simplyjarod@gmail.com

FPRG

Exámenes y test resueltos 2004-2009

(no están todos)

Dpto. Ingeniería de Sistemas Telemáticos
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid

FUNDAMENTOS DE PROGRAMACIÓN. ENERO 2004.

Normas de examen:

- Con libros y apuntes.
- Duración: 2 horas.
- No se contestará ninguna pregunta durante el examen
- Por favor, conteste cada pregunta en una hoja separada, ajustándose a la extensión estimada

Pregunta 1 (2 puntos) [extensión estimada: menos de 10 líneas]

Tenemos una clase para un cajero automático que dispensa dinero sí y solo si hay saldo:

```
class Cajero {
  private int saldo;
  void retirar (int cantidad) { saldo-= cantidad; }
}
```

1. Cree una nueva clase de excepciones, de nombre **`"NoTieneSaldo"`**, cuyo constructor recibe como argumento el saldo disponible (antes de retirar la cantidad).

2. Modifique el método **`"retirar"`** para que lance la anterior excepción cuando no haya suficiente saldo antes de retirar; es decir, no se puede retirar por encima del saldo en cada momento.

```
class NoTieneSaldo extends Exception {
  NoTieneSaldo (int saldo) {
    super(Integer.toString(saldo));
  }
}
```

```
void retirar (int cantidad) throws NoTieneSaldo {
  if (cantidad > saldo)
    throw new NoTieneSaldo(saldo);
  saldo-= cantidad;
}
```

Pregunta 2 (2 puntos) [extensión estimada: menos de 15 líneas]

En un supermercado hay varias cajas donde los clientes pueden pagar. Estas cajas imprimen unos *tickets* que incluyen dos números:

- el primer número identifica la caja en la que paga el cliente
- el segundo número cuenta cuántos clientes han pasado hasta el momento por cualquiera de las cajas del supermercado.

Así por ejemplo, un recibo etiquetado como 4-13 indica que la caja 4 ha cobrado al décimo tercer cliente. Si el siguiente cliente paga en la caja 2, su recibo estará etiquetado como 2-14.

Si la clase **`"Caja"`** sirve para representar a cada una de las cajas del supermercado, se pide completar sus atributos, su constructor, y el método que devuelve las etiquetas de los recibos.

```
class Caja {
  // miembros de la clase a incorporar por el alumno
  Caja (int numero) {
    // a desarrollar por el alumno
  }
  String ticket () {
    // a desarrollar por el alumno
  }
}
```

```
class Caja {
  private static numeroVenta= 0;
  private int numeroCaja;

  Caja (int numero) {
    numeroCaja= numero;
  }

  String ticket () {
    numeroVenta++;
    return numeroCaja + "-" + numeroVenta;
  }
}
```

Pregunta 3 (2 puntos) [extensión estimada: menos de 15 líneas]

Disponemos de un método para calcular el área de un triángulo conocidos sus tres vértices

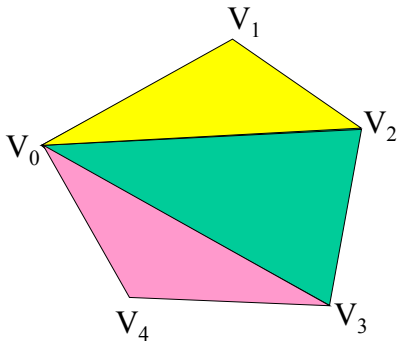
```
static double area (Punto a, Punto b, Punto c) { ... }
```

Por otra parte, representaremos un polígono por medio de una clase que ofrece un método para enumerar la serie de vértices que determinan el polígono:

```
class Poligono {
  Enumeration vertices () {
    // entrega una serie de objetos de tipo Punto
  }
}
```

Desarrolle un método para averiguar la superficie de un polígono con tres o más lados:

```
static double area (Poligono p) {
  // a desarrollar
}
```



Para resolver esta pregunta, considere que el área de un polígono se calcula sumando las áreas de los triángulos componentes, como muestra la figura adjunta: tome el primer vértice V_0 y trace líneas a los vértices sucesivos.

```
static double area (Poligono p) {
  Enumeration enumeracion= p.vertices();
  double total= 0.0;
  Punto v0= (Punto)enumeracion.nextElement();
  Punto v1= (Punto)enumeracion.nextElement();
  while (enumeracion.hasMoreElements()) {
    Punto v2= (Punto)enumeracion.nextElement();
    total+= area(v0, v1, v2);
    v1= v2;
  }
  return total;
}
```

Pregunta 4 (2 puntos) [extensión estimada: menos de 30 líneas]

Suponga que la siguiente clase **"Almacen"**, se utiliza para crear almacenes de objetos. En estos almacenes podrán guardarse objetos que mas tarde podrán recuperarse.

```
/* Almacén de objetos.
 * Los objetos se guardan con una clave, y
 * se recuperan usando es misma clave.
 */
public class Almacen {

    /* Array donde se almacenaran las
     * claves y los datos guardados.
     */
    private Object [][] datos;

    /* Constructor.
     * Inicializa datos para poder almacenar n datos.
     */
    public Almacen(int tamano) {
       // a desarrollar
    }

    /* Guarda un dato y su clave.
     * Si la clave ya existe, se sustituye el dato viejo por el nuevo.
     * Si el array datos esta lleno, se lanza una excepción.
     */
    public void guardar(String clave, Object dato) throws Exception {
       // a desarrollar
    }

    /* Devuelve el dato asociado a la clave dada.
     * Si encuentra la clave, ésta desaparece del almacén.
     * Si no encuentra la clave, devuelve null.
     */
    public Object sacar(String clave) {
       // a desarrollar
    }
}
```

El constructor de esta clase toma como parámetro la capacidad máxima del almacén, es decir, el número máximo de objetos que podrá contener.

El método **"guardar"** se utiliza para meter en el almacén un objeto y su clave. La clave que permite posteriormente identificar el objeto guardado. Si la clave ya existe, se sustituye el dato viejo por el nuevo. Cuando el almacén esta lleno, este método lanza una excepción al intentar guardar un nuevo objeto.

Los objetos se sacan del almacén llamando al método **"sacar".** A este método se le pasa como parámetro la clave que se utilizo al guardar el objeto. Si en el almacén no existe ningún objeto asociado a la clave dada, el método devuelve *null*.

Internamente, los objetos y las claves se almacenan en el atributo **"datos",** que es un *array* de dos dimensiones de tamaño 2xN, siendo N la capacidad del almacén.

