Programación Concurrente

Repaso

Cuando se crea una clase, se crea una "molde" donde se definen atributos y métodos. Cuando se crea un objeto, se instancia una clase, mediante el operador **new**. Todos los objetos de la misma clase tienen los mismos atributos y métodos. Los valores concretos de cada atributo de cada objeto pueden ser diferentes y conforman su estado.

Herencia:

- Nos permite definir una clase como una ampliación de otra.
- Mecanismo que nos ofrece una solución al problema de duplicación de código.
- La característica esencial de esta técnica es que necesitamos describir las características comunes sólo una vez.
- Se denomina relación «**es-un**». La razón de esta nomenclatura radica en que la subclase es una especialización de la superclase.
- La herencia nos permite crear dos clases que son bastante similares evitando la necesidad de escribir dos veces la parte que es idéntica.
- Una **superclase** o clase padre, es una clase que es extendida por otra clase, o que otras clases heredan.
- Una **subclase**, clase hija, es una clase que extiende o amplia a otra clase. Hereda todos los campos y los métodos de la superclase.
- Más de una subclase puede heredar de la misma superclase y una subclase puede convertirse en la superclase de otras subclases. Las clases que están vinculadas mediante una relación de herencia forman una jerarquía de herencia.
- La herencia es una técnica de abstracción que nos permite categorizar las clases de objetos bajo cierto criterio y nos ayuda a especificar las características de estas clases.
- Palabra clave **extends** crea la clase derivada desde la clase base, es decir, define la relación de herencia.
  La frase «extends<SuperClase>» especifica que esta clase es una subclase de la clase SuperClase. La subclase define sólo aquellos campos que son únicos para los objetos de su tipo. Los campos de la superclase se heredan y no necesitan ser incluidos en el código de la subclase.
- **Los constructores pueden llamar a otros constructores**. Se debe utilizar **super** para invocar a un constructor de la clase padre. Se debe utilizar **this** para invocar a un constructor dentro de la clase. Cualquiera de las dos opciones debe ser la primera acción realizada por el constructor.

Excepciones

Un programa correcto es aquel que actúa de acuerdo a su especificación.

Un programa confiable correcto y además tiene un comportamiento previsible, es decir actúa razonablemente no sólo en situaciones normales sino también en circunstancias anómalas, como por ejemplo fallas de hardware. Desde el punto de vista de la aplicación las situaciones consideradas normales dependen del diseñador que analiza el problema.

Una excepción es un evento anormal durante la ejecución que puede provocar que una operación falle; el software que previene este tipo de circunstancias se dice "tolerante a las fallas".

Se puede reparar la falla, capturando la excepción y alcanzando un estado que permita continuar la ejecución pero a veces, el manejo de la excepción se reduce a mostrar un mensaje, porque la situación no es recuperable; en ese caso la operación falla y probablemente el programa se aborta.

Una excepción es una situación anormal o poco frecuente que requiere ser capturada y manejada adecuadamente. Las excepciones pueden ser predefinidas por el lenguaje o definidas por el programador.

La idea es que el programador establezca un manejador que especifique las acciones a realizar cuando se captura una excepción; organizar un programa en secciones para el caso normal y para el caso excepcional.

Tener en cuenta: las excepciones simplifican el desarrollo, prueba y mantenimiento, pero no se debe abusar de ellas.

Terminología

- Lanzar o disparar una excepción (throwing)
- Manejar o capturar una excepción (handling/catching)
- Se responde a una excepción ejecutando una parte del programa escrita específicamente para esa excepción
- El caso normal es manejado en un bloque try
- El caso excepcional es manejado en un bloque catch
- El bloque catch recibe un parámetro de tipo Exception (generalmente llamado e)
- Si se dispara una excepción, la ejecución del bloque try se interrumpe y el control pasa al bloque catch cercano al bloque try, caso contrario el bloque catch es ignorado
- Exception es la clase base de todas las excepciones y cada excepción hereda el método getMessage (este método carga el string dado al objeto-excepción cuando fue lanzada la excepción, por ejemplo: throw new Exception ("Mensaje cargado")).

Un método puede lanzar más de una excepción; los bloques catch inmediatamente después del bloque try son analizados en secuencia para identificar el tipo de excepción. El primer bloque catch que maneja ese tipo de excepción es el único que se ejecuta

Se deben colocar los bloques catch en orden de especificad: los más específicos primero

Se puede agregar un bloque finally después de los bloques try/catch. El bloque finally se ejecuta sin importar si el bloque catch se ejecuta,

Tres Posibilidades para un bloque try-catch-finally

- El bloque try se ejecuta hasta el final sólo si ninguna excepción es lanzada. El bloque finally se ejecuta después del bloque try.
- Una excepción es lanzada en el bloque try y atrapada en el macheo del bloque catch. El bloque finally se ejecuta después del bloque catch.
- Una excepción es lanzada en el bloque try y no existe match en el bloque catch. El bloque finally se ejecuta antes de que el método termine. El código que está después del bloque catch pero no en el bloque finally no sería ejecutado en esta situación.

