El lenguaje Java en la asignatura de Programación Concurrente

Juan Pavón Mestras

Departamento de Sistemas Informáticos y Programación
Escuela Superior de Informática
Universidad Complutense de Madrid

RESUMEN

En menos de dos años desde su aparición (en mayo de 1995), el lenguaje Java ha tenido una amplia difusión a través de Internet, y está siendo adoptado en los planes de estudios de Informática en muchas universidades de todo el mundo. Varias de las ventajas que ofrece con respecto a otros lenguajes tradicionalmente utilizados para la enseñanza de la programación son: su tamaño reducido, lo que hace que sea fácil de aprender y empezar a utilizar, el ser orientado a objetos, la disponibilidad de herramientas para desarrollar programas Java en distintos entornos, y las facilidades de gestión de gráficos y su integración en la Web. Por lo que se refiere a la programación concurrente, ofrece un modelo sencillo, como parte del propio lenguaje, para definir y gestionar múltiples flujos de ejecución (threads), y primitivas de sincronización (similares al paradigma de monitor) y de control de los flujos de ejecución (wait, notify, interrupt, suspend, resume). Esta ponencia muestra cómo ha sido aplicado a la asignatura de Programación Concurrente en la Escuela Superior de Informática de la Universidad Complutense de Madrid.

1. INTRODUCCIÓN

Con la proliferación actual de aplicaciones distribuidas, tanto en el ámbito de las redes de área local como en el de las redes de área extensa (por ejemplo, Internet), la concepción tradicional de programa como entidad aislada respecto a otros programas da paso a la de computación de objetos distribuidos, que interactúan entre sí. Los programas dejan de concebirse para ser ejecutados en una secuencia *arranque -> entrada -> proceso -> salida -> fin*, sin interferencias por parte de otros procesos. Ahora un programa tiene que desarrollarse teniendo en cuenta que puede ser implantado en

distintos entornos, donde cada proceso puede ser activado e invocado por otros procesos cuyo comportamiento no está siempre especificado, y que para realizar sus funciones tiene que competir con otros procesos dinámicamente. Además, el software con el que interactúa puede cambiar durante la existencia del programa, o a partir del propio programa se pueden realizar extensiones para dar nueva funcionalidad y satisfacer nuevos requisitos. Dos técnicas que ayudan al programador para enfrentarse con este tipo de aplicaciones son la orientación a objetos y la concurrencia.

La programación orientada a objetos se ha impuesto recientemente ya que soporta de forma inherente el modelado de entidades (objetos) que interaccionan entre sí. Además, satisface un conjunto de propiedades que contribuyen a mejorar la calidad del software: modularidad, encapsulación, herencia, reusabilidad, seguridad de tipos, definición de interfaces. La idea básica es estructurar el software en objetos en vez de en procedimientos que operan sobre un conjunto de estructuras de datos. Desde el punto de vista del ciclo de vida del software, una consideración importante es que la orientación a objetos permite aplicar los mismos conceptos en distintas actividades, del diseño a la programación, simplemente cambiando el nivel de abstracción. En la enseñanza esto se refleja en la idoneidad del uso del paradigma de orientación a objetos para las asignaturas de Ingeniería de Software y de Programación.

La concurrencia es un elemento inherente a los sistemas distribuidos, donde distintos procesos pueden ejecutarse en maquinas que se comunican a través de una o varias redes. También es importante a nivel de un proceso para modelar mas fácilmente actividades que se pueden realizar en paralelo o para mejorar la eficiencia de la ejecución (por ejemplo, mientras se espera la realización de una operación de entrada/salida o la ocurrencia de un determinado evento o condición pueden realizarse otras acciones).

Teniendo en cuenta la complementariedad de la programación concurrente y la orientación a objetos, parece razonable para la asignatura de Programación Concurrente buscar un lenguaje que los integre. Así se avanza con respecto a la enseñanza tradicional de la asignatura, permitiendo abordar una visión del problema de la programación concurrente dentro del diseño de software para sistemas complejos, como se demanda actualmente.