Se pide escribir el cuerpo del constructor, y de los métodos **"guardar"** y **"sacar".**

```
public Almacen(int n) {
        datos = new Object[2][n];
}
```

```
public void guardar(String clave, Object dato) throws Exception {
    int nDatos= datos[0].length;
    for (int i= 0 ; i < nDatos; i++) {
        String estaClave= datos[0][i];
        if (estaClave != null && estaClave.equals(clave)) {
            datos[1][i] = dato;
            return;
        }
    }
    for (int i= 0 ; i < nDatos; i++) {
        if (datos[0][i] == null) {
            datos[0][i] = clave;
            datos[1][i] = dato;
            return;
        }
    }
    throw new Exception("Almacen completo");
}
```

```
public Object sacar(String clave) {
    int nDatos= datos[0].length;
    for (int i= 0 ; i < nDatos; i++) {
        String estaClave= datos[0][i];
        if ((estaClave != null) && (estaClave.equals(clave))) {
            datos[0][i] = null;
            return datos[1][i];
        }
    }
    return null;
}
```

Pregunta 5 (2 puntos) [extensión estimada: menos de 25 líneas]

Se desea desarrollar un sistema de selección de correo electrónico. Para cada mensaje que se recibe, el sistema realiza varias comprobaciones, indicando finalmente si debe seleccionarse o no el mensaje recibido. Si todas las comprobaciones dan resultado cierto (*true*), el mensaje se selecciona; y si alguna comprobación falla, el mensaje se descarta. El sistema usa las siguientes clases.

```
class Mensaje {
  private String remitente;          // el que envía el mensaje
  private String destinatario;       // al que recibe el mensaje
  private String cuerpo;             // el contenido del mensaje
  private String documentoAsociado; // si lleva un documento asociado
  private boolean urgente;           // es urgente o normal

public Mensaje (String remitente, String destinatario, String cuerpo,
                String documento, boolean urgente) {…}

// suponga que existen métodos públicos para acceder a todos los atributos
}

// Conjunto de comprobaciones de selección
class SelectorMensajes {

  public SelectorMensajes (){…}

  // Añade una nueva comprobación de selección
  public void anadirComprobacion (Comprobacion c) {…}

  // Chequea si el mensaje m satisface todas las comprobaciones
  public boolean satisface (Mensaje m) {…}
}
```

Pregunta 5.1

Escriba la **"interface Comprobación"**, que define cómo son las clases que realizan una comprobación sobre un mensaje.

Pregunta 5.2

Escriba una clase que implemente **"Comprobacion"**, dando por buenos los mensajes los mensajes que sean urgentes y no tengan documento asociado.

Pregunta 5.3

Complete los atributos de la clase "**SelectorMensajes**" y el método **"satisface"**.

```
interface Comprobacion {
  public boolean esValido (Mensaje m);
}
```

```
class ValidoUrgenteSinDocumento implements Comprobacion {
  public boolean esValido (Mensaje m) {
    return m.isUrgente() &&
           ((m.getDocumento () == null) || m.getDocumento ().equals (""));
  }
}
```

```
private Vector comprobaciones;

  public boolean satisface (Mensaje m) {
    for (Enumeration e = comprobaciones.elements(); e.hasMoreElements; ) {
      Comprobacion test= (Comprobacion)e.nextElement();
      if (! test.esValido (m))
        return false;
    return true;
  }
```

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN. SEPTIEMBRE 2004.

Normas de examen:

- Con libros y apuntes.
- Duración: 2 horas.
- Responda a cada pregunta en hojas separadas
- No se contestará ninguna pregunta durante el examen.

Pregunta 1 (2 puntos)

Para obtener la representación en base B de un número decimal N, puede dividirse repetidamente el valor N dado por la base B hasta que el cociente sea 0, de forma que la concatenación en orden inverso de todos los restos resultantes en las divisiones anteriores, es la representación pedida.

Por ejemplo, el número decimal 867 en base 7 se representa como 2 3 4 6. Este valor se calcula dividiendo el valor 867 entre 7, todas las veces que se puede:

```
867/7 = 123 resto=6
123/7 = 17  resto=4
 17/7 = 2   resto=3
  2/7 = 0   resto=2
```

y escribiendo los restos resultantes en orden inverso al de su cálculo.
Nótese que `2*7*7*7 + 3*7*7 + 4*7 + 6` vale `867`.

Se pide escribir un método **recursivo** que imprima por pantalla la representación en base B de un número decimal N. Use un espacio en blanco para separar entre si los restos al imprimirlos.

```
public void pregunta1(int n, int b);
```

Pregunta 2 (3 puntos)

Una máquina que realiza esculturas en mármol es manejada desde un método que toma como parámetro un array de tres dimensiones. Este array representa la figura a esculpir, y sus tres dimensiones se corresponden con los ejes espaciales X, Y y Z, divididos en milímetros. Los valores de este array son de tipo booleano, indicando que puntos del bloque de mármol a esculpir deben eliminarse (**`false`**), y que puntos deben mantenerse (**`true`**). Así, pasándole a este método un array tridimensional de booleanos, la máquina es capaz de esculpir la figura que representa.

En esta pregunta se desarrollará una clase, llamada **`Escultura`**, cuyos objetos representarán las figuras a esculpir, y que se usará para proporcionar al método escultor los arrays con los que trabajará.

Pregunta 2.1 (1 punto):
Sabiendo que la clase **`Escultura`** almacena el array con la forma de la figura como un atributo privado, que este array se le pasa al constructor como parámetro, y que existe un método llamado **`getForma`** que devuelve el array almacenado, se pide escribir el código de esta clase.

Pregunta 2.2 (1 punto):
Suponiendo que se desea esculpir versiones en miniatura de las esculturas, añada a la clase anterior un método llamado **`miniatura(int n)`**. Este método tomará como parámetro el factor de reducción (int n) de la miniatura a producir, y devolverá un nuevo objeto de la clase **`Escultura`** que represente una figura **`n`** veces más pequeña que la original. Para saber que puntos de la miniatura son sólidos y cuales son aire, divida la escultura original en bloques de **`n`**x**`n`**x**`n`** puntos, elija el punto de coordenadas menores del bloque, y use el valor de ese punto. Si el punto elegido es sólido, haga que el punto correspondiente de la miniatura sea sólido, y si es aire haga que sea aire.