Visibilidad

Variables de instancia:

- Públicas: acceso fuera del ámbito de la clase
- Privadas: acceso sólo dentro de la clase
- Protegidas: acceso dentro de la clase y sub-clases

Métodos:

- Los métodos privados no son heredados.
- Los métodos redefinidos no pueden ser menos accesibles
- Si una clase redefine un método de la clase base, la versión en la clase derivada reemplaza a la de la clase base.
- **Si A es una clase derivada de la clase B, entonces A es miembro de ambas clases, A y B.**

Clase Abstracta	Clase Concreta	Interfaz
No se pueden crear objetos. Puede tener métodos abstractos Describe atributos (variables / constantes) y comportamiento (métodos concretos / métodos abstractos) común a sus subclases.	Se pueden crear objetos. No puede tener ningún método abstracto. Todos los métodos están implementados. Puede tener implementaciones diferentes en sus subclases.	No se pueden crear objetos. Todos los métodos son **abstract** (ninguno está implementado) Puede contener atributos, pero siempre declarados **static final** (constantes)

Aclaraciones:

- Como Java no permite herencia múltiple – una clase sólo puede extender una superclase. Esto dificulta que una clase se adecue a más de un comportamiento.
- Una interfaz, por el contrario, permite que una clase implemente una o más interfaces para resolver el problema de mezclar diversos comportamientos en un mismo tipo de objeto.
- Un método se compone de dos (2) partes básicas; encabezado (declaración) y cuerpo. En la declaración se encuentran:
  - ❖ La accesibilidad (que otras clases y objetos pueden invocar al método) private, public, protected.
  - ❖ El tipo de retorno, por ejemplo,void, int, String.
  - ❖ Nombre del método (identificador)
  - ❖ La cantidad y el tipo de parámetros que acepta el método.

  El cuerpo de un método va encerrado entre llaves { … } y contiene las sentencias algorítmicas que definen una funcionalidad.
  Un método abstracto es un método sin cuerpo.

Concurrencia

La **programación concurrente** (PC) se encarga del estudio de las nociones de ejecución concurrente, así como de sus problemas de comunicación y sincronización.

- Ventaja: Velocidad de ejecución. Al subdividir un programa en procesos, éstos se pueden "repartir" entre procesadores o gestionar en un único procesador según importancia.
- Desventaja: Más complicado de programar.

Sistema monoprocesador: La concurrencia se produce gestionando el tiempo de procesador para cada proceso.

Sistemas multiprocesador: Un proceso en cada procesador.

- Con memoria compartida (procesamiento paralelo) se caracterizan por ser fuertemente acoplados. La sincronización y comunicación entre procesos se suele hacer mediante variables compartidas. Los procesadores comparten memoria y reloj.
  Ventaja: aumento de velocidad de procesamiento con bajo coste.
  Desventaja: son escalables sólo hasta decenas o centenares de procesadores.
- Memoria local a cada procesador (sistemas distribuidos) se caracterizan por ser débilmente acoplados. Múltiples procesadores conectados mediante una red. Los procesadores no comparten memoria ni reloj. Los sistemas conectados pueden ser de cualquier tipo. Escalabilidad ilimitada, mayor fiabilidad, alta disponibilidad, aumento de velocidad de ejecución (ej. Internet).

Cuando se habla de concurrencia en programación, se habla de la técnica de hacer que un programa haga más de una cosa a la vez.

Si bien esquemas más distribuidos generan una mejor distribución del trabajo, también provocan mayor necesidad de comunicación.

Proceso:

- Programa secuencial: un solo flujo de control que ejecuta una instrucción y cuando esta finaliza ejecuta la siguiente.
- PROCESO: programa secuencial independiente de las acciones realizadas por otros procesos.
- Los procesos cooperan y compiten. Por tanto, necesitan tareas de colaboración y sincronización.
- Dos procesos son concurrentes cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última.

Concurrencia Vs Paralelismo

Un sistema se dice concurrente si puede soportar 2 o más actividades en progreso a la vez.

Un sistema se dice paralelo si puede soportar 2 o más actividades ejecutándose simultáneamente.

Un programa concurrente tiene múltiples hilos de control lógicos que pueden o no ejecutarse en paralelo.