Como lenguaje que cumple este requisito se ha elegido Java. A pesar de su breve existencia (la versión alfa fue anunciada oficialmente por Sun Microsystems en mayo de 1995), en muy poco tiempo ha sido ampliamente aceptado tanto en el mundo académico como en el industrial, debido, principalmente, a su idoneidad para satisfacer las necesidades de los programadores en el entorno de la Web: gran facilidad de distribución y portabilidad (Java es soportado por una maquina virtual), simplicidad de instalación y ejecución (los *applets* Java se cargan directamente desde un lugar local o remoto a través de Internet en cuanto son necesarios), soporte de comunicaciones y multimedia, y extensibilidad de las aplicaciones.

Un aspecto importante para soportar aplicaciones en un entorno distribuido y para el tratamiento de interfaces de usuario avanzados es el disponer de un modelo de concurrencia de flujos de ejecución. En general, los paquetes de *multi-threading* no integrados en los lenguajes de programación suelen ser bastante difíciles de utilizar. Java, sin embargo, incorpora de una forma natural el concepto de flujo de ejecución (*thread*). Además, permite determinar de forma selectiva qué partes del código están en exclusión mutua, por medio del constructor *synchronized*, y ofrece los métodos *wait*, *notify* y *notifyAll* para coordinar las actividades entre *threads*.

Por otra parte, es fácil encontrar productos, algunos de ellos de libre distribución, para diseñar, compilar y ejecutar código Java. Ello facilita a los alumnos la realización de las prácticas de la asignatura, ya que pueden desarrollarlas tanto en los laboratorios del centro, como en sus computadores personales (Windows o Mac). Otra facilidad desde el punto de vista logístico, es la posibilidad de que se entreguen por medio de correo electrónico o dando simplemente una dirección URL, mecanismos que dotan de mayor flexibilidad la comunicación entre profesor y alumnos.

La experiencia del primer año en que se ha utilizado en la Escuela Superior de Informática de la Universidad Complutense de Madrid ha sido bastante fructífera, como soporte en clase para explicar el temario de la asignatura de Programación Concurrente, y para la realización de una gran diversidad de prácticas. Los alumnos han estado bastante motivados por la elección del lenguaje Java, pues es posible realizar en poco tiempo programas bastante completos en cuanto a funcionalidad y presentación, y también por el interés del conocimiento de este lenguaje de cara a su curriculum profesional. Para el profesor de la asignatura ha sido el medio de expresar la mayoría de los problemas y conceptos desarrollados en la asignatura.

A continuación se introducen brevemente las construcciones definidas en el lenguaje Java para soportar la concurrencia y sincronización de flujos de ejecución (*threads*). Estas se ilustran con dos ejemplos sencillos para que el lector se haga rápidamente una idea de la apariencia de los programas Java. Posteriormente, se describen los objetivos de la asignatura de Programación Concurrente y los pasos que se siguen para instruir a los alumnos en los contenidos de la materia. A partir de aquí se puede discutir dónde es necesaria la elección del lenguaje de programación para la asignatura. La aplicación del lenguaje Java se expone en la sección siguiente, tanto en la parte teórica de la asignatura como, especialmente, en la practica. Finalmente, en las conclusiones se compara con otros lenguajes en base a criterios de la sintaxis y semántica del lenguaje, y también logísticos, como puede ser su disponibilidad y la gran cantidad de documentación y herramientas de soporte existentes.

2. Concurrencia en el lenguaje Java

El lenguaje Java [1] está estructurado en torno al concepto de *clase*. Un programa Java se define como un conjunto de clases. Una clase define un conjunto de atributos (constantes y variables) y funciones (métodos). En ejecución se crean y destruyen dinámicamente instancias de las clases, *objetos*, que interaccionan entre sí invocando métodos. Una clase puede estar definida a partir de otra, *superclase*, de la que hereda las propiedades definidas, extendiendo atributos y funciones, o redefiniendo el comportamiento de algunas funciones. Además, es posible definir clases abstractas, que definen métodos abstractos, esto es, cuyo comportamiento no está especificado (se deja para las subclases).