Pregunta 2.3 (1 punto):
Escriba otro método que devuelva la situación del centro de gravedad de la escultura. El centro de gravedad se calcula como la media aritmética de las coordenadas de todos los puntos sólidos de la escultura. La posición del centro de gravedad se devolverá como un array que almacene tres valores de tipo **`float`**.

Pregunta 3 (2 puntos)

En una lista encadenada de números enteros, deseamos conocer cuantos de ellos son pares (se considera el 0 como número par). Dada la siguiente clase **Nodo**:

```
public class Nodo {
  int valor;
  Nodo siguiente;

  public Nodo (int v){ this ( v, null,);}
  public Nodo (int v, Nodo s,){ valor=v; siguiente =s;}
  public int pares(){
  ...
  }
}
```

Implemente el método **pares** que devuelve la cuenta de números pares contenidos en la lista.

Pregunta 4 (3 puntos)

En un supermercado se han definido las siguientes clases:

- **Producto**: define las características básicas de un producto: identificador del producto (**idproducto**) y precio del mismo (**precio**).
- **Fecha**: define una fecha como un día, un mes y un año.
- **Alimento**: aquellos productos que son comestibles tienen además una fecha de caducidad (**caducidad**).
- **ProductosEcologicos**: define a aquellos productos que cumplen una norma específica de respeto a la naturaleza y que se caracterizan por disponer de un código de origen que garantiza su carácter de ecológicos.

Si disponemos del siguiente código:

```
public class Producto {
   private int idproducto;
   private int precio;
   public Producto (int id, int p){idproducto=id; precio=p;}
   public int dameIdProducto () { return idproducto;}
   public int damePrecio () { return precio;}
}
public class Alimento extends Producto {
   private Fecha caducidad;
   public Alimento(int id, int p, Fecha c){super(id,p); caducidad =c;}
   public Fecha dameCaducidad () { return caducidad;}
}
public class Fecha {
      private int dia;
      private int mes;
      private int año;
      public Fecha (int d, int m, int a){ dia=d; mes=m; año=a;}
      public int dameDia (){ return dia;}
      public int dameMes (){ return mes;}
      public int dameAño (){ return año; }
}
public interface ProductoEcologico {
      public double codigoOrigen ();
}
```

Pregunta 4.1 (1 punto)**:** Codifique la clase **Carne** como una subclase de Alimento

Pregunta 4.2 (1 punto)**:** Codifique la clase **TerneraEcologica** como un tipo de carne que es además un **ProductoEcologico**

Pregunta 4.3 (1 punto): Codifique el siguiente método

boolean hayAlimentosEcologicos(Producto[] Almacen)

que devuelve **true** si en **Almacen** hay algún **Alimento** que sea un **ProductoEcologico**.

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN. SEPTIEMBRE 2004.

Soluciones

Pregunta 1 (2 puntos)

```
public void ejercicio1(int n, int b) {
    if (n >= b)
      ejercicio1(n/b, b);
    System.out.print(n%b + " ");
}
```

Pregunta 2 (3 puntos)

Pregunta 2.1 (1 punto)

```
class Escultura {

    /**
     * Forma de la escultura
     */
    private boolean[][][] forma;

    /**
     * Construye una escultura con la forma que pasan como parametro.
     */
    public Escultura(boolean[][][] forma) {
        this.forma = forma;
    }

    /**
     * Devuelve la forma de la escultura
     */
    public boolean[][][] getForma() {
        return forma;
    }
}
```

Pregunta 2.2 (1 punto)

```
    /**
     * Crea una miniatura de si misma.
     */
    public Escultura miniatura(int n) {

        // Calcula tamaño de la miniatura
        int tx = forma.length / n;
        int ty = forma[0].length / n;
        int tz = forma[0][0].length / n;

        // Crea array para almacenar la miniatura
        boolean[][][] m = new boolean[tx][ty][tz];

        // Esculpir la miniatura
        for (int x = 0; x < tx; x++)
            for (int y = 0; y < ty; y++)
                for (int z = 0; z < tz; z++)
                    m[x][y][z] = forma[x * n][y * n][z * n];

        // Devolver nueva clase con la miniatura
        return new Escultura(m);
    }
```

Pregunta 2.3 (1 punto)

```
    /**
     * Calcula el centro de gravedad de la figura.
     */
    public float[] centroGravedad() {

        // Array para almacenar coordenadas de centro de gravedad
        float cg[] = new float[3];
        cg[0] = cg[1] = cg[2] = 0;

        // Almacena el numero de puntos solidos (trues)
        int peso = 0;

        // Recorre la figura, y actualiza cg y peso.
        for (int x = 0; x < forma.length; x++)
            for (int y = 0; y < forma[0].length; y++)
                for (int z = 0; z < forma[0][0].length; z++)
                    if (forma[x][y][z]) {
                        peso++;
                        cg[0] += x;
                        cg[1] += y;
                        cg[2] += z;
                    }
        // Faltaba dividir por el numero de puntos solidos
        cg[0] /= peso;
        cg[1] /= peso;
        cg[2] /= peso;

        // Retornar el centro de gravedad.
        return cg;
    }
```

Pregunta 3 (2 puntos)

```
public int pares() {
    int cuenta = 0;
    for (Nodo n = this; n != null; n = n.siguiente) {
        if (n.valor % 2 == 0)
            cuenta++;
    }
    return cuenta;
}
```

Pregunta 4 (3 puntos)

Pregunta 4.1 (1 punto)

```
public class Carne
        extends Alimento {
    int proteinas;

    public Carne(int id, int p, Fecha f, int prt) {
        super(id, p, f);
        proteinas = prt;
    }
}
```

Pregunta 4.2 (1 punto)

```
public class TerneraEcologica
        extends Carne
        implements ProductoEcologico {
    double codigoOrigen;

    public TerneraEcologica(int id, int p, Fecha f, int prt, double co) {
        super(id, p, f, prt);
        codigoOrigen = co;
    }

    public double codigoOrigen() {
        return codigoOrigen;
    }
}
```

Pregunta 4.3 (1 punto)

```
boolean hayAlimentosEcologicos(Producto[] almacen) {
    for (int i = 0; i < almacen.length; i++) {
        if (almacen[i] instanceof ProductoEcologico)
            return true;
    }
    return false;
}
```

Fundamentos de Programación

14 de febrero de 2005

Patrón

Pregunta 1 [1 punto]
¿Por qué los métodos de la clase *Math* se pueden usar sin llamar a *new* previamente?

