Universidad Carlos III de Madrid

Programación de Sistemas
GIT, GISC, GISA, GITT

Leganés, 19 de mayo de 2015
Duración de la prueba: 50 min

Examen final. Convocatoria ordinaria. Teoría
Puntuación: 3 puntos sobre 10 del examen

Sólo una opción es correcta en cada pregunta. Cada respuesta correcta suma 0.15 puntos. Cada respuesta incorrecta resta 0.05 puntos. Las preguntas no contestadas no suman ni restan puntos.

Marca: Anula: No uses:

- Marca la respuesta a cada pregunta con una equis ("X") en la tabla de abajo.
- Si marcas más de una opción o ninguna opción, la pregunta se considera no contestada.
- Rellena **tus datos personales** antes de comenzar a realizar el examen.

Modelo: A

Nombre:	Grupo:

Firma:

NIA:

	A	B	C	D		A	B	C	D
1					11				
2					12				
3					13				
4					14				
5					15				
6					16				
7					17				
8					18				
9					19				
10					20				

1.- Supongamos que insertamos algunos elementos en un árbol binario de búsqueda con las siguientes claves numéricas: `1, 2, 3, 4, 5, 6`. En qué orden debemos insertarlos para que después, la operación de búsqueda sea más eficiente?

(a) El orden en que se inserten los elementos no afecta a la eficiencia de las operaciones de búsqueda posteriores.

(b) `6, 5, 4, 3, 2, 1`.

(c) `1, 2, 3, 4, 5, 6`.

(d) *** `3, 1, 5, 2, 6, 4`.

2.- Indica qué hace este método sobre una lista enlazada como las que se ha programado en clase que contenga 2 o más nodos.

```
public void method() {
  if (first != null && first.getNext() != null) {
    Node<E> aux = first;
    while (aux.getNext().getNext() != null) {
        aux = aux.getNext();
    }
    System.out.println(aux.getInfo());
  }
}
```

(a) Imprime todos los elementos de la lista menos el último.

(b) Imprime el último elemento.

(c) *** Imprime el penúltimo elemento.

(d) Imprime todos los elementos de la lista menos los dos últimos.

3.- ¿Cuál de las siguientes estructuras de datos lineales es más eficiente para extraer el elemento que más tiempo lleva almacenado en ella?

(a) *** Una cola.

(b) Ninguna de las estructuras indicadas es eficiente en este caso.

(c) Un árbol de búsqueda binario.

(d) Una pila.

4.- Sea un montículo vacío, de la variante que almacena el elemento con la máxima prioridad en su raíz. Supón que le insertamos algunos elementos con las siguientes prioridades y en este mismo orden: `1, 2, 3, 4, 5` (siendo la prioridad 5 la máxima prioridad). Si imprimimos las prioridades de los elementos en el montículo en *in-order*, ¿en qué orden aparecerán por pantalla?

(a) `1, 3, 4, 2, 5`

(b) *** `1, 4, 3, 5, 2`

(c) `1, 2, 3, 4, 5`

(d) `5, 4, 2, 1, 3`

5.- Selecciona la estructura de datos más eficiente para implementar una cola con prioridad.

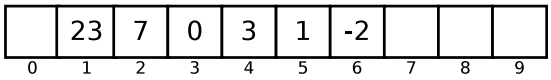
(a) Una cola.

(b) *** Un montículo.

(c) Un árbol binario de búsqueda.

(d) Ninguna de las otras opciones es correcta.

6.- El array de `int`s de tamaño 10 que se muestra a continuación representa un árbol binario de enteros de tamaño 6. Marca cuál de las siguientes frases es correcta:



(a) Ninguna de las otras respuestas es correcta.

(b) *** El árbol cumple los requisitos para ser un montículo.

(c) El árbol tiene exáctamente 2 hojas.

(d) El árbol cumple los requisitos para ser un árbol binario de búsqueda.

7.- Dada la clase `Mammal` (en español, mamífero) y la clase `Bear` (en español, oso) que deriva de la primera, indica qué sentencia sería *incorrecta* para imprimir el valor del atributo protected `especies` (en español, especie) de la clase padre desde un método no estático de la clase hija.

(a) *** `System.out.println(super(especies));`

(b) `System.out.println(((Mammal)this).especies);`

(c) `System.out.println(especies);`

(d) `System.out.println(super.especies);`

8.- Dada la clase `Component` y la clase `Memory` que deriva de la primera, selecciona cuál de las siguientes afirmaciones es correcta:

(a) `Memory` sólo puede implementar la interfaz de `Component`.

(b) Desde el constructor de `Component` se puede acceder a todos los atributos de `Memory` mediante un *downcasting* (también conocido como *narrowing*).

(c) `Memory` sólo puede reescribir un método de `Component` si se trata de un método abstracto.

(d) *** Todos los objetos de la clase `Memory` también son objetos de la clase `Component`.

9.- Dado el siguiente método:

```
public static int x(int n, int m) {
  if (n<=1) {
    return m;
  } else {
    return x(n-1, n + x(n-2,m));
  }
}
```

(a) No es recursivo.

(b) Es recursivo lineal.

(c) Es un caso de recursión mutua.

(d) *** Tiene recursión anidada.

