SISTEMAS DE RIEGO

1) (**THREAD-RUNNABLE**) Implementa una clase Java (`SistemaRiego`) que simule el comportamiento de un **sistema de riego automático** mediante herencia de la clase `Thread`. La clase debe cumplir con los siguientes requisitos:

   a) Atributo `zona`: cadena de caracteres que representa la zona del jardín que se va a regar.
   b) Atributo `duracionRiego`: número entero que indica el tiempo (en segundos) que tarda en regar la zona.
   c) Constructor sin parámetros que inicialice la zona como `"ZonaGeneral"` y la duración del riego como 5 segundos.
   d) Constructor con parámetros que permita establecer la zona y la duración del riego al crear el objeto.
   e) Método `regar()` debe:
      i) Imprimir un mensaje indicando qué zona ha comenzado a regarse.
      ii) Esperar el tiempo especificado que tarda el sistema en realizar el riego.
      iii) Imprimir un mensaje indicando que la zona ha terminado de regarse.
   f) Al iniciar el hilo de cada sistema de riego, debe ejecutarse el proceso de riego desde donde corresponda.

2) Implementa una clase `ControlRiego` que, desde su método `main()`, instancie dos objetos de la clase `SistemaRiego` y los ponga a trabajar.

---

3) (**JOIN**) Re-implementa la clase `SistemaRiego` con las siguientes especificaciones:

   a) Debe implementar la interfaz `Runnable` en vez de extender de `Thread`.
   b) Atributos:
      i) `zona`: cadena de caracteres que representa la zona del jardín que se va a regar.
      ii) `duracionRiego`: número entero que indica el tiempo (en segundos) que tarda en regar la zona.
      iii) `dependencia`: objeto de tipo `Thread` que representa otro hilo del que este sistema depende. Si está informado, debe esperar a que ese hilo termine antes de comenzar el riego.
   c) Constructores: los mismos que en el apartado 1, con la salvedad de que el nuevo atributo `dependencia` se iniciará, en ambos, como `null`.
   d) Método `setDependencia`: permite establecer el hilo del que depende cada sistema de riego antes de comenzar su ejecución.
   e) Método `regar()` debe:
      i) Comprobar si su ejecución depende de otro hilo, en cuyo caso debe esperar a que éste finalice su ejecución.
      ii) Imprimir un mensaje indicando qué zona ha comenzado a regarse.
      iii) Esperar el tiempo especificado que tarda el sistema en realizar el riego.
      iv) Imprimir un mensaje indicando que la zona ha terminado de regarse.
   f) Al iniciar el hilo de cada sistema de riego, debe ejecutarse el proceso de riego desde donde corresponda

4) Re-Implementa una clase `ControlRiego` que, desde su método `main()`, instancie tres objetos de la clase `SistemaRiego` (`Z1`, `Z2` y `Z3`), cree y lance los hilos que correspondan:

   - `Z1`: zona: `Zona-Z1"`, duración: 3s, dependencia: `null`
   - `Z2`: zona: `"Zona-Z2"`, duración: 4s, dependencia: `null`
   - `Z3`: zona: `"Zona-Z3"`, duración: 2s, dependencia: `Z2`

HOSPITAL

1. (INTERRUPCIONES) Implementar un programa en Java que simule la interacción entre dos hilos mediante el uso de interrupciones.

a. Clase `Medico`

- Atributo `nombre` para identificar al profesional.
- Extiende de `Thread`.
- Al ejecutarse, debe mostrar infinitamente el mensaje:
  `"Estoy atendiendo pacientes sin parar"`
- Si recibe una interrupción, debe capturarla y mostrar el mensaje:
  `"Me han llamado para una reunión urgente"`

b. Clase `CoordinadorUrgencias`

- Atributo `nombre` para identificar al coordinador.
- Atributo `doctor`: objeto de tipo `Medico`.
- También extiende de `Thread`.
- Al ejecutarse, permanece dormido durante 5 segundos (simulando que revisa los informes de urgencias).
- Luego, interrumpe al `doctor` para que deje de atender y vaya a una reunión urgente.

c. Clase `Hospital`

- Desde su método `main()`, se deben crear dos objetos:
  - Uno de tipo `Medico`
  - Otro de tipo `CoordinadorUrgencias`
- Se lanzan ambos como hilos.

---

2. (MONITOR) En la unidad de urgencias, hay un solo respirador disponible. Dos médicos están atendiendo casos críticos y ambos intentan usar el respirador al mismo tiempo. Para simular esta situación y evitar conflictos, se pide implementar las siguientes clases en Java:

Clase Respirador

- Atributo privado `enUso` de tipo `boolean`, que indica si el respirador está siendo usado.
- Método liberar() debe quedar en espera mientras el respirador esté libre. Una vez que alguien la haya utilizado, cambiará el valor del atributo false y mostrará el mensaje: "Respirador Libre".
- Método usar() debe cambiar el valor del atributo enUso a true y mostrar el mensaje: "Respirador ocupado".

Clase Urgencias

- Crea una clase `Urgencias` que en su método main incluya dos médicos y los lance a trabajar.
- MedicoA recibirá una instancia del respirador e intentará liberarlo todo el rato.
- MedicoB recibirá la misma instancia del respirador e intentará usarlo todo el rato.

Añade la sincronización necesaria para garantizar que, si un cirujano está usando el respirador, el otro debe esperar. Además, todos los hilos deben ver el cambio de estado del respirador inmediatamente.