Para soportar la concurrencia, Java define la clase *Thread*, cuyas instancias (objetos) representan cada una un flujo secuencial de ejecución, que son ejecutados por la máquina virtual Java de forma concurrente (entrelazados). La actividad de un *thread* está definida en un método *run*. Además, la clase *Thread* ofrece varios métodos de control, entre ellos:

- *start* para iniciar el flujo de ejecución, llamando al método *run*.
- *stop* para terminar la actividad de un objeto *Thread*.
- *suspend* para pararlo temporalmente, hasta que se reanude su actividad con *resume*.
- *sleep* para suspender su actividad durante un periodo de tiempo especificado en milisegundos.
- *join* para suspender al llamante hasta que el objeto *Thread* llamado termine.
- *interrupt* para abortar un *sleep*, *wait* o *join*, produciendo una excepción.

Una forma de crear un flujo de ejecución es definiendo una clase que herede de la clase *Thread*. En el siguiente programa el método *main*[1] crea diez *threads*, pasándole a cada uno un número en su

[1] El método *main*, al ser declarado como *static*, es un método de la clase, no de cada instancia de la clase u objeto. El intérprete Java intenta ejecutar el método *main* cuando se le da una clase. Los métodos *run*, *start*, o el constructor, son definidos para cada objeto de la clase, al igual que las variables como en este caso *numero_*, de las que hay una instancia para cada objeto.

constructor. Inmediatamente tras su construcción, arranca cada nuevo objeto de la clase *EjemploThread* con el método *start*. El método *run* indica qué hace el flujo de ejecución. En este caso es un bucle del que se sale por una interrupción, en el cual se imprime un número, el que se le pasó como argumento al constructor, y se espera un tiempo aleatorio entre 0 y 1000 milisegundos.

```
import java.lang.Math;

class EjemploThread extends Thread {
  int numero_;

  EjemploThread (int n) {  // constructor
        numero_ = n;
  }
        public void run() {
        try {
                    while (true) {
                          System.out.println(numero_);
                          sleep((long)(1000*Math.random()));
                    }
        } catch (InterruptedException e) {
              return; // acaba este thread
        }
  }

  public static void main (String args[]) {
        int i;

        for (i=0; i<10; i++)
              new EjemploThread(i).start();
  }
}
```

Obsérvese también la construcción para el uso de las excepciones, con *try* y *catch*. La ventaja del mecanismo de excepciones es que permite separar la descripción del flujo de ejecución normal de los tratamientos que se requieren ante circunstancias alternativas, por ejemplo de fallos o de eventos no comunes, que se tratan dentro del bloque de instrucciones de la parte *catch*.

La sincronización entre *threads* se puede definir a nivel de método, usando la palabra *synchronized* en la cabecera de la definición del método, o para un bloque de instrucciones por medio de la construcción *synchronized (objeto) { código(); }*.

Además, existe el método *wait*, que suspende la ejecución de un *thread* hasta que sea despertado por otro que invoque los métodos *notify* y *notifyAll*.

Como ejemplo de la utilización de estos mecanismos, véase una forma de definir un semáforo:

```
public class Semaforo {
  private int cuenta_ = 0;

  public Semaforo(int cuentaInicial) {  // constructor
        cuenta_ = cuentaInicial;
  }

  public synchronized void P() {
        while (cuenta_ <= 0)
              try { wait(); }
              catch (InterruptedException ex) {}
        --cuenta_;
  }
```

```
    public synchronized void V() {
          ++cuenta_;
          notify();
    }
}
```

