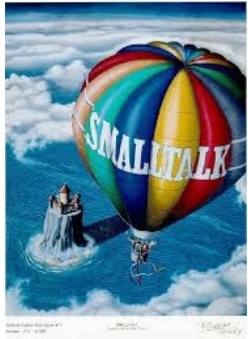




Departamento de Programación  
Facultad de Informática  
Universidad Nacional del Comahue



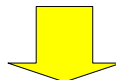
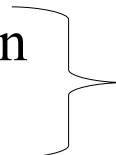
# Programación Concurrente



*Instrumentos de la  
concurrency*



# Trabajando con Hilos en Java

- Recordemos ...
  - Construcción de Hilos
    - Crear una subclase de Thread.
    - Definir una clase Runnable y proveerla al constructor de la clase Thread como argumento
  - Comunicación entre hilos
    - El mensaje `interrupt()` enviado a un hilo produce que el hilo sea interrumpido seteando su estado de interrupción a `true`.
      - si el hilo esta en *wait*, *join*, o *sleep*,  `InterruptedException`
    - El mensaje `join()` enviado a un hilo produce la suspensión del llamador hasta que el hilo target se completa.
      - join sin creación
      - join sin start ¿qué sucede?

# POO concurrente

correctitud

- Sistema orientado a objetos: vistas complementarias

- Centrado en objetos:

- colección de objetos interconectados

- Centrado en actividades

- Colección de actividades posiblemente concurrentes
    - Cada actividad puede envolver varios hilos
  - Un objeto puede estar envuelto en múltiples actividades
  - Una actividad puede abarcar múltiples objetos

seguridad

reusabilidad

viveza

performance

# POO concurrente

correctitud

- Sistema orientado a objetos: vistas complementarias

seguridad

- Centrado en objetos:

Comportamiento inesperado en ejecución,  
las cosas comienzan a funcionar mal

- Centrado en actividades

viveza

Las cosas dejan de ejecutarse – sin comportamiento

# POO Concurrente

- Sistema orientado a objetos: vistas complementarias

- Centrado en objetos

reusabilidad

calidad

La utilidad de objetos y clases  
a través de varios contextos

- Centrado en actividades

performance

Las actividades se ejecutan pronto y rápidamente

# POO concurrente - Seguridad

- Programación *secuencial*
  - Seguridad de tipos
  - Cuestiones de seguridad de tipos chequeadas por el compilador automáticamente ; no llega a ejecución
- Programación *concurrente*
  - Seguridad de tipos + dimensión temporal
  - No pueden ser chequeadas automáticamente; disciplina del programador

Todos los objetos en el sistema mantienen **estados consistentes**: todos sus atributos y los atributos de objetos de los que dependen Poseen valores legales y significativos

# POO concurrente - Seguridad

- Cada método público en cada clase debería llevar un objeto de un estado consistente a otro.
- Objetos seguros pueden ocasionalmente estar en estado de inconsistencia en medio de los métodos, pero no intentan iniciar nuevas acciones en estados inconsistentes

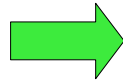
Cada objeto se diseña para ejecutar acciones solo cuando está habilitado lógicamente para ello

Todos los mecanismos son implementados de forma apropiada

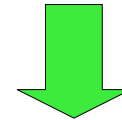
No hay errores por inconsistencia de objetos

# POO concurrente - Seguridad

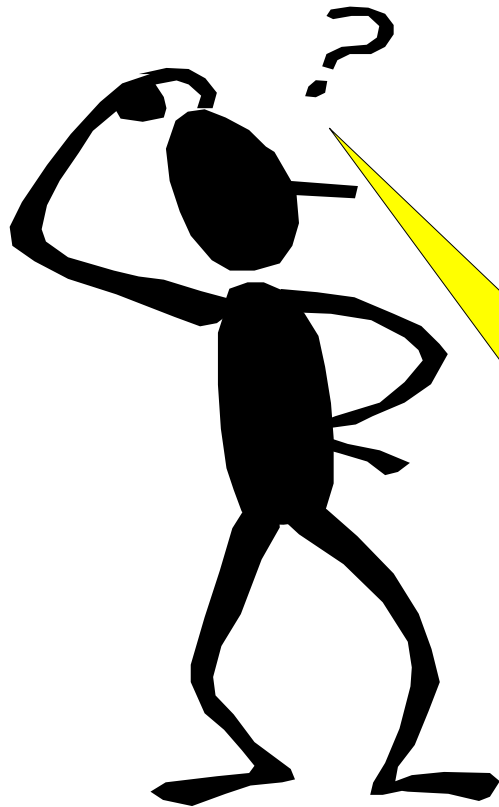
Asegurar consistencia



Emplear técnicas de exclusión



Garantizar la atomicidad de acciones públicas




Cada acción se ejecuta hasta que se completa sin interferencia de otras

Un balance de cuenta bancaria es incorrecto después de un intento de retirar dinero en medio de una transferencia automática



# POO concurrente - Seguridad

- Inconsistencias  **condiciones de carrera**
- Conflictos de lectura/escritura: *un hilo lee un valor de una variable mientras otro hilo escribe en ella. El valor visto por el hilo que lee es difícil de predecir – depende de que hilo ganó la “carrera” para acceder a la variable primero*
- Conflictos de escritura/escritura: *dos hilos tratan de escribir la misma variable. El valor visto en la próxima lectura es difícil de predecir*

