# Práctica 1: Intérprete de instrucciones

**Fecha de entrega:** 17 de Noviembre de 2013, 16:00

**Objetivo:** Iniciación a la orientación a objetos y a Java; uso de arrays y enumerados; manipulación de cadenas con la clase `String`; entrada y salida por consola.

## 1. Introducción

Una máquina virtual es un software que simula a un ordenador y que puede ejecutar programas como si fuese un ordenador real. Puedes encontrar más información en `http://es.wikipedia.org/wiki/Máquina_virtual`. El ordenador simulado puede ejecutar programas como si se tratase de un ordenador real. Este tipo de aplicaciones tiene diferentes usos aunque el más extendido es la "prueba" de sistemas operativos sin tener que cambiar el que utilizan habitualmente. La máquina virtual que pretendemos implementar a lo largo del curso es una máquina de pila, cuyos principales componentes son una *pila de operandos* y una *memoria*. La pila de operandos almacena los datos que se generan durante la ejecución mientras que en la memoria podemos almacenar algunos de esos datos. Es decir, la memoria sería similar a una tabla de variables de un programa.

A lo largo del curso realizaremos distintas prácticas encaminadas a finalizar con una aplicación completa de máquina virtual (a la que llamaremos TPMV), capaz de ejecutar programas creados para esta máquina, y cuya sintaxis iremos definiendo a lo largo del curso. La aplicación final permitirá mostrar el *estado* de la máquina durante la ejecución así como modificar parte del mismo utilizando tanto un interfaz de consola como de ventana. El objetivo de esta primera práctica es construir los primeros bloques lógicos necesarios.

## 2. Descripción de la práctica

En esta primera práctica vamos a crear una versión inicial muy sencilla de nuestra máquina virtual. En concreto la aplicación será capaz de ejecutar unas pocas instrucciones *bajo demanda*.

Al arrancar la aplicación, se iniciarán todas las estructuras de la TPMV y se presentará un *prompt* en donde el usuario irá tecleando las distintas instrucciones que se quieren ejecutar. Tras cada ejecución, la aplicación mostrará el estado en el que ha quedado la máquina después de esa instrucción y volverá a pedir al usuario la siguiente instrucción a ejecutar.

Es posible que la ejecución de alguna instrucción falle (lo que en una máquina real provocaría posiblemente una *interrupción hardware*). En ese caso la aplicación simplemente mostrará un mensaje indicando que no ha sido posible la ejecución y pedirá una nueva instrucción.

La aplicación terminará cuando se ejecute la instrucción `HALT` que, como se describe en la sección 3, provoca la parada de la máquina.

## 3. Descripción de TPMV en la práctica 1

TPMV está compuesta de dos partes muy simples:

- Una *memoria* capaz de almacenar datos. La unidad mínima de memoria es el entero, es decir en cada celda almacena un entero completo (y no un *byte* como suele ocurrir en las máquinas reales). La capacidad de la memoria es *ilimitada*, es decir se podrá escribir en cualquier dirección ($\geq 0$), hasta que la memoria de la máquina física subyacente "aguante".
- Una *pila de operandos* en la que se realizan las operaciones. Gran parte de las distintas instrucciones de la máquina virtual trabajan sobre la pila de operandos, cogiendo de ella valores y/o dejando en ella resultados.

En esta primera práctica la TPMV tiene un conjunto reducido de instrucciones que no la permitirán ejecutar programas sofisticados. La mayoría de ellas *no* tienen parámetros (pues trabajan directamente con la pila de operandos). Sólo unas pocas de ellas tienen un parámetro de tipo entero. El conjunto de instrucciones admitidas es:

- `PUSH n`: apila en la pila de operandos el entero `n` de la instrucción.
- `POP`: extrae un elemento de la cima de la pila.
- `DUP`: duplica el valor de la cima de operandos.
- `FLIP`: invierte los dos valores superiores de la pila de operandos. Es decir, la cima y el valor inmediatamente debajo de ésta ("subcima") se intercambian, de forma que la subcima pasa a ser la cima y viceversa.
- `LOAD pos`: lee de memoria el valor almacenado en `pos` y lo apila en la pila de operandos.
- `STORE pos`: escribe en la posición `pos` de memoria el contenido de la cima de la pila de operandos, que se elimina.
- `ADD`, `SUB`, `MUL`, `DIV`: operaciones aritméticas de suma, resta, multiplicación y división. Todas ellas utilizan como operadores la subcima y cima de la pila de operandos que son sustituidos por el resultado de la operación. El primer operador es *la subcima* (esto es importante para las operaciones no conmutativas).