La definición de la clase Semaforo consta de un miembro privado (*cuenta_*) que indica cuantos procesos pueden pasar el semáforo. Si *cuenta_* es inicializado en el constructor con el valor 1, entonces se tiene un semáforo binario. Los métodos P y V son declarados como *synchronized*. Esto quiere decir que mientras haya un *thread* ejecutando uno de estos métodos, ningún otro *thread* puede hacerlo. El bloqueo que pone un *thread* al entrar a ejecutar uno de estos métodos sólo se libera al acabar el método o al quedarse suspendido, como ocurre si se ejecuta la sentencia *wait* en el método *P*. Esto es necesario para dejar la posibilidad de que otro *thread* ejecute el método *V* y al realizar *notify* permita despertar a un *thread* que hubiera hecho *wait*.

Para planificar la utilización del procesador, es posible definir distintos niveles de prioridad para los *threads*. La prioridad de un *thread* es inicialmente la misma que la del que lo creó, pero puede ser modificada con la operación *Thread.setPriority*, entre los valores *Thread.MIN_PRIORITY* y *Thread.MAX_PRIORITY*. Existe el método *Thread.yield* para dejar el control y dar paso a otros threads de la misma prioridad.

3. Enseñanza de la asignatura Programación Concurrente

El objetivo de la asignatura es que los alumnos adquieran la capacidad de diseñar programas concurrentes fiables, considerando también otras propiedades importantes que definen la calidad del software, como son la estructuración, la robustez, la fiabilidad, la reutilización, y la extensibilidad del software.

Se consideran varios aspectos en la aplicación de la programación concurrente, como por ejemplo: diseño de sistemas reactivos (esto es, aquellos que pueden realizar varias actividades a la vez, cada una como respuesta a una entrada o petición), control de actividades (posible con las operaciones de suspensión, continuación o terminación en los flujos de ejecución), modelado de sistemas reales (donde cada entidad puede aparecer comportándose de forma autónoma a las demás), interacciones asíncronas (cuando un objeto envía un mensaje a otro y sigue ejecutándose sin tener que esperar a que el otro procese el mensaje), paralelismo (aunque no se disponga de máquinas con varios procesadores, el procesamiento de varios flujos de control entrelazados permite evitar retrasos producidos por algunas actividades que no tienen que completarse para poder iniciar otras actividades).

La corrección de los programas concurrentes se define como consecuencia del análisis de una serie de propiedades de seguridad y viveza, más una reflexión sobre el coste que determinados mecanismos de sincronización, la construcción de *threads*, y el cambio de contexto, pueden tener en el diseño de un programa.

El método docente empleado consta de varios pasos:

1. La presentación genérica del concepto (por ejemplo, monitor, guarda, paso de mensajes síncrono y asíncrono, excepción, transición) como solución a un problema determinado, o contrastándolo con otros conceptos.
2. Descripción del concepto mediante construcciones existentes en un lenguaje específico o patrones de programación desarrollados en el lenguaje.

3. Aplicación a ejemplos concretos, reflexionando sobre el análisis de la corrección de la solución empleada, y posibles variaciones. Aquí es interesante considerar como la solución presentada podría ser extendida o reutilizada en otros contextos al inicialmente planteado.
4. Proposición de prácticas a los alumnos, para que elaboren por sí mismos los conceptos desarrollados en clase.

La elección de un lenguaje concreto es importante en los pasos 2, 3 y 4. Es aquí donde se ha considerado el lenguaje Java. La utilización de un sólo lenguaje es importante como medio de forzar al alumno a adecuarse a un modelo y un conjunto de facilidades específicos, para explorar cómo desarrollar las herramientas de que dispone en la resolución de distintos problemas. Ello le permite sensibilizarse sobre qué beneficios tienen ciertas construcciones del lenguaje empleado, y qué limitaciones plantea. En este último sentido, una de las prácticas a desarrollar consiste en realizar clases que definan construcciones no presentes en el lenguaje, y resolver algunos problemas con ellas.

