Ingeniería Informática - Depto. de Informática e Ingeniería de Sistemas
Examen de Programación 2 - 16 de Septiembre de 2011

- Disponer sobre la mesa en lugar visible un ***documento de identificación*** provisto de fotografía. Escribir ***nombre y dos apellidos*** en cada una de las hojas de papel de examen.
- El examen consta de tres partes: 1ª parte (problemas 1, 2 y 3), 2ª parte (problemas 4 y 5) y 3ª parte (problema 6º). Cada parte del examen debe comenzar a resolverse en una nueva hoja, para facilitar su corrección por el equipo de profesores de la asignatura.
- El tiempo total previsto para realizar el examen es de ***tres horas***. No está permitido consultar libros ni apuntes, excepto los documentos facilitados por los profesores de la asignatura: *Breve resumen del lenguaje Java*, *Capítulo 1 de las Notas del curso* y *Documentación de las clases Java utilizadas en la asignatura*.

PARTE 1ª

Problema 1º (1.0 punto)

Un número natural es cuadrado perfecto si es el cuadrado de un número natural. El 0 es cuadrado perfecto ya que $0 = 0^2$. El 1 es cuadrado perfecto ya que $1 = 1^2$. El 4 es cuadrado perfecto ya que $4 = 2^2$. El 9 es cuadrado perfecto ya que $9 = 3^2$. El 16 es cuadrado perfecto ya que $16 = 4^2$. Y así sucesivamente. En cambio, ni el 2, ni el 3, ni el 5, ni el 6, ni el 7, ni el 8, ni el 10, etc. son cuadrados perfectos.

Se pide especificar formalmente los siguientes dos métodos Java, escribiendo los correspondientes predicados pre y postcondición.

```
/* [n] ha de ser un número natural y el método devuelve [true] si [n] es un cuadrado perfecto
   y [false] si no lo es */
public static boolean esCuadradoPerfecto (int n)
```

```
/* [x] ha de ser un número natural y el método devuelve el menor número natural mayor que [x]
   que sea cuadrado perfecto */
public static int siguienteCuadradoPerfecto (int x)
```

Problema 2º (1.0 punto)

Se pide:

1. Demostrar formalmente que el método *iSumar* que se muestra a continuación no es correcto. Para ello es preciso **probar formalmente** la no corrección de alguna de sus instrucciones.
2. Modificar la especificación del método *iSumar* para que sea correcto su diseño, sin alterar nada de su código. La modificación realizada no debe alterar la corrección del método *sumar*.

```
/* Pre: n ≥ 0 */
/* Post: sumar(n) = (Σα ∈ [1, n].α) */
public static int sumar (int n) {
    return iSumar(n,0,0);
}

/* Pre: n ≥ 0 ∧ hasta ≥ 0 ∧ hasta ≤ n ∧ suma ≥ 0 */
/* Post: iSumar(n, hasta, suma) = (Σα ∈ [1, n].α) */
private static int iSumar (int n, int hasta, int suma) {
    if (hasta == n)
        return suma;
    else
        return iSumar(n, hasta+1, suma+hasta+1);
}
```

Problema 3º (1.5 puntos)

Se pide escribir, para cada uno de los dos métodos de búsqueda en tabla que se presentan a continuación, **un predicado invariante y una función de cota** para sus respectivos bucles. Cada predicado invariante ha de ser lo suficientemente fuerte como para sustentar las pruebas de corrección del código de su método (no se pide aportar ninguna prueba) y cada función de cota debiera permitir probar la terminación de su bucle (tampoco se piden las pruebas).

```
/* Pre: (∃α ∈ [0, T.length − 1].T[α] = x) */
/* Post: T[busquedaGarantizada(T, x)] = x */
public static int busquedaGarantizada (int[] T, int x) {
    int desde = 0;
    while ( T[desde]!=x ) {
        desde++;
    }
    return desde;
}
```

```
/* Pre: cierto */
/* Post: ((∃α ∈ [0, T.length − 1].T[α] = x) → T[busqueda(T, x)] = x)
                    ∧((∀α ∈ [0, T.length − 1].T[α] ≠ x) → busqueda(T, x) < 0) */
private static int busqueda (int[] T, int x) {
    int posicion = 0;
    boolean esta = false;
    while ( posicion! = T.length  &&  !esta ) {
        esta = T[posicion]==x;
        posicion++;
    }
    if (esta)
        return posicion-1;
    else
        return -1;
}
```

PARTE 2ª

Problema 4º (3.0 puntos)

Escribir de forma clara y legible, el conjunto de pruebas formales que constituyen la demostración rigurosa de la corrección del método *divNatural* de la clase *Operaciones*. Los objetos de la clase *Operaciones* tienen definidos dos atributos privados de tipo int, *cociente* y *resto*, cuyos valores son modificados por el método *divNatural*.

```
/* Pre: num ≥ 0 ∧ den > 0 */
/* Post: num = cociente * den + resto ∧ 0 ≤ resto ∧ resto ≤ den */
public void divNatural (int num, int den) {
    if (num < den) {
        cociente = 0;
        resto = num;
    }
    else {
        divNatural(num-den, den);
        cociente++;
    }
}
```

Problema 5º (1.0 puntos)

Deducir de forma rigurosa el orden de la función de coste de una invocación *divNatural(n,2)*, dependiendo del valor del parámetro *n*. Se valorarán exclusivamente los siguientes aspectos de la solución presentada: ecuaciones recurrentes, resolución de las ecuaciones anteriores, deducción del orden de la función de coste y claridad y limpieza de los cálculos y explicaciones.

PARTE 3ª

Problema 6º (2.5 puntos)

Los alumnos matriculados en las diferentes asignaturas de una titulación universitaria se encuentran registrados en un fichero binario con la siguiente estructura:

```
<ficheroMatriculas> ::=  {  <matricula>  }
<matricula> ::= <nip> <codigo>
<nip> ::= int
<codigo> ::= int
```

Los datos del fichero no están ordenados.

Se pide diseñar el método Java *alumMatric* que se especifica a continuación, junto con los métodos auxiliares que sean necesarios, sin utilizar **ni un solo bucle** en el código de ninguno de ellos. En la solución se valorará tanto la corrección del código de los métodos como la precisión de sus especificaciones.

```
/**
 * Pre: [nombre] corresponde al nombre de un fichero binario que almacena las matriculas
 *      de los alumnos de un cierto curso universitario
 * Post: Devuelve [true] si y solo si el alumno cuyo número de identificación personal universitaria
 *       es igual a [nip] está matriculado en alguna asignatura del curso según consta en el fichero
 *       [nombre]; en caso contrario devuelve [false]
 */
public static boolean alumMatric (String nombre, int nip)
```