1 [+1,00] Porque sus métodos son estáticos.
2 [-0,50] Porque siempre existe un método de esta clase que se crea al arrancar el programa.
3 [-0,50] No es cierto; siempre hay que crear un objeto con *new* antes de poder usar los métodos.

Pregunta 2 [1 punto]
Dado el siguiente código, indique qué aparecerá en pantalla:

```
int[] m= new int[10];
for (int i= 0; i < m.length; i++)
  m[i]= i;
for (int i= 0; i < m.length; i= i + 2)
System.out.print(m[i] + " ");
```

1 [+1,00] 0 2 4 6 8
2 [-0,33] 0 2 4 6 8 10
3 [-0,33] 1 3 5 7 9
4 [-0,33] 1 3 5 7 9 11

Pregunta 3 [1 punto]
Dada una clase *Punto* para trabajar con puntos en un espacio bidimensional, y el siguiente código, indique qué ocurrirá al ejecutar.

```
Punto p= new Punto(0.0, 0.0);
Punto q= new Punto(0.0, 0.0);

if (p == q)
  System.out.println("Son iguales");
else
  System.out.println("Son distintos");
```

1 [+1,00] Imprimirá "Son distintos" porque se comparan referencias.
2 [-0,50] Imprimirá "Son iguales" porque los objetos son iguales.
3 [-0,50] Imprimirá "Son distintos" porque las variables son diferentes.

Pregunta 4 [1 punto]
Dado el siguiente código para recorrer una lista encadenada, indique que ocurrirá al ejecutar.

```
for (Nodo nodo= cabecera; nodo != null; nodo= nodo.getSiguiente()) {
  nodo= new Nodo(null, nodo);
  }
```

1 [+1,00] Salvo que la lista estuviera vacía, nos metemos en un bucle sin fin.

2 [-0,50] Todos los elementos de la lista pasan a "null".

3 [-0,50] Se inserta un elemento "null" entre cada dos elementos de la lista original.

Pregunta 5 [1 punto]

Dado el siguiente código, indique qué ocurriría al ejecutar.

```
public class Igualdad {
  private int i= 10;
  public static void main(String[] args) {
    Igualdad i1= new Igualdad();
    Igualdad i2= new Igualdad();
    System.out.println(i1.equals(i2));
  }
}
```

1 [+1,00] Imprime "false".

2 [0] Imprime "true".

3 [0] Se produce un error de compilación por no estar definido el método "equals".

4 [0] Se produce un error de ejecución por no estar definido el método "equals".

Pregunta 6 [1 punto]

Dado el siguiente código, indique qué ocurriría al ejecutar.

```
public class Recursivo {
  public void procedimiento(int i) {
    System.out.print(i);
    if (i == 0) return;
    procedimiento(i--);
  }

  public static void main(String[] args) {
    Recursivo recursivo= new Recursivo();
    recursivo.procedimiento(5);
  }
}
```

1 [+1,00] Ninguna de las otras respuestas es correcta.

2 [-0,50] Imprime "54321".

3 [-0,50] Imprime "543210".

Pregunta 7 [1 punto]

Dado el siguiente código, indique qué ocurriría al llamar al método *m*.

```
class Clase3 {
  private int i;

  public void m() {
    for (int i= 0; i < 5; i++)
      System.out.print(this.i);
  }
}
```

1 [+1,00] Imprime "00000".

2 [-0,50] Imprime "01234";

3 [-0,50] Imprime infinitos "0".

Pregunta 8 [1 punto]
En Java, ¿puede una clase implementar varias interfaces simultáneamente?

1 [+1,00] Sí
2 [-1,00] No

Pregunta 9 [1 punto]
En Java, ¿puede una clase extender (heredar de) varias clases simultáneamente?

1 [+1,00] No
2 [-1,00] Sí

Pregunta 10 [1 punto]
¿Cuánto vale *"i"* después de ejecutar lo siguiente?

```
int i= 1; metodo(i);

void metodo(int i) { i= 2; }
```

1 [+1,00] 1
2 [-1,00] 2

Pregunta 11 [1 punto]
¿Cuánto vale *"s"* después de ejecutar lo siguiente?

```
String s= "1"; metodo(s);

void metodo(String s) { s= "2"; }
```

1 [+1,00] "1"
2 [-1,00] "2"

Pregunta 12 [0 puntos]
Todos los atributos de una clase tienen que inicializarse en el constructor.

1 [0] No es estrictamente necesario; pero sí muy conveniente.
2 [0] Cierto.
3 [0] Falso.

Pregunta 13 [1 punto]
Todos los atributos de una clase tienen que que ser privados.

1 [+1,00] No es estrictamente necesario; pero sí muy conveniente.
2 [-0,50] Cierto.
3 [-0,50] Falso.

Pregunta 14 [1 punto]
Un array de objetos, *"Object[] array"* puede contener objetos de cualquier clase.

1 [+1,00] Cierto.
2 [-1,00] Falso.

Pregunta 15 [1 punto]
Si sumamos un valor entero y otro real, el resultado será ...

1 [+1,00] real.
2 [-0,33] entero.
3 [-0,33] se provocará un error de compilación.
4 [-0,33] se provocará un error de ejecución.

Pregunta 16 [1 punto]
Indique cual de los siguientes elementos no forma parte de la signatura de un método:

1 [+1,00] El tipo del resultado que devuelve.
2 [-0,33] El orden de los argumentos.
3 [-0,33] El nombre del método.
4 [-0,33] El tipo de cada argumento.

Pregunta 17 [1 punto]
¿Qué pasaría con el siguiente código?

```
public static void main(String[] argumentos) {
  int i= 100;
  byte b= i;
  System.out.print(b);
}
```

1 [+1,00] Se produce un error de compilación.
2 [-0,50] Se produce un error de ejecución.
3 [-0,50] Imprime "100".

Pregunta 18 [1 punto]
Indique qué pasaría con el siguiente código:

```
class Madre { ... }
class Hija1 extends Madre { ... }
class Hija2 extends Madre { ... }

Hija1 h1= new Hija1(...);
Madre m= h1;
```

1 [+1,00] Compila y ejecuta.
2 [-1,00] Provoca un error de compatibilidad de tipos.

Pregunta 19 [1 punto]
Indique qué pasaría con el siguiente código:

```
class Madre { ... }
class Hija1 extends Madre { ... }
class Hija2 extends Madre { ... }

Hija1 h1= new Hija1(...);
Hija2 h2= new Hija2(...);
h2= (Hija2)h1;
```

1 [+1,00] Provoca un error de compatibilidad de tipos.