Por otra parte, y para facilitar la tarea de traducir entre distintas notaciones, se incluye un tema dedicado a revisar distintos lenguajes de programación concurrente, como son Ada, Smalltalk, Occam, programación funcional paralela en Haskell, Modula-2, Pascal-FC, y librerías para soportar la programación concurrente en otros entornos, como Unix, por ejemplo para los lenguajes C y C++.

4. Uso del lenguaje Java en la asignatura de Programación Concurrente

El lenguaje Java no ofrece todos los mecanismos de concurrencia y sincronización de procesos existentes, y, en este respecto, existen alternativas como Pascal-FC, que pueden ser más completos. Es por tanto que no todo el desarrollo de la asignatura se realiza utilizando el lenguaje Java como soporte. Sin embargo, es importante que el alumno sepa desarrollar completamente un conjunto limitado de conceptos para resolver cualquier tipo de problemas, sabiendo establecer las restricciones de los elementos de que dispone. En este sentido, es importante profundizar en la utilización de un sólo lenguaje en la asignatura, y desarrollar patrones, aparte de técnicas para especificar y razonar rigurosamente sobre la corrección del diseño de los programas concurrentes, con el propósito de:

- Incitar a la reutilización de prácticas de programación que han demostrado tener éxito.
- Como medio para enseñar técnicas útiles y razonar sobre lo que hacen y por qué.

Un patrón es una solución recurrente a un problema común, aplicable dentro de un contexto. Los patrones surgen a partir de la experiencia en la realización de programas, y su objetivo es facilitar la labor del programador mediante la identificación de problemas y la provisión de soluciones a éstos, indicando los costes asociados. La ventaja de los patrones es evitar que el programador tenga que *reinventar la rueda*. En este aspecto, el libro escrito por Doug Lea, *Concurrent Programming in Java* [2], resulta muy útil para el desarrollo de la asignatura, ya que está organizado en torno a la idea de patrones de diseño para programación concurrente orientada a objetos. En gran parte, este libro ha facilitado la elección del lenguaje Java como soporte de la asignatura ya que las referencias que se encuentran en la literatura de programación concurrente no están adaptadas a la problemática actual del software orientado a objetos.

Por otra parte, como herramienta fundamental para el desarrollo de la asignatura se consideran las prácticas. A programar se aprende programando. En este sentido, Java tiene ventajas obvias sobre otros lenguajes, especialmente porque al tratarse de un lenguaje orientado a objetos es posible introducir nuevas nociones progresivamente mediante la definición de nuevas clases, y permite estructurar el software en componentes (definiciones de clases) relativamente cortos, que pueden probarse individualmente. Además, Java proporciona paquetes para integrar de forma sencilla los elementos de entrada/salida de los computadores actuales, como son gráficos, ventanas, y el control del ratón, lo que motiva, e incluso divierte, a los alumnos en la realización de las prácticas. Otro

aspecto interesante es el soporte de *javadoc* para generar documentación de los programas en formato HTML, ayudando a mejorar la calidad del software producido.

El curso de Programación Concurrente consta de 6 créditos de teoría y 3 de prácticas. Esto quiere decir, 30 horas de prácticas (10 horas por crédito). Teniendo esta disponibilidad en cuenta, el primer año se han planteado las siguientes prácticas:

1. Desarrollo de un programa que simula el control de accesos a un local. El propósito de esta primera práctica es familiarizar a los alumnos con el lenguaje, la orientación a objetos, los mecanismos de excepciones, de creación de *threads*, y las primitivas de control y sincronización de *threads*.
2. Realización de construcciones de sincronización no existentes en el lenguaje: semáforos, regiones críticas, regiones críticas condicionales, monitores con variables condicionales y variables condicionales con prioridad.
3. Realización de algunos algoritmos clásicos de concurrencia, tales como el problema de los filósofos, el problema de la barbería, etc., con variantes a partir del planteamiento inicial con el propósito de estudiar el impacto de extender la funcionalidad en la solución desarrollada.
4. Problema de planificación de recursos. Se trata de un problema común tratado en programación concurrente. Es importante ver cómo se pueden definir distintas estrategias.
5. Proyecto a definir por los alumnos. Aquí se promociona la realización en grupo, con el propósito de presentar un proyecto ambicioso y estudiar la estructuración del mismo en distintos componentes que interactúan entre sí.

