Programación de Sistemas
Examen final (convocatoria extraordinaria)

Leganés, 28 de junio de 2018
Duración de la prueba: 40 minutos

Examen convocatoria extraordinaria (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.*

- 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.

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.- En el esqueleto de código que se presenta a continuación, ¿cómo debe implementarse en la línea 03 la zona indicada con `/* TO DO */`?

```
01 public class Point{
02         private double x, y;
03         public Point(double x, double y) {  /* TO DO */ }
04         public Point() {  /* TO DO */ }
05         public double getX() {  /* TO DO */ }
06         public void setX(double x){  /* TO DO */ }
07         public double getY() {  /* TO DO */ }
08         public void setY(double y){  /* TO DO */ }
09         public double distance() {/* TO DO */ }
10         public double distance(Point otroPunto) {/* TO DO */ }
11  }
```

(a) *** `this.x = x; this.y = y;`

(b) `x = this.x; y = this.y;`

(c) Los argumentos del constructor deben tener un nombre diferente al de los atributos de la clase. De otro modo, no puede implementarse.

(d) No es necesario incluir código, los argumentos ya dicen los valores que tendrán `x` e `y`.

2.- Tenemos definidas las clases `A` y `B` en un mismo paquete como se muestra a continuación. Elige la opción válida para lograr que un programa imprima la siguiente salida: `MethodA MethodB`

```
public class A {
    public static void methodA(){
        System.out.println("MethodA");
    }
}
public class B {
    public void methodB(){
        System.out.println("MethodB");
    }
}
```

(a) *** `B b = new B(); A.methodA(); b.methodB();`

(b) `A.methodA(); B.methodB();`

(c) `A a = new A(); a.methodA(); B.methodB();`

(d) `A a = new A(); a.methodA(); b.methodB();`

3.- Se tienen definidas las siguientes clases en fichero diferentes. ¿Qué salida produce el programa?

```
package p;
public class Padre {
    public int publica;
    private int privada;
```

```
    protected int protegida;
    public Padre(int a, int b, int c){
        publica   = a;
        privada   = b;
        protegida = c;
    }
}
package p;
public class Hijo extends Padre{
    public Hijo(int a, int b, int c){
        super(a,b,c);
    }
}
package p;
public class Main {
    public static void main(String[] args) {
        Padre padre = new Padre(10,20,30);
        System.out.print (padre.publica);
        System.out.print (" " + padre.protegida);
        Hijo hijo = new Hijo(1,2,3);
        System.out.print (" " + hijo.publica);
        System.out.print (" " + hijo.protegida);
    }
}
```

(a) *** 10 30 1 3

(b) 10 0 1 0

(c) 1 3 1 3

(d) 10 30 10 30

4.- Dado el siguiente programa. ¿Qué afirmación es cierta?

```
public class Padre {
    public Padre(){}
    public void imprime(){
        System.out.print("imprimePadre ");
    }
}
public class Hijo extends Padre{
    public Hijo(){}
    public void imprime(){
        System.out.print("imprimeHijo ");
    }
}
public class Hija extends Padre {
    public Hija(){}
    public void imprime(){
        System.out.print("imprimeHija ");
    }
```

```
}
public class MainFamilia {
    public static void main(String[] args) {
        Padre[] familia = {new Padre(), new Hijo(), new Hija()};
        for (int i=0; i<3; i++){
            familia[i].imprime();
        }
    }
}
```

(a) *** En la instrucción *for* se utiliza polimorfismo. La salida del programa es: `imprimePadre imprimeHijo imprimeHija`.

(b) La salida del programa es: `imprimePadre imprimePadre imprimePadre`.

(c) En la instrucción *for* se utiliza sobrecarga. La salida del programa es: `imprimePadre imprimeHijo imprimeHija`.

(d) En la instrucción *for* se utiliza sobrecarga. La salida del programa es: `imprimePadre imprimePadre imprimePadre`.

5.- Dado el siguiente programa donde todas las clases están dentro del mismo paquete pero en ficheros diferentes, indique qué afirmación es correcta.

```
public abstract class A {
    private int i;
    // Declarar entero j
    A(int i){ this.i = i;}
    public abstract void metodo(int i);
    public String toString(){
        return "i = " + i + " j = " + j;
    }
}
public class B extends A{
    public B(int entero){
        super(entero);
    }
}
public class Main {
    public static void main(String[] args) {
        B b = new B(10);
        System.out.println(b.toString());
    }
}
```

(a) *** El programa no compila pues `B` debe implementar el método `metodo(int i)`.

(b) El programa imprime: `i = 10 j = 10`.

(c) El programa no compila, debe quitarse en la declaración del método `metodo` de la clase A la palabra clave `abstract`.

(d) El programa imprime: `i = 10 j = 0`.

