Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3641 - Lenguajes de Programación I
Septiembre-Diciembre 2009

Carnet:_____

Nombre:_____

Quiz

(20 puntos)

Pregunta 0	Pregunta 1	Total
10 puntos	10 puntos	20 puntos

Pregunta 0 - 10 puntos

Esta pregunta consta de cinco (5) subpreguntas de selección, numeradas de 0.0 a 0.4. Cada una de las subpreguntas viene acompañada de cuatro posibles respuestas (a, b, c, d), entre las cuales sólo **una** es correcta. Ud. deberá marcar la opción que considere correcta o, si desea no contestar la subpregunta, marcar la última opción (**e**) que indica que Ud. prefiere omitir la respuesta.

Cada una de las cinco (5) subpreguntas tiene un valor de dos (2) puntos. Tres subpreguntas malas eliminan una buena. Las subpreguntas omitidas (marcadas en la opción (**e**)) no suman ni restan puntos.

0.0. La definición de algunos lenguajes, como es el caso de Java, especifica detalladamente cómo deben ser implementados los tipos básicos de números enteros y números reales. Esto es, cuántos bits deben ser utilizados para cada uno de los tipos básicos y cuál debe ser la aritmética binaria a ser utilizada (complemento a 2, complemento a 1, signo-magnitud, etcétera). En contraposición con esto, la definición de algunos otros lenguajes, como es el caso de C, da libertad a la implementación del lenguaje en cuanto a la decisión de tales detalles parcial o totalmente.

En ambos casos, tal decisión de diseño puede favorecer o no a las siguientes características del lenguaje:

I. la portabilidad de los programas escritos en el lenguaje.

II. la portabilidad de la implementación del lenguaje.

III. la eficiencia de mantenimiento de los programas escritos en el lenguaje.

IV. la eficiencia de ejecución de los programas escritos en el lenguaje.

V. la facilidad de implementación del lenguaje.

En un lenguaje del grupo de Java, ¿cuáles características se ven favorecidas?

a) Sólo I y III.

b) Sólo I, II y III.

c) Sólo IV.

d) Ninguna de las combinaciones anteriores.

e) No sabe / No contesta.

0.1. Continuando con la pregunta anterior 0.0, en un lenguaje del grupo de C, ¿cuáles características se ven favorecidas?

a) Sólo I y III.

b) Sólo I, II y III.

c) Sólo IV.

d) Ninguna de las combinaciones anteriores.

e) No sabe / No contesta.

0.2. Suponga que se cuenta con la siguiente peculiar implementación de un cierto lenguaje de alto nivel L, ejecutable sobre la arquitectura de una cierta máquina M pero que genera código para otra arquitectura diferente de una máquina M':

- un compilador de L, escrito en el lenguaje de máquina de M, que genera código en el lenguaje de máquina de M';
- el mismo compilador, pero escrito a alto nivel en el mismo lenguaje L.

Se quiere construir otra implementación de L igualmente peculiar pero contraria: ejecutable sobre M' pero que genere código para M. ¿Cuál sería la manera más eficiente, en términos de horas/persona (esto es, en tiempo de programación), de lograr esto?

a) Dar el compilador de alto nivel como entrada al compilador de bajo nivel, y reescribir el resultado para que genere M.

b) Reescribir el compilador escrito a alto nivel para que genere M, y dárselo como entrada al compilador de bajo nivel.

c) Reescribir el compilador escrito a bajo nivel para que genere M, y dárselo como entrada al compilador original de bajo nivel.

d) Es imposible lograr lo que se pide.

e) No sabe / No contesta.

0.3. El libro de texto de Scott presenta una descripción general de las fases por las que atraviesa un programa durante el proceso de compilación, siendo estas fases análisis lexicográfico y sintáctico, análisis semántico, generación de código (posiblemente intermedio y finalmente objeto), y opcionalmente mejoramiento (mal llamado optimización) de código.

Considere el siguiente fragmento de Java correspondiente al cuerpo de un método:

```
{
    int n, m;  boolean b;

    n = 3;  m = 0;  b = true;
    n = n / m;  n = n / b;
}
```

En el proceso de compilación y ejecución de este fragmento de código, ¿en cuál de las fases se detectaría el primer error?

a) En la fase de análisis sintáctico.

b) En la fase de análisis semántico.

c) En la fase de generación de código.

d) Durante la ejecución del código generado por el compilador.

e) No sabe / No contesta.