2 [-1,00] Compila y ejecuta.

Pregunta 20 [1 punto]

Dado

```
if (a > 4)
  System.out.println("test1");
else if (a > 9)
  System.out.println("test2");
else
  System.out.println("test3");
```

¿Qué valor debe tener *"a"* para que se imprima "test2"?

1 [+1,00] No hay forma.
2 [-0,25] Menor que 4.
3 [-0,25] Entre 4 y 9.
4 [-0,25] Mayor que 9.
5 [-0,25] Menor que 4 o mayor que 9.

Dpto. Ingeniería de Sistemas Telemáticos
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid

FUNDAMENTOS DE PROGRAMACIÓN. Febrero 2005.

- ✔ Con libros y apuntes
- ✔ Duración: 2 horas
- ✔ Responda a cada pregunta en hojas separadas
- ✔ Todas las preguntas tienen el mismo peso
- ✔ No se contestará ninguna pregunta durante el examen

1. Pregunta 1

En equipos electrónicos es habitual representar el color por medio de 3 componentes: rojo (R), verde (G) y azul (B). Así un color queda definido por medio de tres valores, RGB. En este ejercicio a cada componente le asignaremos un valor real entre 0.0 y 1.0, de forma que algunos colores típicos quedan definidos como en la tabla de la derecha.

(R, G, B)	***color***
(1.0, 0.0, 0.0)	rojo
(0.0, 1.0, 0.0)	verde
(0.0, 0.0, 1.0)	azul
(1.0, 1.0, 1.0)	blanco
(0.0, 0.0, 0.0)	negro

Se pide programar una clase *"Color"* que tenga:

1. tres atributos privados para los componentes R, G y B
2. un constructor al que se le pasan los valores de los tres componentes:
   - si algún valor fuera negativo, se deja en 0.0
   - si algún valor es mayor que 1, se deja en 1.0
3. tres métodos para extraer cada componente: *getR(), getG() y getB()*
4. un método *"gris()"* que modifica los atributos privados para pasar a *blanco y negro*:
   - lo que se espera de este método es que calcule el valor medio de los tres componentes y ponga los tres componentes a dicho valor
5. un método *"filtro(cr, cg, cb)"* que modifica los atributos privados aplicando un filtro:
   - se le pasan tres coeficientes, el de rojo, el de verde y el de azul
   - el método multiplica cada componente por el coeficiente correspondiente; así, filtro(1.0, 0.0, 0.0) sería un filtro de rojo, que sólo deja pasar el componente R
   - si algún coeficiente fuera menor que 0.0, se lanza una excepción
   - si el resultado de multiplicar un componente por su coeficiente es mayor que 1.0, el valor debe acotarse a 1.0
6. las constantes *ROJO, VERDE, AZUL, BLANCO y NEGRO*

2. Pregunta 2

Se pide que programe el método siguiente (suponiendo que *"m1"* y *"m2"* son distintas de *null*):

int[ ][ ] mezcla (int[ ][ ] m1, int[ ][ ] m2)

Este método combina dos matrices bidimensionales de números enteros produciendo una nueva matriz. Las matrices *"m1"* y *"m2"* pueden tener dimensiones diferentes; por ejemplo *"m1"* puede ser 2x3 y *"m2"* puede ser 4x7. En cualquier caso, son matrices rectangulares.

La matriz resultante debe tener tamaño suficiente para contener a las otras dos matrices *(m1* y *m2)*. Así, con *"m1"* de dimensiones 2x3 y *"m2"* de dimensiones 4x7 la matriz resultante *"r"* tendrá dimensiones 4x7. Los valores de la matriz resultante se calculan de la siguiente forma:

r[i][j]= 0,
si la posición (i, j) no existe en m1 ni en m2

r[i][j]= m1[i][j],
si la posición (i, j) no existe en m2; pero sí en m1

r[i][j]= m2[i][j],
si la posición (i, j) no existe en m1; pero sí en m2

r[i][j]= (m1[i][j] + m2[i][j]) / 2,
si la posición (i, j) existe en m1 y en m2

3. Pregunta 3

Suponga que existe una clase *"Punto"* para los puntos del espacio bidimensional en coordenadas cartesianas (x, y). Dicha clase consta de constructor con esos parámetros y métodos accesores públicos. Deberá escribir clases para diferentes tipos de movimientos de los puntos. Todas estas clases tienen un método para mover el punto que se les pasa como parámetro, devolviendo un nuevo punto con las nuevas coordenadas tras el movimiento.

Pregunta 1. Escriba la interfaz *"Movimiento",* con un método que devuelve un punto como resultado de haber movido el punto original tras ***"t"*** unidades de tiempo.

Pregunta 2. Escriba la clase *"Lineal"* que permite modelar el movimiento lineal, con velocidades lineales en los ejes x e y. Estas velocidades son atributos que toman valores en el constructor.

Pregunta 3. Escriba la clase *"Angular"* que gira el punto con respecto al origen de coordenadas, según una velocidad angular de giro (esta velocidad es un atributo al que se le da un valor en el constructor). Para obtener el ángulo de un punto (x, y) con respecto al origen puede llamar a *Math.atan2(y, x).* Para obtener las coordenadas cartesianas de un punto, use las operaciones

*radio * coseno(ángulo) y radio * seno(ángulo)*

Pregunta 4. Escriba el método *"mueve"* de la clase siguiente. Esta clase contiene una lista de movimientos. En su método "mueve", aplica todos los movimientos de la lista al punto. La clase *"Lista"* es la que se ha utilizado en el tercer ejercicio del curso.

```
public class Compuesto implements Movimiento {
  private Lista movimientos;

  public Compuesto() {
    this.movimientos = new Lista();
  }

  public void addMovimiento(Movimiento m) {
    movimientos.mete(m);
  }

  public Punto mueve(Punto p, double t) {
    // RELLENAR
  }
}
```

Dpto. Ingeniería de Sistemas Telemáticos
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid

FUNDAMENTOS DE PROGRAMACIÓN. Febrero 2005.