- `OUT`: escribe el caracter almacenado en la cima de la pila. Dado que en la pila de operandos lo que se manejan son enteros, deberá primero convertirlo al tipo carácter. La operación tendrá un comportamiento indeterminado si el entero no es un carácter válido.

- `HALT`: para la máquina.

# 4. Ejemplo de ejecución

A continuación mostramos un ejemplo de ejecución de nuestra simulación. Observa que las instrucciones *no* son sensibles a mayúsculas ni minúsculas (puedes escribirlas indistintamente). Eso sí, cuando la aplicación tiene que mostrar una instrucción, lo hará utilizando el mnemónico en mayúsculas.

En el momento de mostrar el estado de la máquina, la aplicación muestra la pila y la memoria tras la ejecución de cada instrucción. Si no tienen elementos, se indica `<vacia>`. En el caso de la memoria eso viene a significar que alguna celda de memoria ha sido escrita en algún momento. La pila se mostrará entera en una única línea con los números separados por espacios, comenzando desde la base de la pila.

A continuación aparece un ejemplo de ejecución. El texto en negrita representa lo que el usuario de la aplicación introduce por teclado.

```
Instruccion a ejecutar: push 3
Comienza la ejecución de PUSH 3
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: 3
Instruccion a ejecutar: STORE 0
Comienza la ejecución de STORE 0
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: <vacia>
Instruccion a ejecutar: Load 0
Comienza la ejecución de LOAD 0
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: 3
Instruccion a ejecutar: dup
Comienza la ejecución de DUP
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: 3 3
Instruccion a ejecutar: add
Comienza la ejecución de ADD
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: 6
Instruccion a ejecutar: push 3
Comienza la ejecución de PUSH 3
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: 6 3
Instruccion a ejecutar: flip
Comienza la ejecución de FLIP
```

```
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: 3 6
Instruccion a ejecutar: sub
Comienza la ejecución de SUB
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: -3
Instruccion a ejecutar: halt
Comienza la ejecución de HALT
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [0]:3
Pila de operandos: -3
```

A continuación aparece un pequeño ejemplo de ejecución donde alguna de las instrucciones fallan. Como se ve, el mensaje de error es único para todas las instrucciones.

```
Instruccion a ejecutar: dup
Comienza la ejecución de DUP
Error en la ejecución de la instruccion
Instruccion a ejecutar: push 2
Comienza la ejecución de PUSH 2
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: 2
Instruccion a ejecutar: add
Comienza la ejecución de ADD
Error en la ejecución de la instruccion
Instruccion a ejecutar: push 0
Comienza la ejecución de PUSH 0
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: 2 0
Instruccion a ejecutar: div
Comienza la ejecución de DIV
Error en la ejecución de la instruccion
Instruccion a ejecutar: halt
Comienza la ejecución de HALT
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: <vacia>
```