5. CONCLUSIONES

En el apartado 2 se muestra como Java proporciona de una forma sencilla la creación y definición de *threads*, y los mecanismos para controlarlos y sincronizarlos. La simplicidad de la sintaxis y la semántica del lenguaje ha sido uno de los principales criterios para elegir Java. Es importante que el alumno pueda centrarse desde el principio en los aspectos que se quieren estudiar, en nuestro caso, concurrencia y sincronización de *threads*. Un lenguaje como Ada, sin embargo, no cumpliría este requisito. Smalltalk podría considerarse a este respecto, pero Java tiene la ventaja de parecerse más a otros lenguajes convencionales (especialmente C++), con los que los alumnos ya están familiarizados, y por tanto el tiempo de aprendizaje es muy corto (dos sesiones de dos horas y la primera práctica han sido suficientes).

Por lo que se refiere al mecanismo básico de sincronización de *threads*, el uso de *synchronized* en Java ofrece ventajas con respecto al mecanismo general de monitor, tal como está definido, por ejemplo, en el lenguaje CC-Modula, especialmente porque *synchronized* permite definir la sincronización en varios niveles de detalle. Esto es interesante en cuanto obliga al alumno a reflexionar sobre dónde es verdaderamente necesaria la sincronización para garantizar la exclusión mutua, y cómo mejorar la eficiencia de ejecución, esto es, dotar de mayor viveza, al sistema resultante.

También se ha considerado relevante el hecho de tratarse de un lenguaje orientado a objetos, fuertemente tipado, por las razones que se han aducido antes sobre el interés de acostumbrar a los alumnos a prácticas de construcción de software de calidad, bien estructurado, robusto, fiable, extensible y reusable.

Hay otros criterios, de carácter externo, que también se han tenido en cuenta. Especialmente la popularidad mundial alcanzada por el lenguaje Java actualmente. Su aplicación en la Web y su aceptación en la industria hacen deseable que los alumnos lo conozcan. En concreto, el dominio de

los mecanismos de concurrencia del lenguaje les aporta un valor añadido respecto a cualquier otro programador Java.

Es importante que el lenguaje esté claramente especificado y que su semántica y sintaxis sea estable y normalizada. Gracias a ello, hay muchas herramientas para Java disponibles en distintos entornos de trabajo en computadores personales y estaciones de trabajo, que lo hacen fácilmente accesible. La disponibilidad de entornos de desarrollo de libre distribución es, en este sentido, un factor a considerar, especialmente de cara a los alumnos.

Por otra parte, sería interesante considerar la utilización del lenguaje en otras asignaturas del plan de estudios. En concreto, ha empezado a utilizarse para el desarrollo de las prácticas de la asignatura de Ingeniería del Software, que consisten en realizar aplicaciones para servidores Web. Debería discutirse su utilización como primer lenguaje, sustituyendo al Pascal tradicional. Como se comentaba al principio, los sistemas actuales son concebidos como un conjunto de componentes de software, por lo cual es más crítico dominar las técnicas de encapsulación, estructuración y coordinación de componentes, que el realizar una buena estructura de datos (éstas ya vienen dadas en librerías). Desde esta perspectiva, Java puede ser un buen vehículo para introducir estos conceptos desde los cursos de iniciación a la programación.

Referencias

[1] K. Arnold, J. Gosling, *El lenguaje de programación Java™*. Addison-Wesley/Domo, 1997

[2] D. Lea, Concurrent Programming in Java™. Design Principles and Patterns. Addison-Wesley, 1997.