1. Pregunta 1

Color.java

```
public class Color {
    private double rojo;
    private double verde;
    private double azul;

    public Color(double rojo, double verde, double azul) {
        this.rojo = acota(rojo);
        this.verde = acota(verde);
        this.azul = acota(azul);
    }

    private double acota(double valor) {
        if (valor < 0.0) return 0.0;
        if (valor > 1.0) return 1.0;
        return valor;
    }

    public double getR() {
        return rojo;
    }

    public double getG() {
        return verde;
    }

    public double getB() {
        return azul;
    }

    public void gris() {
        double medio = (rojo + verde + azul) / 3;
        rojo = medio;
        verde = medio;
        azul = medio;
    }

    public void filtro(double cr, double cg, double cb)
            throws Exception {
        if (cr < 0.0) throw new Exception("coeficiente rojo negativo");
        if (cg < 0.0) throw new Exception("coeficiente verde negativo");
        if (cb < 0.0) throw new Exception("coeficiente azul negativo");
        rojo = acota(rojo * cr);
        verde = acota(verde * cg);
        azul = acota(azul * cb);
    }

    public static final Color ROJO = new Color(1.0, 0.0, 0.0);
    public static final Color VERDE = new Color(0.0, 1.0, 0.0);
    public static final Color AZUL = new Color(0.0, 0.0, 1.0);
    public static final Color NEGRO = new Color(0.0, 0.0, 0.0);
    public static final Color BLANCO = new Color(1.0, 1.0, 1.0);
}
```

2. Pregunta 2

int[][] mezcla(int[][] m1, int[][] m2)

```
int[][] mezcla(int[][] m1, int[][] m2) {
    int filasM1 = m1.length;
    int columnasM1 = m1[0].length;
    int filasM2 = m2.length;
    int columnasM2 = m2[0].length;

    int filasResultado = Math.max(filasM1, filasM2);
    int columnasResultado = Math.max(columnasM1, columnasM2);
    int[][] resultado = new int[filasResultado][columnasResultado];

    for (int i = 0; i < filasResultado; i++) {
        for (int j = 0; j < columnasResultado; j++) {
            boolean estaEnM1 = (i < filasM1) && (j < columnasM1);
            boolean estaEnM2 = (i < filasM2) && (j < columnasM2);

            if (estaEnM1) {
                if (estaEnM2) {
                    resultado[i][j] = (m1[i][j] + m2[i][j]) / 2;
                } else {
                    resultado[i][j] = m1[i][j];
                }
            } else {
                if (estaEnM2) {
                    resultado[i][j] = m2[i][j];
                } else {
                    resultado[i][j] = 0;
                }
            }
        }
    }
    return resultado;
}
```

3. Pregunta 3

Movimiento.java

```
public interface Movimiento {
    public Punto mueve(Punto p, double t);
}
```

Lineal.java

```
public class Lineal implements Movimiento {
    private double vx;
    private double vy;

    public Lineal(double vx, double vy) {
        this.vx = vx;
        this.vy = vy;
    }

    public Punto mueve(Punto p, double t) {
        return new Punto(p.getX() + (vx * t), p.getY() + (vy * t));
    }
}
```

Angular.java

```
public class Angular implements Movimiento {
    private double w;

    public Angular(double w) {
        this.w = w;
    }

    public Punto mueve(Punto p, double t) {
        double r = Math.sqrt((p.getX() * p.getX()) + (p.getY() * p.getY()));
        double a = Math.atan2(p.getY(), p.getX()) + (w * t);

        return new Punto(r * Math.cos(a), r * Math.sin(a));
    }
}
```

Compuesto.java

```
import java.util.*;

public class Compuesto implements Movimiento {
    private Lista movimientos;

    public Compuesto() {
        movimientos = new Lista();
    }

    public void addMovimiento(Movimiento m) {
        movimientos.mete(m);
    }

    public Punto mueve(Punto p, double t) {
        for (Iterator i = movimientos.iterator(); i.hasNext();) {
            Movimiento m = (Movimiento) i.next();
            p = m.mueve(p, t);
        }
        return p;
    }
}
```

Fundamentos de Programación

7 de febrero de 2006

Pregunta 1 [1 punto]

Indique qué escribe una llamada al método *m()*:

```
void m() {
   try {
      System.out.print("1");
      throw new Exception("2");
   } catch (Exception e) {
      System.out.print("3");
   } finally {
      System.out.print("4");
   }
}
```

1 **[+1,00]** 134
2 **[-0,33]** 1234
3 **[-0,33]** 13
4 **[-0,33]** 123

Pregunta 2 [1 punto]

Indique qué ocurre con el siguiente fragmento de programa.

```
int x1 = 10, x2 = 20, total = 0;
x1 + x2 = total;
System.out.println("TOTAL =" + total);
```

1 **[+1,00]** Produce un error de compilación.
2 **[-0,50]** Escribe: TOTAL = 0
3 **[-0,50]** Escribe: TOTAL = 30

Pregunta 3 [1 punto]

¿Cuánto vale la variable *i* en el momento de lanzarse la excepción?

```
int a[] = new int[10];
int i = 0;
while (i <= a.length)
     System.out.print(a[i++]);
```

1 **[+1,00]** 10
2 **[-0,33]** 11
3 **[-0,33]** 9
4 **[-0,33]** 0

Pregunta 4 [1 punto]

Indique qué se imprime al ejecutar el siguiente fragmento de programa.

```
Vector v1 = new Vector();
Vector v2 = new Vector();
if (v1 == v2)
   System.out.println("iguales");
else
   System.out.println("distintos");
```

1 **[+1,00]** distintos.
2 **[-1,00]** iguales.

Pregunta 5 [1 punto]

¿Por qué los métodos de la clase Math se pueden usar sin llamar a new previamente?

1 **[+1,00]** Porque son estáticos.
2 **[-0,50]** Porque Math es una clase predefinida de java.
3 **[-0,50]** Falso, es necesario llamar a new previamente.

Pregunta 6 [1 punto]

Dado el siguiente fragmento de código:

```
class Ejemplo {
   private int x = 0;
   Ejemplo(int x) { this.x = x; }
   Ejemplo() { this(1); }
}
```

¿Cuánto vale el campo (o atributo) *x* de un objeto creado mediante la llamada *new Ejemplo();*?