10.- Indica cuál de las siguientes afirmaciones es correcta:

(a) Un mismo objeto manejador de eventos puede capturar eventos de varios componentes, pero cuando recibe un evento no es posible conocer en cuál de ellos se ha producido dicho evento.

(b) Es imposible tener un panel con una distribución `FlowLayout` dentro de un panel con una distribución `BorderLayout`.

(c) Un mismo objeto manejador de eventos sólo puede capturar eventos de un único componente.

(d) *** Para un mismo componente se puede registrar más de un escuchador de eventos.

11.- ¿Qué devolvería el siguiente método al ejecutar la sentencia `method(5)`?

```
public static int method(int n) {
  if (n<=2) {
    return 1;
  } else {
    return 5*(n-1)*method(n-2);
  }
}
```

(a) Ninguna de las otras respuestas es correcta.

(b) 100

(c) 120

(d) *** 200

12.- Dado el siguiente método:

```
public static void method(int n) {
  if (n>1) {
    method(n/2);
  }
  System.out.print(n%2);
}
```

(a) *** Imprime la representación en binario de n, siendo n natural.

(b) Imprime los números entre 1 y $n/2$, siendo n natural.

(c) No tiene caso base.

(d) Imprime los números pares entre n y 2.

13.- Dada la clase `Employee` con el método abstracto `double getSalary()` y la clase `Programmer` que deriva de ella, indica cuál de las siguientes afirmaciones es *incorrecta*:

(a) *** Todos los métodos de `Employee` tienen que ser abstractos.

(b) Para crear una instancia de `Programmer`, esta clase debe implementar el método `getSalary()`.

(c) `Employee` puede tener un constructor.

(d) No puede crearse ninguna instancia de `Employee`.

14.- Dada una cola vacía, indica qué devolvería la llamada al método `front()` de la cola al terminar de ejecutar la siguiente secuencia de operaciones: `enqueue(5); front(); dequeue(); enqueue(3); enqueue(7); dequeue();`

(a) 3

(b) 5

(c) Ninguna de las otras opciones son correctas.

(d) *** 7

15.- Indica cuál de las siguientes afirmaciones es *incorrecta*.

(a) `GridLayout` permite organizar los componentes en pantalla en una matriz bidimensional.

(b) `FlowLayout` y `BorderLayout` se pueden asociar a cualquier panel dentro de la interfaz.

(c) Para colocar los componentes por coordenadas es necesario "anular" previamente el *layout* con `setLayout(null)`.

(d) *** Cuando se programa un escuchador `ActionListener`, no es necesario implementar obligatoriamente su método `actionPerformed()`.

16.- Dada la clase `Course` (en español, asignatura) y la clase `Programming` que deriva de la primera, y el siguiente fragmento de código, indica cuál de las líneas de código es *incorrecta*:

```
Course course = new Course("Programming");
Programming programming = new Programming();
```

(a) `course = programming;`

(b) *** `programming = (Programming) course;`

(c) `course = (Programming) programming;`

(d) `course = (Course) programming;`

17.- Indica cuál de las siguientes afirmaciones es *incorrecta*:

(a) *** La altura de un árbol binario completo de n elementos es $\lfloor \log_2(n+1) \rfloor, \forall n \in \mathbb{N}$.

(b) Todos los árboles binarios de búsqueda son árboles ordenados.

(c) Todos los subárboles de un árbol binario de búsqueda son árboles binarios de búsqueda.

(d) Todos los subárboles de un montículo son montículos.

18.- Dadas dos interfaces `I1` e `I2`, una clase `C1` que implementa `I1` y otra clase `C2` que deriva de `C1` e implementa `I2`, selecciona cuál de las siguientes instrucciones es *incorrecta*:

(a) `I1 ref = new C1();`

(b) `I1 ref = new C2();`

(c) *** `I2 ref = new C1();`

(d) `I2 ref = new C2();`

19.- En una `deque` implementada con una lista enlazada (no doblemente enlazada) en la que se tiene una referencia al primer nodo (`top`) y al último (`tail`), indica cuál de las siguientes operaciones es en general más costosa.

(a) *** `removeLast`

(b) `insertLast`

(c) `removeFirst`

(d) `insertFirst`

20.- El árbol binario de búsqueda que utilizamos en las prácticas tenía un método `insert` que recibía como argumentos la nueva clave y el nuevo valor a insertar. Suponiendo que al árbol se le añade un método de extracción que reemplaza siempre por la izquierda, indica el orden en que se imprimirían los valores por pantalla al ejecutar las siguientes operaciones:

```
BSTree<Integer> bst = new BSTree<Integer>();
bst.insert(new Integer(10), new Integer(10));
bst.insert(new Integer(15), new Integer(15));
bst.insert(new Integer(12), new Integer(12));
bst.insert(new Integer(5), new Integer(5));
bst.insert(new Integer(6), new Integer(6));
bst.insert(new Integer(17), new Integer(17));
bst.insert(new Integer(3), new Integer(3));
bst.remove(new Integer(10));
System.out.println(bst.toStringPostOrder());
```

(a) `6, 5, 3, 15, 12, 17`

(b) *** `3, 5, 12, 17, 15, 6`

(c) `3, 5, 6, 12, 15, 17`

(d) `3, 6, 12, 17, 5, 15`