Práctica 1

Máquina Virtual
(curso 2016-2017)

Puri Arenas
Facultad de Informática
UCM

Descripción de la Práctica 1

Implementación de una **máquina virtual** simple que permite ejecutar instrucciones máquina secuenciales.

La Práctica 1 se implementa como un interprete de comandos:

- `HELP: muestra la ayuda`
- `QUIT: cierra la aplicación`
- `NEWINST BC: añade el bytecode BC al programa bytecode`
- `RUN: ejecuta el programa bytecode almacenado`
- `RESET: inicializa el programa bytecode`
- `REPLACE N: reemplaza el bytecode número N por un nuevo bytecode que se solicita por teclado al usuario.`

No se distingue entre mayúsculas y minúsculas.

La Máquina Virtual (CPU)

La máquina virtual está compuesta por:

- Una *pila de operandos* (`OperandStack`) que utilizan los bytecodes para poder ejecutarse.
- Una *memoria* (`Memory`) donde se almacenan datos.

La máquina virutal se implementa a través de la clase CPU.

```
public class CPU {

  private Memory memory = new Memory();
  private OperandStack stack = new OperandStack();
  private boolean halt = false;

  public boolean isHalted() {...}
  public String toString(){...}
  public boolean execute(ByteCode bc) {...}
}
```


Clases necesarias para la Práctica 1

- **`Main:`** Lanza la aplicación. Crea un objeto del tipo **`Engine`**, e invoca a su método `start`.
- **`Engine:`** Es el interprete de comandos.
- **`Command:`** Encapsula los distintos comandos de la aplicación (`RUN`, `HELP`, `NEWINST BC`, `...`)
- **`CommandParser:`** Parsea un `String` y lo transforma en un comando, o en `null` si no se ajusta a la sintaxis de ninguno de ellos.
- **`CPU:`** Maquina virtual capaz de ejecutar un bytecode.
- **`Memory.`** Memoria ilimitada de la máquina virtual.
- **`OperandStack.`** Pila de operandos de la máquina virtual.
- **`ByteCode:`** Encapsula los distintos bytecodes de la práctica.
- **`ByteCodeParser:`** Parsea un `String` y lo transforma en un bytecode, o en `null` si no se ajusta a la sintaxis de ningulo de ellos.
- **`ByteCodeProgram.`** Almacena una secuencia de bytecodes.

Clase Main

```
public class Main {
 public static void main(String args[]) {
   Engine engine = new Engine();
   engine.start();
 }
}
```


Clase Engine

```
public class Engine {

   private boolean end; // controla que no se ha ejecutado el comando QUIT
   private ByteCodeProgram bcProgram;
   private static Scanner in = new Scanner(System.in);


   public Engine(){
     this.bcProgram = new ByteCodeProgram();
     this.end = false;
   }

   public void start(){...}
   public boolean executeCommandRun() {...}
   public static boolean showHelp() {...}
   public boolean endExecution() { this.end = true; }
   public boolean addByteCodeInstruction(ByteCode bc) {...}
   public boolean resetProgram() {...}
   public boolean replaceByteCode(int replace) {...}

}
```


Clase Engine

```
public void start(){
  this.end=false;
  String line = "";
  while (!end) {
    // muestra el prompt
    Command command = null;
    // lee una línea "line"
    command = CommandParser.parse(line);
    if (command == null) // mensaje de error
    else {
        // mensaje informativo del comando a ejecutar
        if (!command.execute(this)) // mensaje de error
        // si bcProgram no es vacio, mostrar el programa bytecode
  }
System.out.println("Fin de la ejecucion....");
in.close();
}
```


Clase Engine

```
public boolean excuteCommandRun() {
  CPU cpu = new CPU(); // se puede poner como atributo?
  // inicialización de variables
  while (i< this.bcProgram.getNumberOfByteCodes() && ...) {
     ByteCode instr = this.bcProgram.getByteCode(i);
     if (cpu.execute(instr)) {
       ... // muestra el estado de la CPU
       i++;
     }
     else … // salir del bucle
  }
  return ... // ejecución correcta?
}
```

Clase Command

```
public class Command {
   private ENUM_COMMAND command;
   private ByteCode bytecode;
   private int replace;

   public Command(ENUM_COMMAND comando, int replace){
    this.command = comando;
    this.replace = replace;
   }
   public Command(ENUM_COMMAND comando) {
    this.command = comando;
    this.bytecode = null;  this.replace = -1;
   }
   public Command(ENUM_COMMAND comando, ByteCode bc) {
    this.command = comando;
    this.bytecode = bc;  this.replace = -1;
   }
   public String toString() {...}
   public boolean execute(Engine engine){
      if (this.command == ENUM_COMMAND.QUIT) {
          engine.endExecution();
          return true;
      }
      else if (this.command == ENUM_COMMAND.HELP) {
          Engine.showHelp();
          return true;
      }
     else if (this.command == ENUM_COMMAND.RUN) return engine.excuteCommandRun();
     else ...
   }
}
```


Clase CommandParser

```
public class CommandParser {

public static Command parse(String line) {
    // elimina los caracteres en blanco iniciales
    line = line.trim();
   // admite varios blancos separando las palabras.
   String []words = line.split(" +");
   if (words.length == 0) return null;
   else {
    words[0] = words[0].toLowerCase();
    if (words.length==1){
      if (words[0].equalsIgnoreCase("HELP")) return new Command(ENUM_COMMAND.HELP);
      else  ...
    }
    else if (words.length==2){
      if (words[0].equalsIgnoreCase("NEWINST")){
        String bytecode = words[1];
        ByteCode bc = ByteCodeParser.parse(bytecode);
        return new Command(ENUM_COMMAND.NEWINST,bc);
     }
     else ...
    }
    else if (words.length==3){
      if (words[0].equalsIgnoreCase("NEWINST")){
        String bytecode = words[1] + " " + words[2];
        ByteCode bc = ByteCodeParser.parse(bytecode);
        return new Command(ENUM_COMMAND.NEWINST,bc);
     }
      else return null;
   else return null;
}
```