1 **[+1,00]** 1
2 **[-1,00]** 0

Pregunta 7 [1 punto]

Siendo

```
void doble(int x) { x = 2 * x; }
```

¿Cuánto vale *i* después de ejecutar lo siguiente?

```
int i = 1;
doble(i);
```

1 [+1,00] 1

2 [-1,00] 2

Pregunta 8 [1 punto]

Dada la siguiente clase:

```
class Punto {
   public int x = 0, y = 0;
}
```

y el siguiente método:

```
void m(Punto p) { p.x = 1; p.y = 2; }
```

Indique qué se escribe al ejecutar:

```
Punto p = new Punto();
m(p);
System.out.print(p.x + "" + p.y);
```

1 [+1,00] 1 2

2 [-1,00] 0 0

Pregunta 9 [1 punto]

Dado el siguiente fragmento de código:

```
class Uno {
   public int x;
   Uno(int x) {
      this.x = x;
      System.out.print(x);
   }
}
class Dos extends Uno {
   Dos(int x) {
      super(2 * x);
      System.out.print(x);
   }
}
```

Diga qué escribe la llamada *new Dos(1);*

1 [+1,00] 21
2 [-0,33] 22
3 [-0,33] 12
4 [-0,33] 11

Pregunta 10 [1 punto]

Indique qué formas de definir la clase HombreLobo son posibles:

```
class HombreLobo extends Hombre, Lobo {  }
class HombreLobo extends Hombre implements Lobo {  }
class HombreLobo implements Hombre, Lobo {  }
```

1 [+1,00] las dos últimas.
2 [-0,33] las dos primeras.
3 [-0,33] la primera y la última.
4 [-0,33] las tres son correctas.

Pregunta 11 [1 punto]

¿Se pueden crear objetos de clases con métodos abstractos?

1 [+1,00] No.
2 [-1,00] Sí.

Pregunta 12 [1 punto]

Dadas las siguientes clases:

```
class Perro extends Animal {}
class Gato extends Animal {}
```

Indique qué pasa con este código:

```
Animal a = new Perro();
Gato   g = (Gato) a;
```

1 [+1,00] Produce un error de ejecución al intentar convertir un objeto perro en gato.
2 [-0,50] Es correcto ya que las clases Perro y Gato son Animales.
3 [-0,50] Produce un error de compilación ya que los perros nunca son gatos.

Pregunta 13 [1 punto]

Dado el siguiente fragmento de código:

```
boolean esPositivo(int x) {
   System.out.print("P");
   return x > 0;
}
boolean esNegativo(int x) {
   System.out.print("N");
   return x < 0;
}
```

Indique qué se escribe al ejecutar:

```
int a = 10;
if (esPositivo(a) || esNegativo(a))
   System.out.print("A");
```

1 [+1,00] PA

2 [-1,00] PNA

Pregunta 14 [1 punto]

Dado el siguiente fragmento de código:

```
int x[] = {1, 2, 3, 4};

for (int i: x)
  System.out.print(i);
```

1 [+1,00] Escribe 1234

2 [-0,50] Escribe 0123

3 [-0,50] Produce un error de ejecución: el array no esta creado.

Pregunta 15 [1 punto]

Indique el resultado en la pantalla al ejecutar el siguiente fragmento de código

```
for (int i = 0; i > 5; i ++) {
  System.out.print (" " + i + " " + (i * i));
}
```

1 [+1,00] nada.

2 [-0,33] error de compilación.

3 [-0,33] 0 0 1 1 2 4 3 9 4 16

4 [-0,33] 0 0 1 1 2 4 3 9 4 16 5 25

Pregunta 16 [1 punto]

Indique cómo se lanza una excepción:

1 [+1,00] throw new Exception();
2 [-0,33] throws new Exception();
3 [-0,33] throw Exception();
4 [-0,33] throws Exception

Pregunta 17 [1 punto]

¿Qué imprime en la pantalla el siguiente fragmento de código:

```
Punto p = new Punto();
System.out.println (p);
```

1[+1,00] si en la clase Punto hay un método toString, imprime el resultado de llamarlo; si no, el resultado de llamar al método toString de Object.
2 [-0,50] si en la clase Punto hay un método toString, imprime el resultado de llamarlo; si no, lanza excepción.
3 [-0,50] las coordenadas del punto.

Pregunta 18 [1 punto]

Dadas las siguientes clases:

```
interface A {
  void a1 ();
}
interface B {
  void b1 ();
}
class C implements A, B {
  void a1 () { }
  void b1 () { }
}
```

Qué resultado daría la siguiente sentencia:

```
A a = new C();
a.b1();
```

1 [+1,00] error de compilación en la segunda línea, el método *b1* no está declarado en la *interface A*

2 [-0,50] error de compilación en la primera línea, no se puede asignar un objeto de clase *C* a una referencia *A*.

3 [-0,50] se produce una operación de especialización de tipo *(downcasting)*, y se llama al método *b1*.

Pregunta 19 [1 punto]

El siguiente método comprueba que dos arrays cuadrados son traspuestos. Indique qué línea falta.

```
boolean inversa (double [][] a, double [][] b) {
  for (int i = 0; i < a.length; i ++) {
 for (int j = 0; j < a[i].length; j ++) {
   -- LÍNEA QUE FALTA --
 }
  }
  return true;
}
```

1 [+1,00] `if (a[i][j] != b[j][i]) return false;`

2 [-0,50] `if (a[i][j] == b[j][i]) return true;`

3 [-0,50] `if (a[i][j] == b[j][i]) return true; else return false;`

Pregunta 20 [1 punto]

Dada la siguiente clase:

```
class Prueba {
  public void main (String[] args) {
 System.out.println ("Hola a todos");
  }
}
```

Compilamos e intentamos ejecutar ...

```
> java Prueba
Exception in thread "main" java.lang.NoSuchMethodError:
main
```

¿Por qué?

1[+1,00] no existe el método main con la cabecera adecuada para poder ejecutar.

2 [-0,50] se ha producido un error de compilación porque no está bien construido el método main.

3 [-0,50] se ha compilado con éxito, pero no se ha ejecutado bien porque falta el nombre del fichero (Prueba.class).

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN. FEBRERO 2006.

Normas de examen:

- Con libros y apuntes.
- Duración: 2 horas.
- Responda a cada problema en hojas separadas
- No se contestará ninguna pregunta durante el examen.

Problema 1 (3.5 puntos)

Para multar a los conductores que conducen demasiado rápido por las autopistas de peaje, vamos a desarrollar un sistema que calcula la velocidad media a la que han circulado y, si superan la velocidad máxima permitida, el sistema procederá a multarles. Para ello, se tomarán los datos de los coches cuando entran y salen de la autopista. Con estos datos se calculará la velocidad media y se emitirán las multas pertinentes.

Los datos tomados a la entrada y salida de la autopista se representan con un objeto de tipo **Registro** con 3 campos:

```
private String matricula;  // matrícula del coche
private double hora;       // instante de entrada o de salida
private double kilometro;  // punto kilométrico de entrada o salida
```

También se disponen de métodos para acceder a dichos campos o atributos (*getters)*.

