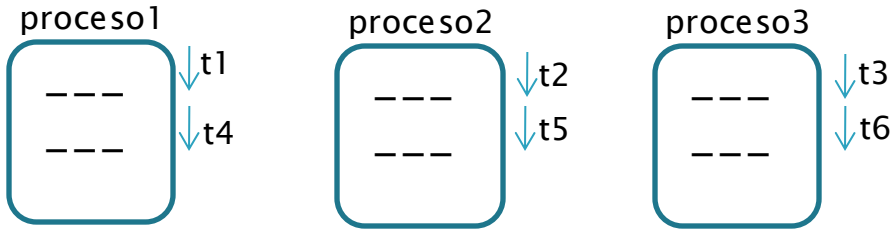
Concurrencia/Multitarea

Definición

- Ejecución de varios procesos simultáneamente.
- A diferencia de la ejecución secuencial, el gestor de multitarea de Java reparte el tiempo de CPU entre los procesos



Implementación de tareas

- Para implementar el código de las tareas existen dos posibilidades en la multitarea clásica:
  - Extender la clase Thread
  - Implementar la interfaz Runnable

Extensión de la clase Thread

- Proporciona el método run(), que debe ser sobrescrito para incluir el código de la tarea/tareas.

```
public class Tarea extends Thread{
    public void run(){
        //código de la tarea
    }
}
```

Se define una o varias subclases de Thread, dependiendo si las tareas realizarán la misma función o diferente

- Para poner en ejecución concurrente las tareas, se llamará al método start() de Thread, que invoca al gestor de multitarea de Java para que ponga los objetos en ejecución concurrente:

```
Tarea t1=new Tarea();
Tarea t2=new Tarea();
t1.start();
t2.start();
```

Interfaz Runnable

- Incluye el método run() y permite a las clases que la implementan poder heredar otras clases:

```
public class Tarea implements Runnable{
    public void run(){
        //código de la tarea
    }
}
```

- Para poner las tareas en ejecución, se crearán instancias de Thread a partir del objeto Runnable:

```
Tarea obj=new Tarea();
Thread t1=new Thread(obj);
Thread t2=new Thread(obj);
t1.start();
t2.start();
```

Revisión conceptos

Si MyThread es una subclase de Thread y MyRun una implementación Runnable, indica cual de los siguientes bloques crea de forma correcta una serie de tareas en ejecución concurrente:

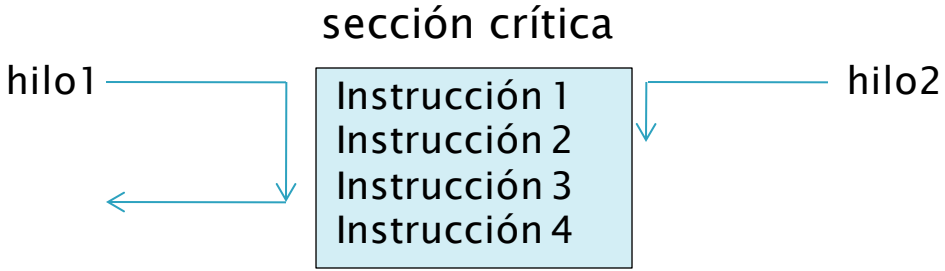
A. (new MyRun()).start();(new MyThread()).start();
B. (new MyThread()).run();(new MyThread(MyRun)).start();
C. (new MyThread()).start();(new Thread(new MyRun())).start();
D. (new Thread(MyThread).start());(new MyRun()).start();

Respuesta

La respuesta correcta es la C. Para crear un objeto de una subclase de Thread y ponerlo en ejecución concurrente, si instancia la clase y se llama a start(). en el caso de una implementación de Runnable, se debe instanciar Thread a partir del Runnable.

Condiciones de carrera

- Se da cuando dos o más hilos acceden a un recurso compartido.
- Si un hilo debe completar un proceso con el recurso (sección crítica) antes de que otro lo manipule, se puede producir una situación anómala



- Solución: Sincronización

Sincronización

- Consiste en bloquear el acceso a un bloque de código a otros hilos, hasta que el hilo que entró primero complete su ejecución
- Se utiliza la palabra *synchronized* para delimitar el bloque sincronizado.
- Cuando un hilo entra en el bloque sincronizado, adquiere el monitor del objeto y no lo suelta hasta completarlo

recurso compartido

```
synchronized(obj){
    int value=obj.getValue();
    value++;
    obj.setValue(value);
}
```

La palabra synchronized se puede utilizar en la definición de un método para sincronizar el método completo

Revisión conceptos

Explica porqué la siguiente implementación de run() no aplica correctamente la utilización de bloques sincronizados para evitar condiciones de carrera

```
public void run(){
   StringBuilder ob=new StringBuilder();
   synchronized(ob){
     ob.append(this.toString());
   }
}
```

Respuesta

Cada hilo tendrá su propia instancia de StringBuilder, por lo que no se estará sincronizando acceso a ningún recurso compartido. Lo lógico sería que la variable StringBuilder fuera, por ejemplo, un atributo estático para ser compartido por todos los procesos