6.- Las siguientes clases están definidas dentro del mismo paquete, pero en ficheros diferentes. Indica qué afirmación es correcta.

```
 1 public class Persona {
 2     protected String nombre;
 3     Persona(String s){ nombre = s;}
 4 }
 5 public class Alumno extends Persona {
 6     private String id;
 7     Alumno(String s, String id) {
 8         super(s);
 9         this.id = id;
10     }
11 }
12 public class Main {
13     public static void main(String[] args) {
14         Persona p1 = new Persona("Lucia");
15         Persona p2 = new Alumno("Carla", "123);
16         Alumno  a1 = new Alumno("Antonio", "456");
17         Alumno  a2 = (Alumno) new Persona("Carolina");
18     }
19 }
```

(a) *** En la línea 15 se realiza una compatibilidad de tipos hacia arriba válida y en la línea 17 se realiza una compatibilidad de tipos inválida

(b) En la línea 15 se realiza una compatibilidad de tipos hacia arriba válida y en la línea 17 se realiza una compatibilidad de tipos válida.

(c) En la línea 15 se realiza una compatibilidad de tipos hacia arriba inválida y en la línea 17 se realiza una compatibilidad de tipos válida

(d) En la línea 15 se realiza una compatibilidad de tipos hacia arriba inválida y en la línea 17 se realiza una compatibilidad de tipos inválida.

7.- La clase `Fecha` tiene implementado el método `enRango()` como sigue. La clase `TestEnRango` tiene la siguiente estructura. ¿Qué instrucción debemos colocar en la línea 14?

```
public boolean enRango(int valor, int min, int max){
    return (valor >= min && valor <= max);
}
```

```
 1 import static org.junit.Assert.*;
 2 import org.junit.BeforeClass;
 3 import org.junit.Test;
 4 public class TestEnRango  {
 5     public static Fecha fecha;
 6     @BeforeClass
 7     public static void inicializacion(){
 8          fecha = new Fecha();
 9     }
10     @Test public void testAntes() {...}
```

```
11     @Test public void testLimiteInferior() {...}
12     @Test
13     public void testEnMedio() {
14         // TO DO
15     }
16     @Test public void testLimiteSuperior() {...}
17     @Test public void testDespues() {...}
18 }
```

(a) *** `assertEquals(fecha.enRango(16, 1, 31), true);`

(b) `assert(enRango(16, 1, 30));`

(c) `assert (fecha.enRango(16, 1, 30), true);`

(d) `assertEquals(enRango(16, 1, 31), true);`

8.- Se tiene un método que devuelve `true` cuando el año que recibe como parámetro es bisiesto y `false` en caso contrario: `public boolean bisiesto(int anio)`. La siguiente lista indica un conjunto de años que son bisiestos y otros que no lo son: Años bisiestos: 1604, 1608, 1612, 1616... 1684, 1688, 1692, 1696, 1704 (1700 no es bisiesto porque es divisible entre 100 y no es divisible entre 400), 1708, 1712... 1792, 1796, 1804 (1800 no es bisiesto porque es divisible entre 100 y no es divisible entre 400), 1808, 1812... 1892, 1896, 1904 (1900 no es bisiesto porque es divisible entre 100 y no 400), 1908, 1912... 1992, 1996, 2000 (si es bisiesto, es divisible entre 100 y 400), 2004, 2008, 2012... 2092, 2096, 2104 (2100 no es bisiesto porque es divisible entre 100 y no 400)... 2196, 2204... 2296, 2304... 2396, 2400 (si es bisiesto, es divisible entre 100 y 400), 2404... ¿Qué conjunto de pruebas harías para comprobar la validez de `public boolean bisiesto(int anio)`?

(a) *** 1604, 1900, 2000, 2001

(b) 1604, 2400

(c) 1604, 1900, 2000, 2400

(d) 1604, 1605, 2000

9.- El siguiente método pretende calcular el cuadrado de un número basándose en la siguiente fórmula: para valores de n mayores que 1, *(n-1)*(n-1) = n*n – 2*n + 1*, aunque su implementación no es correcta. Elige la opción correcta de las siguientes para que el método calcule correctamente el cuadrado de números mayores que 1.

```
1 public int cuadrado (int n){
2     if (n <= 1){
3         return 1;
4     } else {
5         return (cuadrado (n-1) - 2*(n-1) + 1);
6     }
7 }
```