Un programa paralelo puede ejecutarse más rápidamente que un programa secuencial ejecutando diferentes partes simultáneamente. Puede o no tener más de un hilo de control lógico.

Concurrencia es sobre tratar con muchas cosas a la vez.

Paralelismo es sobre hacer muchas cosas a la vez.

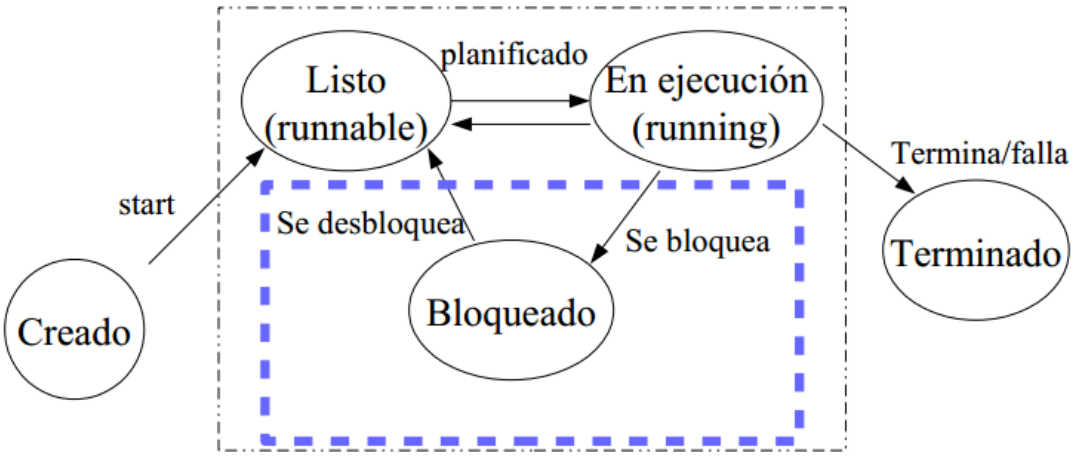
Concurrencia es sobre estructura, y paralelismo es sobre ejecución.

La concurrencia provee una forma de estructurar una solución para resolver un problema que puede ser paralelizable (pero no necesariamente lo es).

La comunicación se usa para coordinar las ejecuciones independientes.

Los hilos (**unidad de concurrencia**) permiten que varias corrientes de flujo de control de programas coexistan dentro de un proceso; estos comparten los recursos del proceso (memoria, etc.) **pero tienen su propia pila de llamadas, variables locales, contador, etc.**

Estados posibles de un Hilo



Los hilos que están es estado listo se mantienen en una colección, organizados según su prioridad.

Un hilo se bloquea concorde al contexto.

El cambio a listo se produce cuando termina su tiempo de CPU.

Cuando un hilo es desbloqueado vuelve a esa colección de listos.

Un problema propio de la Programación Concurrente es el indeterminismo (diferentes posibilidades en cuanto al orden o flujo de ejecución).

Problemas en lenguajes de alto nivel: Una instrucción de alto nivel se convierte en un conjunto de instrucciones máquina, estas instrucciones son las que se ejecutan concurrentemente, el indeterminismo implica que el resultado pueda ser incorrecto y es por esto que son necesarias ciertas restricciones en el flujo de ejecución.

En el paradigma orientado a objetos el modelo concurrente incluye características de los modelos de objetos: activo (hilos) y pasivo (recurso compartido), **se asocian las tareas a los métodos (el recurso compartido es el responsable de la sincronización) y los hilos que utilizan esas tareas a objetos (activos).**

En este contexto un objeto puede estar envuelto en múltiples actividades y una actividad puede incluir muchos objetos.

Cuando se produce la comunicación entre objetos se puede identificar un objeto cliente (el emisor del mensaje), y un objeto servidor (el receptor del mensaje), y en algunos casos un objeto agente o intermediario. En un escenario concurrente los objetos clientes son los activos y los servidores son pasivos. Esto es porque un servidor no hace nada a menos que se le indique, permanece a la espera de recibir un mensaje de parte de otro objeto.

Sección crítica: porción de código con variables compartidas y que debe ejecutarse en exclusión mutua.

Exclusión mutua: si un proceso está en su sección crítica, entonces ningún otro proceso puede ejecutar su sección crítica.

