Informe

Lenguajes de Programación

Integrantes:

Franco Lautaro Carranza

LU:110630

Alejandro Alberto Stessens

LU:83262

A)

Aclaración: La clase “Principal” no va a tener INST ni VTs ya que el método main es estático y no tenemos métodos ni atributos de instancias.

Esquematización de los INSTs:

Aclaración: “Ref” en este caso es la dirección donde se encuentra ubicada la VT en la memoria de datos (Memoria D).

INST de la clase “A”

Ref a VTA
Int v1
In t v2

INST de la clase “B”

Ref a VTB
Int v1
Int v2
Int v3

INST de la clase “C”

Ref a VTC
Int v1
Int v2
Int v3

Esquematización de las VTs:

Aclaración: “Ref” en este caso es la dirección donde se encuentra ubicado el método en el código (Código C).

VT de la clase “A”

Ref init() (de A)
Ref m1() (de A)
Refm2(int w1,int w2) (de A)

VT de la clase “B”

Ref init() (de B)
Ref m1() (de A)
Ref m1(B xb,int w1,int w2) (de B)
Ref m2(int w1,int w2) (de A)

VT de la clase “C”

Ref init() (de C)
Ref m1() (de A)
Ref m1(B xb,int w1,int w2) (de B)
Ref m2(int w1,int w2) (de C)

Esquematización de todos los registros de activación (RA) de todos los métodos:

Abreviaturas:

PTR = Puntero de retorno cuando se vuelve de la invocación

ED = Enlace dinámico

VL = Variable Local

PM = Parámetro

RA para llamada al método main:

PTR
ED
A oA = new C() (VL)

RA para llamada al método init de A,B y C:

PTR
ED
THIS (objeto instancia con la que se invoca al método)

RA para la llamada al método m1 de A:

Espacio para valor de retorno
PTR
ED
THIS (objeto instancia con la que se invoca al método)
Int value (VL)
Int a (VL)
Int rtn (VL)

RA para la llamada al método m2 de A:

Espacio para valor de retorno
PTR
ED
THIS (objeto instancia con la que se invoca al método)
Int w1 (PM)
Int w2 (PM)

RA para la llamada al método m1 de B:

Espacio para valor de retorno
PTR
ED
THIS (objeto instancia con la que se invoca al método)
B xB (PM)
Int w1 (PM)
Int w2 (PM)

RA para la llamada al método m2 de C:

Espacio para valor de retorno
PTR
ED
THIS (objeto instancia con la que se invoca al método)
Int w1 (PM)
Int w2 (PM)
B xB = new B() (VL)

B)

Explicando de manera intuitiva para realizar el for del método m1 de la clase A, se utilizó saltos para que el Program Counter (PC) del código itere sobre las mismas instrucciones del programa.

Primeramente, se utilizó un salto condicional (JumpT) para traducir la condición de finalización del for (a<10 & value*2 <10) , si esta condición se cumple el JumpT me salta a donde está la traducción del cuerpo del for , si no se cumple esta condición no se salta y el PC avanza a la próxima instrucción que sería la traducción de las sentencias cuando se sale del cuerpo del for.

Cuando se salta al cuerpo del for y se ejecuta la traducción, para que itere debemos hacer a lo último un salto (Jump) a la dirección del salto condicional (JumpT) para que nuevamente se chequee la condición pero ahora con los valores **“a”** y **“value”** actualizados, así se decide si se sigue ciclando o se finaliza.

C)

La única complicación que tuvimos es que para asignarle el valor a **“rtn”** debíamos realizar la suma de los valores de retorno de las invocaciones al método m2. Esto significa que había que reservar estos dos valores de retorno ya que lo íbamos a necesitar para realizar la suma.

Para resolver este problema, no necesitamos reservar un campo más en el registro de activación de la llamada a m1 de la clase A, con el campo de la variable local **“rtn”** nos alcanzó porque lo utilizamos para reservar el valor temporal también a este campo.

Es decir, primero invocamos al primer término (this.m2(value,a)) creando su respectivo RA y su valor de retorno lo reservamos en el RA de m1 de la clase A en el campo de la variable local **“rtn”** , en ese momento el campo de **“rtn”** va a tener un valor incorrecto , más preciso sería un valor temporal , solo el del primer término de la suma.

Luego invocamos al segundo término (this.m2(v1,v2)) y con su valor de retorno lo que hacemos es sumárselo al valor temporal que tenía el campo **“rtn”** y este resultado asignárselo nuevamente a **“rtn”** para que ahora si tenga el valor correcto.

D)

Algunas complicaciones fueron que tuvimos que modelar el if, también la creación del objeto de la clase B que luego con este objeto invocábamos a m1 pasando como parámetro a this y esto generaba un poco de confusión, debido a que con un objeto invocábamos al método m1 y con el otro objeto (this) lo pasábamos como parámetro.

De manera intuitiva para traducir el if, se realizó la misma técnica que realizamos para el for , es decir con un salto condicional (JumpT) ,si se cumplía la condición (w1<0 | w2<0) el PC saltaba al cuerpo del if , si no el PC pasaba a la próxima instrucción que sería la sentencia que sigue cuando termina el cuerpo del if en el código.

Luego para realizar la creación del objeto xB tuvimos que guardar en el RA de m2C la referencia al objeto xB que sería la dirección en la memoria heap (INST). Luego invocando a xb.init() le seteamos los valores de los atributos almacenándolos en el INST de xB .

Por ultimo lo que hicimos para invocar a xB.m1(this,-1,w2), es hacer el RA para m1 y pusimos como el objeto this (objeto instancia con la que se invoca al método) a xB que estaría almacenado en el RA de este método (m2C) como variable local y luego pusimos al this del RA de este método (m2C) como parámetro del RA de m1.

También otra observación muy importante fue que había dos return en el método ,la complicación de esto es que el método podía finalizar de dos maneras (en el return del if o el return de afuera del if).Por lo que cuando finalizaba la invocación del método que estaba en el return , en cualquiera de estos dos return , cuando se volvía de la llamada a la invocación, se debía actualizar el registro actual, libré y saltar al puntero de retorno que se almaceno en el RA de la llamada de m2 de la clase C en este caso. Por lo tanto, se utilizó el mismo set de instrucciones de finalización del método, es decir, hicimos que el puntero de retorno de los dos RA que estaban en la invocación que tenían los returns apunten a la misma instrucción, para así cuando se volvía de la invocación pasen por las mismas instrucciones del código de traducción que en este caso era para finalizar el método m2 de la clase C.