(a) *** Línea 5: sustituir por `return (cuadrado(n-1) + 2*n - 1);`

(b) Línea 5: sustituir por `return (cuadrado(n-1) - 2*cuadrado(n-2) + 1);`

(c) Línea 5: sustituir por `return ((n-1)*(n-1) - 2*n + 1);`

(d) Línea 5: sustituir por `return (cuadrado(n-1) - 2*cuadrado(n-1) + 1);`

10.- Para escribir un número decimal `n` en su correspondiente número binario `b` es necesario seguir los siguientes pasos: (1) `n<2` implica que `b=n`; (2) `n>=2` implica que `b=n/2` seguido de `n%2`. ¿Qué instrucciones deben colocarse en `L1` y `L2` para que siguiente método devuelva una cadena de caracteres que represente el número binario de `n`?

```
public String d2b(int n){
    String s;
    if (n < 2){
        // L1
    } else{
        int m = n % 2;
        // L2
    }
    return s;
}
```

(a) *** `L1: s = new String(n); y L2: s = d2b (n/2) + m;`

(b) `L1: s = new String(n); y L2: s = d2b (m) + n;`

(c) `L1: s = new String(0); y L2: s = d2b (m) + n;`

(d) `L1: s = new String(1); y L2: s = d2b (n/2) + m;`

11.- Para concatenar dos listas enlazadas ya existentes...

(a) *** hay que recorrer una lista hasta llegar a su último elemento y asignar como siguiente el primero de la otra lista

(b) hay que recorrer las dos listas hasta llegar a sus últimos elementos y asignar como siguiente al último de una de ellas el último de la otra

(c) hay que crear una nueva lista de tamaño la suma de los elementos de las listas existentes y pasar todos los elementos de las dos listas existentes a la nueva lista creada

(d) hay que asignar como siguiente al primer elemento de una lista el primer elemento de la otra lista

12.- En un árbol binario de búsqueda que representa los destinos a los que vuela una aerolínea se inserta la siguiente información de forma secuencial, actuando dicha información también como clave (ordenación alfabética): “Washington”, “Toronto”, “Madrid”, “Barcelona”, “Berlin”, “Amsterdam”, “Lisbon”, ¿Cuál es la altura del árbol resultante?

(a) *** 5

(b) 3

(c) 4

(d) 7

13.- Se insertan en un montículo (*max-heap*) los siguientes elementos secuencialmente 4,6,8,2,1,3 y luego se extrae el elemento con clave 6. ¿Cuál es el recorrido postorden del montículo resultante?

(a) *** 2,1,4,3,8

(b) 1,3,2,4,8

(c) 2,3,1,4,8

(d) 1,2,3,4,8

14.- En una lista doblemente enlazada que utiliza nodos dummy (referenciados por *top* y *tail*) y que está vacía se cumple que:

(a) *** *tail.getPrev()* es *top*

(b) *top.getNext()* es *tail.getPrev()*

(c) *top.getNext()* es *null*

(d) *tail* y *top* apuntan al mismo nodo

15.- ¿Cuántos intercambios (*swaps*) son necesarios para ordenar el siguiente array 5,3,4,1,2 de menor a mayor utilizando Heap Sort? Ten en cuenta que el nodo con clave 5 será la raíz, los nodos con claves 3 y 4 sus hijos izquierdo y derecho, respectivamente, y los nodos con claves 1 y 2 los hijos izquierdo y derecho de 3.

(a) *** 6

(b) 5

(c) 8

(d) 7

16.- En una lista enlazada simple en la que cada nodo conoce únicamente quién es el nodo siguiente, y en la que tenemos una referencia al primer nodo (*top*) y al último nodo (*tail*). Indica cuál de las siguientes operaciones es más costosa.

(a) *** Extracción por el final

(b) Extracción por el principio

(c) Inserción por el principio

(d) Inserción por el final

17.- Si insertamos uno a uno (y en el orden dado) los nodos con claves 3,1,2,5,7,4,6 en un árbol binario de búsqueda. ¿Qué nodo quedaría como hijo izquierdo del nodo cuya clave es 5 después de terminar el proceso de inserción?

(a) *** El nodo cuya clave es 4

(b) El nodo cuya clave es 2

(c) El nodo cuya clave es 6

(d) El nodo cuya clave es 5 no tendría hijo izquierdo

18.- Dada una pila *p* y una cola *q* vacías. Indica qué devolvería la llamada al método *front()* de la cola al terminar de ejecutar la siguiente secuencia de operaciones: *p.push(3); p.push(2); p.push(1); q.enqueue(p.pop()); q.enqueue(5); p.top(); q.dequeue()*.

(a) *** 5

(b) 3

(c) 1

(d) 2

19.- Dada una lista enlazada con *first* referenciando al primer nodo, indica qué hace el siguiente método:

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

(a) *** Lanza una *NullPointerException*.

(b) Imprime la información del último nodo.

(c) Imprime la información del penúltimo nodo.

(d) Imprime la información de todos los nodos de la lista.

20.- Dado un array desordenado de mil elementos, ¿cuál de los siguientes algoritmos requiere más cantidad de memoria para ordenarlo?

(a) *** Merge Sort

(b) Heap Sort

(c) Quick Sort

(d) Insertion Sort