A continuación presentamos un último ejemplo de ejecución donde se muestra el uso de la memoria de la máquina:

```
Instruccion a ejecutar: push 7
Comienza la ejecución de PUSH 7
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: 7
Instruccion a ejecutar: push 98
Comienza la ejecución de PUSH 98
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: 7 98
Instruccion a ejecutar: out
```

```
Comienza la ejecución de OUT
b
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: 7
Instruccion a ejecutar: add
Comienza la ejecución de ADD
Error en la ejecución de la instruccion
Instruccion a ejecutar: flip
Comienza la ejecución de FLIP
Error en la ejecución de la instruccion
Instruccion a ejecutar: push 7
Comienza la ejecución de PUSH 7
El estado de la maquina tras ejecutar la instrucción es:
Memoria: <vacia>
Pila de operandos: 7 7
Instruccion a ejecutar: store 20
Comienza la ejecución de STORE 20
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [20]:7
Pila de operandos: 7
Instruccion a ejecutar: dup
Comienza la ejecución de DUP
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [20]:7
Pila de operandos: 7 7
Instruccion a ejecutar: store 30
Comienza la ejecución de STORE 30
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [20]:7 [30]:7
Pila de operandos: 7
Instruccion a ejecutar: load 20
Comienza la ejecución de LOAD 20
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [20]:7 [30]:7
Pila de operandos: 7 7
Instruccion a ejecutar: load 20
Comienza la ejecución de LOAD 20
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [20]:7 [30]:7
Pila de operandos: 7 7 7
Instruccion a ejecutar: div
Comienza la ejecución de DIV
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [20]:7 [30]:7
Pila de operandos: 7 1
Instruccion a ejecutar: store 20
Comienza la ejecución de STORE 20
El estado de la maquina tras ejecutar la instrucción es:
Memoria: [20]:1 [30]:7
Pila de operandos: 7
Instruccion a ejecutar: storeeeee 20
Error: Instrucción incorrecta
Instruccion a ejecutar: halt
Comienza la ejecución de HALT
El estado de la maquina tras ejecutar la instrucción es:
```

```
Memoria: [20]:1 [30]:7
Pila de operandos: 7
```

## 5. Implementación

Para lanzar la aplicación se ejecutará la clase `tp.pr1.mv.Main`, por lo que se aconseja que todas las clases desarrolladas estén en el paquete `tp.pr1.mv` (o subpaquetes de él).

Para implementar la práctica necesitarás, al menos, las siguientes clases:

- Memory: Representa la memoria de la maquina. Recuerda que la memoria de la máquina es ilimitada, y se puede escribir en cualquier dirección mayor o igual que 0. Esto significa que cuando se almacena o se lee en cualquier dirección no se generará ningún error de ejecución a menos que sea provocado por la memoria de la máquina física externa, en cuyo caso no haremos nada. Para poder visualizar la memoria por pantalla necesitamos saber qué posiciones de memoria han sido utilizadas. Sólo consideraremos que una posición de memoria ha sido utilizada si se ha escrito en ella, es decir se ha ejecutado alguna instrucción `STORE` sobre esa posición.
- OperandStack: Implementa la pila de operandos donde se van apilando elementos de tipo entero. Las instrucciones que modifican la pila de operandos son `PUSH n`, `POP`, `FLIP`, `LOAD` y las operaciones aritméticas.
- Instruction: Implementa las distintas instrucciones que puede manejar nuestra máquina virtual. Para representar las distintas instrucciones utiliza un tipo enumerado.
- CPU: Es la unidad de procesamiento de nuestra máquina virtual. Contiene una memoria, una pila de operandos y una varible booleana para determinar si la ejecución ha terminado, es decir, si se ha ejecutado la instrucción `HALT`. En esta clase se encuentra el método público boolean execute(Instruction instr), que es el encargado de ejecutar la instrucción que le llega como parámetro modificando convenientemente la memoria y/o la pila de operandos. Si la ejecución genera un error el método devuelve false.
- InstructionParser: Es la clase encargada de parsear un string que contiene una posible instrucción. Concretamente dispone de un método Instruction parse(String s) que devuelve la instrucción almacenada en s o bien null si s no representa ninguna instrucción de las permitidas.
- Main: Es la clase que contiene el método main de la aplicación.

El resto de información concreta para implementar la práctica será explicada por el profesor durante las distintas clases de teoría y laboratorio. En esas clases se indicará qué aspectos de la implementación se consideran obligatorios para poder aceptar la práctica como correcta y qué aspectos se dejan a la voluntad de los alumnos.

## 6. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicada en la cabecera de la práctica.

El fichero debe tener al menos el siguiente contenido[1]:

[1] Puedes incluir también opcionalmente los ficheros de información del proyecto de Eclipse

- Directorio `src` con el código de todas las clases de la práctica.
- Fichero `alumnos.txt` donde se indicará el nombre de los componentes del grupo.