del atributo `time` del objeto Clock pasado como parámetro con el de la instancia que ejecutó el método.
Por otro lado, existe una cuestión de diseño, que depende del comportamiento que se esté buscando para un objeto clock, y es que quizá no sería correcto que un método que sincroniza dos Clock resetee al Clock pasado por parámetro.

(3) La operación de asignación, en este caso, estaría creando dos referencias al mismo objeto, conceptualmente, no son dos relojes distintos sincronizados, sino el mismo reloj dos veces. Cualquier modificación en clock1 modificaría también clock2 (siempre dentro del scope del método).
SOLUCIÓN: mencionada en (2).

(4) El método `reset()` espera un parámetro
SOLUCIÓN: Crear un método reset a nivel de instancia que no reciba parámetros, y que resetee el valor de `time` de la instancia del objeto que lo llamó.

Aparte de los errores resaltados, también podría recomendarse el uso de accessors y nombres de parámetros más autoexplicativos.

Posible solución

```
public class Clock {

    private long time;

    public void setTime(long time){
       this.time = time;
    }

    public long getTime(){
       return this.time;
    }

    public Clock(){
       this.time = System.currentTimeMillis();
    }

    //modifico el método reset original para que funcione en un
entorno de clase (agrego static)
    public static void reset(Clock r){
       r.time = System.currentTimeMillis();
    }

    //creo un método reset que funcione en un entorno de instancia
    public void reset(){
       this.setTime(System.currentTimeMillis());
    }

    public static Clock sinc(Clock r1, Clock r2){
```

```
      //con el código original esto no compila porque reset no era
un método estático
      reset(r1); //podría no corresponder
      r2 = new Clock();
      r2.time = r1.time;
      return r2;
    }

    public void sinc(Clock r1){
      this.setTime(r1.getTime());
    }

    public static void main(String[] args) {
      Clock clock1 = new Clock();
      Clock clock2 = null;
      sinc(clock1, clock2);
      System.out.println(clock1.equals(clock2));
      //en el código original no está definido un método reset que
funcione a nivel de instancia
      clock2.reset();
      System.out.println(clock2.equals(clock1));
    }

}
```

Luego de corregir problemas de compilación los resultados devueltos serian:

`false`

`NullPointerException` (por lo mencionado en (2))

Si se corrige el código para que clock2 sea una referencia a clock1, como está intencionado en el código original, los resultados serían

`true`

`true`

Si no se redefine el método `equals()` la comparación verifica que sea el mismo objeto, y en este caso, lo es.

Con la solución propuesta, donde la sincronización de clocks mantiene diferenciadas las instancias de los objetos Clock

```
clock2.sinc(clock1);
```

Los resultados serían

```
false

false
```

Ya que las instancias son distintas. Si lo que se desea es que la comparación verifique el valor del campo `time` de los Clock, lo más adecuado sería redefinir el método `equals(Clock r1)`

```
public function equals(Clock r1){
      return this.getTime()==r1.getTime();
}
```