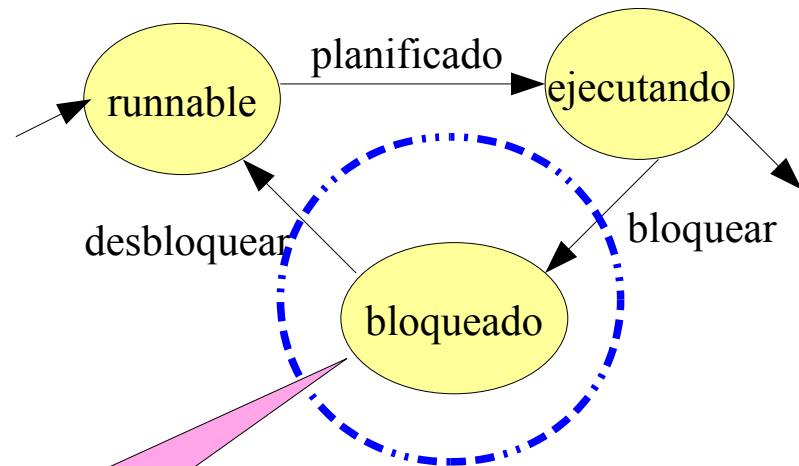
# POO concurrente - Viveza

- Sistemas *vivos*
  - cada actividad eventualmente progresa hacia su finalización.
  - cada método invocado eventualmente se ejecuta
  - Pero ... una actividad puede fallar en progresar por alguna razón

# POO concurrente - Viveza

Pero ... una actividad puede fallar en progresar por alguna razón ...

- Bloqueo - cerrojo
- Espera - wait
- Entrada
- Contención de CPU
- Falla



Representa  
distintas situaciones  
de bloqueo

# POO concurrente - Viveza

- Problema serio: falta de progreso permanente
  - Deadlock: dependencias circulares
  - Livelock: una acción falla continuamente
  - Starvation: un hilo espera por siempre, la maquina virtual falla siempre en asignarle tiempo de CPU
  - Falta de recursos: un grupo de hilos tienen todos los recursos, un hilo necesita recursos adicionales pero no puede obtenerlos
  - Otros (investigar)

# Exclusión

- En un sistema seguro cada objeto se protege de violaciones de integridad.
- Las técnicas de exclusión preservan los invariantes de los objetos y evitan efectos indeseados
- Las técnicas de programación logran la exclusión previniendo que múltiples hilos modifiquen o actúen de forma concurrente sobre la representación de los objetos.

Exclusión mutua: mantener estados consistentes de los objetos evitando interferencias NO deseadas entre actividades concurrentes

# Exclusión

- Estrategias básicas:

- Eliminar la necesidad de control de exclusión asegurando que los métodos nunca modifican la representación de un objeto



*inmutabilidad*

- Asegurar dinámicamente que solamente 1 hilo por vez puede acceder el estado del objeto usando cerrojos y construcciones relacionadas



*sincronización*

- Asegurando estructuralmente que solamente 1 hilo por vez, puede utilizar un objeto dado ocultándolo o restringiendo el acceso a él.



*confinamiento*

# Exclusion - inmutabilidad

- Si un objeto no puede cambiar su estado, nunca puede encontrar conflictos o inconsistencias cuando múltiples actividades intentan cambiar su estado.
- Los objetos inmutables mas simples no tienen campos internos, sus métodos son sin estado (*Stateless*)
- *Aplicaciones*
  - TDA: Integer, Date, Fraction, etc. - Instancias de estas clases nunca alteran los valores de sus atributos. Proveen métodos para crear objetos que representan nuevos valores
  - Contenedores de valores: es conveniente establecer un estado consistente una vez y mantenerlo por siempre.
  - Representaciones de estado compartidas: en general se trata de clases de utilidad.

# Exclusión - Sincronización

- Los bloqueos protegen contra conflictos de almacenamiento a bajo nivel y contra fallas de invariante a alto nivel
- Con 2 hilos ejecutándose de manera concurrente, sin bloqueos ni sincronización, típicamente la mayoría de las trazas no muestran la violación de seguridad. Pueden pasar muchos testeos correctos, pero... eventualmente en algún momento se dará la falla.



- Sincronizar los métodos evita los conflictos



# Exclusión - Sincronización

- Cada instancia de Object y sus subclases posee bandera de bloqueo (lock)
- Los tipos primitivos (no objetos) solo pueden bloquearse a través de los objetos que los encierran
- No pueden sincronizarse variables individuales
- Las variables pueden declararse como “volátiles”
- Los objetos arreglos cuyos elementos son tipos primitivos pueden bloquearse, pero sus elementos NO

# Exclusión - Sincronización

- Para tener en cuenta
  - El modificador “synchronized”:
    - no se hereda automáticamente cuando se redefine un método,
    - no se puede utilizar en una interfaz,
    - no se puede utilizar en la signatura de un constructor,
    - aplicado a un bloque SI se puede utilizar dentro de un constructor.

# Exclusión – Sincronización - Java

- Java utiliza **Synchronized** para sincronizar el trabajo sobre secciones críticas.
- 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 ejecutando algún bloque sincronizado con ese objeto.
- Si **otro thread** ha realizado el **bloqueo (lock)**, 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 bloquea (lock)** el objeto y entra a ejecutar el bloque.
- 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.
- El bloqueo es sobre un objeto en particular.
- Si hay dos bloques synchronized que hacen **referencia a distintos objetos**, la ejecución de estos bloques **no será mutuamente excluyente**.

# Exclusión – Sincronización - Java

- El mecanismo de sincronización funciona si TODOS los accesos a los *datos delicados* ocurren dentro de bloques sincronizados
- Los datos delicados protegidos por bloques sincronizados deben ser privados
- ¿ Sincronizar el método o sincronizar un bloque ?
  - Ventaja
  - Desventaja