Programas Concurrentes Correctos - Tipos de propiedades:

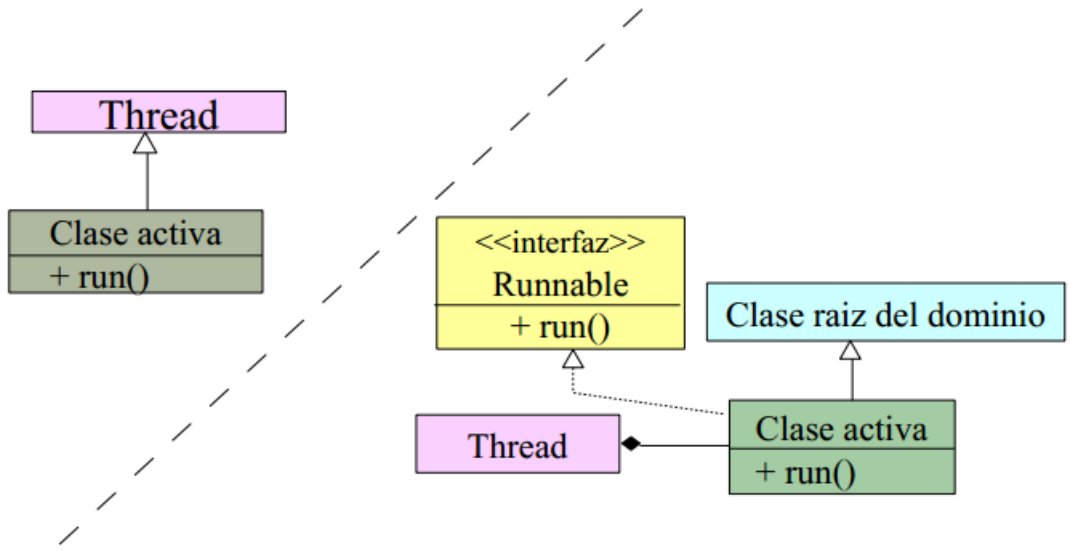
- Seguridad (**safety**): Son aquellas que aseguran que nada malo va a pasar durante la ejecución del programa. Si un comportamiento accede a la sección crítica debería dejarla en un estado consistente. Para asegurar consistencia se emplean las técnicas de **exclusión mutua**, garantizando la **atomicidad**.
- Viveza (**liveness**): Son aquellas que aseguran que algo bueno pasará eventualmente durante la ejecución del programa, es decir, **progresará**.

Posibles fallas de **progreso permanente:**

I. **Deadlock** (Interbloqueo pasivo), dependencias circulares entre locks.
II. **Livelock** (Interbloqueo activo), una acción que se intenta continuamente, continuamente falla.
III. **Starvation**, la máquina virtual falla siempre en asignar tiempo de CPU a un hilo.
IV. Señales pérdidas, un hilo permanece dormido porque empezó a esperar después de que una notificación se produjo.
V. Cierre de monitores anidados, un hilo en espera mantiene un lock que otro hilo que debe despertarlo está esperando.
VI. Agotamiento de recursos, un grupo de hilos mantienen todos los recursos.
VII. Falla distribuida, una máquina remota conectada por un socket se hace inaccesible o se rompe.

Los lenguajes de programación incorporan características que permiten expresar la concurrencia directamente. Las técnicas para producir actividades concurrentes pueden ser:

- Manuales: Utilizando llamadas al SO o con **bibliotecas de software**.
- Automáticas: Las detecta el SO en forma automática.



Un hilo se crea en Java instanciando la clase Thread. El código que ejecuta un thread está definido por el método run() que tiene todo objeto que sea instancia de la clase Thread. La ejecución del thread se inicia cuando sobre el objeto Thread se ejecuta el método start(). De forma natural, un thread termina cuando en run() se alcanza una sentencia return o el final del método.

Los procesos que se ejecutan en paralelo, son objetos que poseen una prioridad, la cual puede asignarse en un principio e ir modificándose a lo largo de la ejecución (En general, la prioridad la asigna el SO).

La prioridad de un proceso describe la importancia que tiene ese proceso por sobre los demás.

Constructores de la clase Thread:

- Thread()
- Thread(Runnable threadOb)
- Thread(Runnable threadOb, String threadName)
- Thread(String threadName)

Mecanismos de sincronización en JAVA

Bloques **synchronized**

- El lock es sobre un objeto en particular.
- El bloque synchronized lleva entre paréntesis la referencia a un objeto (this).
- Si el lock está libre, entonces el thread actual bloquea (lock) el objeto y entra a ejecutar el bloque.
- Cada vez que un thread intenta acceder a un bloque sincronizado le pregunta a ese objeto si no hay algún otro thread ejecutando algun bloque sincronizado con ese objeto, si es el caso entonces el thread que preguntó es suspendido y puesto en espera hasta que el lock se libere.
- 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.
- Si hay dos bloques synchronized que hacen referencia a distintos objetos, la ejecución de estos bloques **no** será mutuamente excluyente.

- 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 lo mismo:

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

```
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

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

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