0.4. C# es un lenguaje orientado por objetos bastante similar a Java, aunque se diferencia de este último en varias características particulares. Una de tales características es el permitir que durante la programación se diferencien explícitamente lo que Scott en su libro de texto llama constantes manifiestas (obvias) o de momento de compilación (*manifest constants* o *compile-time constants*) y constantes de momento de elaboración (*elaboration-time constants*). Las primeras sólo pueden depender de valores conocidos estáticamente mientras las segundas no. C# permite hacer la diferenciación mediante las palabras claves `const` y `readonly`, respectivamente.

Cuando en C# estas palabras claves son aplicadas a una variable local entera de una subrutina recursiva, esto ocasiona:

a) En el primer caso que la variable sea forzosamente almacenada en pila y en el segundo caso que pueda ser almacenada en región estática.

b) En ambos casos que la variable sea forzosamente almacenada en región estática.

c) En ambos casos que la variable sea forzosamente almacenada en pila.

d) En el primer caso que la variable pueda ser almacenada en región estática y en el segundo caso que sea forzosamente almacenada en pila.

e) No sabe / No contesta.

Pregunta 1 - 10 puntos

En el lenguaje Java, las variables de tipo arreglo almacenan en realidad referencias a arreglos. Esto hace que las variables de tipo arreglo y los arreglos sean objetos independientes entre sí, entre los cuales se pueden crear asociaciones (*bindings*). Tal independencia se ve reflejada en los tiempos de vida de éstos, de manera tal que el tiempo de vida x de una variable de tipo arreglo y el tiempo de vida y de un arreglo con el que esta variable pueda ser asociada (*bound*) no tienen por qué ser el mismo, pudiendo x estar incluido en y, o incluirlo, o tener una relación de solapamiento con éste.

(Sin embargo, note que, gracias a que el diseño de Java es bastante limpio, el tiempo de vida de una asociación entre una variable de tipo arreglo y un arreglo siempre está incluido en los tiempos de vida de la variable y del arreglo en cuestión. En general, en Java siempre ocurre que para que una asociación esté viva es necesario que estén vivos los dos objetos asociados.)

Se desea que Ud. construya dos pequeños programas en Java que muestren ejemplos de solapamiento entre los tiempos de vida antes referidos. Específicamente, para cada uno de los dos siguientes casos, construya un pequeño programa en Java en el que haya una asociación (*binding*) entre una variable de tipo arreglo y un arreglo, y que tal asociación cumpla con la siguiente condición:

- la variable nace antes que el arreglo y también muere antes que el arreglo;
- el arreglo nace antes que la variable y también muere antes que la variable.

Aparte de la variable y el arreglo en cuestión, sus programas pueden tener cualquier otra cantidad de objetos (variables, métodos, etcétera) que Ud. considere necesarios.

Nota: Toda variable local a un método nace al inicio de la ejecución del método, independientemente del punto específico de su declaración. No considere implementaciones exóticas de Java en las que una declaración de variable local es considerada una instrucción cuya ejecución genera el nacimiento de la variable.

Con la variable naciendo antes que el arreglo y también muriendo antes que el arreglo

```
class C0 {
  static int[] foo;

  public static void main (String[] args) {
    bar();
    foo = null;
  }

  static void bar() {
    int[] baz;
    foo = baz = new int[42];
  }
}
```

La variable `baz` se crea *antes* que el arrego de 42 posiciones, y también muere *después* del arreglo, en virtud de ser una variable local, y que `foo` mantiene acceso al arreglo al salir de `bar()`.

Con el arreglo naciendo antes que la variable y también muriendo antes que la variable

```
class C1 {
  static int[] foo;

  public static void main (String[] args) {
    foo = new int[42];
    bar();
  }

  static void bar() {
    int[] baz = foo;
    baz = foo = null;
    ...
  }
}
```

El arreglo de 42 posiciones se crea *antes* que la variable `baz`, y también muere *después* de la variable, pues (en teoría) no hay manera de accederlo y será recuperado por el recolector de basura, antes de salir de `bar()`.