

LTP 2B - L1 - EXAMEN PRIMER PARCIAL - 28 DE OCTUBRE DE 2020

Alumno:

1 Cuestiones (40%)

1.1. (10%) Describe el mecanismo de paso de parámetros por valor (*call-by-value*). Ilústralo con un ejemplo.

1.2. (10%) Describe las principales características del paradigma concurrente. Discute los problemas planteados por el uso de la concurrencia. Da al menos un ejemplo de lenguaje de programación que permita usar la concurrencia.

1.3. (10%) ¿Cuál es el propósito del *análisis semántico* durante la compilación? Da ejemplos que ilustren su uso.

1.4. (10%) Describe cómo se usan los *compiladores* en la implementación de lenguajes de programación. Da al menos dos ejemplos de lenguajes compilados.

2 Problemas (60%)

2.1. (20%) Las siguientes clases de Java describen herramientas e instrumentos de ferretería que pueden almacenarse en diversos *recipientes* capaces de soportar un peso máximo medido en gramos. (...)

1. ¿Podría `protected` reemplazarse por `private` en la línea 22? ¿Y en la 20? Justifica cada respuesta.

Solución. En la línea 22 de la definición de la clase `Recipient`, i.e.,

```
22 protected long loadCapacity=0;
```

no sería posible porque el atributo `loadCapacity` no podría utilizarse en la línea 40 del constructor de la subclase `Box`, i.e.,

```
40 public Box() { loadCapacity = MAXBOX; }
```

y tampoco en la línea 45 del constructor de la subclase `Tray`, i.e.,

```
45 public Tray() { loadCapacity = MAXTRAY; }
```

En cambio, el atributo `tools` declarado en la línea 20 de la definición de la clase `Recipient`, i.e.,

```
20 protected Tool tools[ ];
```

sí podrá hacerse `private` porque no hay referencias a él en ninguna de las subclases de `Recipient`.

2. ¿Hay algún uso de la *sobrecarga*? ¿Y de la *coerción*? ¿Dónde?

Solución. No hay sobrecarga, pero se utiliza *coerción implícita* en la línea 31, i.e.,

```
31 loadCapacity = loadCapacity - tool.getWeight();
```

para restar `tool.getWeight()` (de tipo `int`) de `loadCapacity` (de tipo `long`). En la línea 28, i.e.,

```
28 if (tool.getWeight() > loadCapacity || i>MAXTOOLS) return false;
```

se da un caso similar al comparar con el operador `>` los valores de `tool.getWeight()` y `loadCapacity`.

3. ¿Cuál de los siguientes grupos de instrucciones provocan un error? Si no es así, ¿cuál es el resultado (*true/false*) de `r.addPiece(t)` en cada caso? Justifica tus respuestas.

- | | |
|---|---|
| <pre>Recipient r = new Recipient(); (1) Tool t = new Tool(); r.addPiece(t);</pre> | <pre>Recipient r = new Recipient(); (2) Tool t = new Hammer(); r.addPiece(t);</pre> |
| <pre>Recipient r = new Box(); (3) Tool t = new Drill(); r.addPiece(t);</pre> | <pre>Recipient r = new Tray(); (4) Hammer t = new Hammer(); r.addPiece(t);</pre> |
| <pre>Tray r = new Tray(); (5) Tool t = new Tool(); r.addPiece(t);</pre> | <pre>Box r = new Recipient(); (6) Tool t = new Hammer(); r.addPiece(t);</pre> |

4. Como parte de las llamadas a `r.addPiece(t)` en el código (1)-(6) anterior, ¿cuál es, en cada caso, el resultado devuelto por `tool.getWeight()` en la línea 28?

Solución.

- En (2) el resultado de `tool.getWeight()` es 500.
- En (3) el resultado de `tool.getWeight()` es 3000.
- En (4) el resultado de `tool.getWeight()` es 500.

- 2.2.** (10%) Considera la siguiente clase `Main`: (...)

- ¿Hay algún tipo de uso de la *sobrecarga*? ¿Y de la *coerción*? ¿Dónde?

Solución.

- Respecto a la *sobrecarga*, en la línea 58, i.e.,

```
58 uWeight = uWeight + t.tools[i].getWeight();
```

se utiliza el operador de suma para números enteros largos y en la 59, i.e.,

```
59 System.out.println("Area: "+uWeight);
```

se utiliza el operador de suma de cadenas de caracteres. También hay *sobrecarga* en el uso de `println` en las líneas 55 i.e.,

```
55 System.out.println(s.getNTools());
```

y 59 (ver arriba), ya que en el primer caso (línea 55) se aplica sobre argumentos enteros y en el segundo (línea 59) sobre cadenas de caracteres.

- Respecto a la *coerción*, en la línea 58 (ver arriba) se utiliza coerción implícita de enteros (`t.tools[i].getWeight()`) a *enteros largos*. En la línea 59 (ver arriba) se utiliza coerción implícita de enteros largos (`uWeight`) a *strings*.

- ¿Podría la variable `t` de la línea 49 declararse de clase `Recipient` en vez de `Tray`? Justifica tu respuesta.

Solución. La variable se declara en la línea 49 como sigue:

```
49 private static Tray t = new Tray();
```

y la variación propuesta sería

```
49' private static Recipient t = new Tray();
```

con lo que no habría diferencias en el comportamiento del programa, ya que no se utilizan atributos ni métodos específicos de `Tray` a través del objeto `t`.

- ¿Dónde se alojan los *objetos* referenciados por las variables `h`, `d`, y `t` (líneas 47-49)? ¿Y las *variables* `h`, `d` y `t`?

Solución. Las declaraciones son las siguientes:

```
47 private static Tool h = new Hammer();
48 private static Tool d = new Drill();
49 private static Tray t = new Tray();
```

Los objetos se alojan en el *heap* mientras que las variables, al ser declaradas `static`, se alojan en el área de memoria estática.

2.3. (20%) El lenguaje de programación C incluye una *instrucción for* con el siguiente formato:

```
for (init ; test ; inc) inst
```

donde `init` inicializa una variable ‘contador’, `test` es una condición booleana que normalmente hace referencia a dicha variable, `inc` modifica el contador. Finalmente, `inst` es un bloque de instrucciones (*cuerpo del bucle*). Al principio de la ejecución se ejecuta la instrucción `init`. Las instrucciones del cuerpo del bucle se repiten mientras la condición `test` sea *cierta*. Tras cada ejecución de `inst` se ejecuta `inc`.

1. Realizar los cambios necesarios en la sintaxis de SIMP para poder utilizar dicha instrucción. ¿Qué nuevos símbolos *terminales* aparecen?
2. Realizar los cambios necesarios en la semántica de paso pequeño y paso grande (*small-step/big-step*) de SIMP para poder ejecutar programas que incluyan la nueva instrucción.

3. Mostrar la traza de ejecución completa para el siguiente programa:

```
n := 3;  
s := 0;  
for (i := 1 ; i <= n ; i := i+2 )  
    s := s + i
```

2.4. (10%) ¿Es correcta la siguiente terna de Hoare? Justifica tu respuesta indicando los cálculos necesarios para obtenerla.

{ $x < 0$ }

$y := x+1$;

 if $x > y$ then $z := x$ else $z := y$

{ $2z \geq x+y$ }