Clase ByteCode

- Un programa bytecode en realidad representa la compilación de un programa imperativo, como veremos en la Práctica 3.
- Los bytecodes de la Práctica 1 son muy simples y realmente corresponden a la traducción de asignaciones. Para ello se utiliza una tabla de variables tv que nos dará su índice asociado.
- Veamos algunos ejemplos:

x = 2	push 2 ; store 0	tv[0]=x
y = x * 5	load 0 ; push 5 ; mul ; store 1	tv[1]=y
z = y / x	load 1 ; load 0 ; div ; store 2	tv[2] =z
write z	load 2; out	

Clase ByteCode

x = 2	push 2 ; store 0	tv[0]=x
y = x * 5	load 0 ; push 5 ; mul ; store 1	tv[1]=y
z = y / x	load 1 ; load 0 ; div ; store 2	tv[2] =z

- Si ejecutamos el programa imperativo obtenemos que:
  - x = 2; y = 10; z = 5;
- Si ejecutamos el programa bytecode asociado obtenemos:

```
push 2          stack[0] = 2
store 0         stack = empty,  memory[0] = 2 (x=2)
load 0          stack[0] = 2
push 5          stack[1] = 5
mult            stack[0] = 10
store 1         stack = empty,  memory[1] = 10 (y=10)
load 1          stack[0] = 10;
load 0          stack[1] = 2;
div             stack[0] = 5
store 2         stack = empty,  memory[2] = 5 (z=5)
```

Clase ByteCode

```
public class ByteCode {

  private ENUM_BYTECODE name;
  private int param;

  public ByteCode(ENUM_BYTECODE op) {
    this.name = op;
  }
  public ByteCode(ENUM_BYTECODE mnemo, int param) {
   this.name = mnemo;
   this.param = param;
  }
  public ENUM_BYTECODE getOpcode() { return this.name; }
  public int getParam() { return this.param; }
  public String toString() {
     return this.name + (name.getNumArgs() > 0 ? (" " + param) : "");
  }
}
```


Tipo Enumerado ENUM_BYTECODE

```
public enum ENUM_BYTECODE {
   ADD, SUB, MUL, DIV,
   PUSH(1), LOAD(1), STORE(1),
   OUT, HALT;

   private int numArgs;

   ENUM_BYTECODE() { this(0); }

   ENUM_BYTECODE(int n) {
    numArgs = n;
   }
   public int getNumArgs() { return numArgs; }
}
```


Clase Memory

```
public class Memory {

  private Integer[] memory;
  private int size;
  private final static int MAX_MEM = 200;
  private boolean empty;

  public Memory() {
   this.empty = true;
   this.memory = new Integer[Memory.MAX_MEM];
   this.size = Memory.MAX_MEM;
   for (int i=0; i< this.size; i++) this.memory[i]=null;
  }
  private void resize(int pos){
    // pone como nueva capacidad del array pos*2, en caso
    // de que pos >= this.size
  }
  public boolean write(int pos, int value) {...}
  public int read(int pos) {...}
  public String toString(){
     if (this.empty) return "<vacia>";
     else {
      String s = "";
      for (int i = 0; i < this.size;  i++)
       if (this.memory[i]!=null) s = s + " [" + i +"]:" + this.memory[i];
     }
    return s;
  }
}
```


Clase OperandStack

```
public class OperandStack {
  public static final int MAX_STACK = 100;
  private int[] stack;
  private int numElems;

  public OperandStack() {
    this.numElems=0;
    this.stack = new int[OperandStack.MAX_STACK];
  }

  public boolean isEmpty() { … }
  public boolean push(int value) { … }
  public int pop() { … }
  public int getNumElems() { … }
  public String toString(){
     if (this.numElems==0) return "<vacia>";
     else {
      String s="";
      for (int i=0; i < this.numElems; i++) s = s + this.stack[i] +  "  ";
      return s;
     }
  }
}
```

Clase CPU

```
public class CPU {

  private Memory memory = new Memory();
  private OperandStack stack = new OperandStack();
  private boolean halt = false;

  public boolean isHalted() { … }
  public String toString(){
     String s = System.getProperty("line.separator") +
     "Estado de la CPU: " + System.getProperty("line.separator") +
     "  Memoria: " + this.memory + System.getProperty("line.separator") +
     "  Pila: "  + this.stack + System.getProperty("line.separator");
     return s;
  }
  public boolean execute(ByteCode bc) {
   if (isHalted()) return true;
   else if (bc.getOpcode()==ENUM_BYTECODE.PUSH) return this.pila.push(bc.getParam());
   else if (bc.getOpcode() == ENUM_BYTECODE.LOAD) {
     int pos = bc.getParam();
     return this.stack.push(memory.read(pos));
   }
   else ...
  }
}
```


Clase ByteCodeProgram

```
public class ByteCodeProgram {

  private static final int MAX_INSTR = 100;
  private ByteCode[] bcprogram;
  private int numBc;

  public ByteCodeProgram(){
    this.bcprogram = new ByteCode[ByteCodeProgram.MAX_INSTR];
    this.numBc = 0;
  }
  public boolean addBCInstruction(ByteCode instr) {...}
  public int getNumberOfByteCodes() {...}
  public ByteCode getByteCode(int i) { return this.bcprogram[i]; }
  public String toString(){...}
  public void reset() {...}
  public void replace(int replace, ByteCode bc) {...}
}
```