Desarrollaremos la clase **GestorDeMultas** que procesa los registros creados a las entrada y salidas de los coches en la autopista, e imprime por pantalla las multas que deban ponerse. Esta clase es la siguiente:

```
class GestorDeMultas {
    // Velocidad media máxima permitida.
    public static final double MAX_VEL = 120;

    // Tamaño del array tabla.
    private static final int TAMANO = 100;

    // Array donde se guardan los objetos Registro
    private Registro[] tabla;

    // constructor
    public GestorDeMultas() {...}

    // Metodo llamado cuando un coche entra en la autopista.
    public void entrada(Registro reg) {...}

    // Metodo llamado cuando un coche sale de la autopista.
    public void salida(Registro reg) {...}
}
```

Pregunta 1.1: Escriba el cuerpo del constructor para que inicialice adecuadamente los atributos de la clase.

Pregunta 1.2: Escriba el cuerpo del método **entrada**. Este método se llama cuando un coche entra en la autopista, almacenando en la tabla el registro que describe esa entrada. Si la tabla esta llena, debe eliminarse el registro más antiguo para dejar sitio para el nuevo registro.

Pregunta 1.3: Escriba el cuerpo del método **salida**. Este método se llama cuando un coche sale de la autopista. Si aún existe el registro de entrada de este coche en la tabla, debe calcularse la velocidad media entre la entrada y la salida, e imprimir por pantalla una multa si se ha excedido la velocidad media máxima permitida. Además, se borrará el registro de entrada de la tabla. El formato del mensaje de multa debe ser como el del siguiente ejemplo:

```
Multa: 1234ABC Velocidad 234.1 entre 10.5 y 11.12 horas, entre Kms 100 y 220
```

Si no se encuentra el registro de entrada, se ignora el registro de salida.

Si hubiera varios registros de entrada para la matrícula dada, se usará el más reciente, eliminándose todos.

Problema 2 (3 puntos)

Las partículas atómicas poseen la doble naturaleza de comportarse unas veces como onda y otras como corpúsculo. El comportamiento de onda lo resumidos en su longitud de onda (λ), y el de corpúsculo en su masa (m) y su velocidad (v). Representaremos λ, m y v como números reales.

Pregunta 2.1: Escriba el interfaz de comportamiento como onda con un único método que proporciona la longitud de onda.

Pregunta 2.2: Escriba el interfaz de comportamiento como corpúsculo con dos métodos: uno para conocer la masa y otro para conocer la velocidad.

Pregunta 2.3: Escriba la clase partícula que implementa los interfaces onda y corpúsculo

Pregunta 2.4: Escriba una clase de prueba con:

a) Un método con la siguiente cabecera:

```
public static void imprimeOndas (List<Onda> lista)
```

o, alternativamente, utilizando la clase Vector:

```
public static void imprimeOndas (Vector<Onda> lista)
```

b) Un método *main* donde se construye una Lista de Ondas, se crean y almacenan 3 partículas en esta lista, y se imprime su contenido mediante el método *imprimeOndas*.

Problema 3 (3.5 puntos)

Se desea construir una clase llamada **Trayectoria** para almacenar los puntos por los que pasa un atleta en una prueba deportiva. Suponga que existe la clase **Punto** que representa a los puntos del espacio tridimensional con coordenadas x, y, z (z es la altura sobre el nivel del mar). Esta clase tiene constructor, métodos de acceso (*getters*), métodos de modificación (*setters*) y un método para calcular la distancia a otro punto; todos públicos.

El siguiente fragmento de código corresponde a la clase **Trayectoria**, donde los puntos suspensivos identifican las partes que faltan.

```
class Trayectoria {
  private ... puntos;
  private static final double DISTANCIA_MIN = 5.0;
  private static final double ALTURA_MAX = 30.0;

  public Trayectoria() {...}

  public void addPunto(Punto p) {...}

  public double getPendienteMaxima() throws Exception {...}
}
```

Pregunta 3.1: Defina el campo (o atributo) **puntos** como una lista genérica (*List*) o un *Vector* genérico de objetos de clase **Punto**. Escriba también el constructor correspondiente.

Pregunta 3.2: Escriba el método **addPunto** para añadir un nuevo punto a la trayectoria, con las siguientes condiciones:

- si la distancia al último punto almacenado es menos que **DISTANCIA_MIN**, no se añade.
- si la diferencia de altura con el último punto almacenado es mayor que **ALTURA_MAX**, haremos la altura del nuevo punto igual a la altura del último punto almacenado.

Pregunta 3.3: Escriba un método (en la clase Punto) para calcular la pendiente entre dos puntos. La pendiente se obtiene llamando al método **Math.atan2(dz, dxy)** y quedándonos con el valor absoluto; siendo "**dz**" la diferencia de alturas, y "**dxy**" la distancia en horizontal entre los puntos (esto es: considerando únicamente sus coordenadas x, y).

Pregunta 3.4: Escriba un método para calcular la pendiente mayor en la trayectoria. Si la trayectoria contiene menos de dos puntos lance una excepción.

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN. FEBRERO 2006.

SOLUCIONES

Problema 1

Pregunta 1.1:

```
    public GestorDeMultas() {
        tabla = new Registro[MAXIMO];
    }
```

Pregunta 1.2:

```
  public void entrada(Registro registro) {
    int mas_antiguo = 0;

    for (int i = 0; i < tabla.length; i++) {
      if (tabla[i] == null) {
        tabla[i] = registro;
        return;
      }
      if (tabla[mas_antiguo] != null
          && tabla[i].getHora() < tabla[mas_antiguo].getHora()) {
        mas_antiguo = i;
      }
    }
    tabla[mas_antiguo] = registro;
  }
```

Pregunta 1.3:

```
  public void salida(Registro salida) {
    Registro entrada = null;

    for (int i = 0; i < tabla.length; i++) {
      if (tabla[i] != null
          && tabla[i].getMatricula().equals(salida.getMatricula())) {
        if (entrada == null
            || entrada.getHora() < tabla[i].getHora()) {
          entrada = tabla[i];
        }
        tabla[i] = null;
      }
    }
    if (entrada != null) {
      double distancia =
        salida.getKilometro() - entrada.getKilometro();
      double tiempo = salida.getHora() - entrada.getHora();
      double velocidad = distancia / tiempo;

      if (velocidad > MAX_VEL) {
        System.out.print("Multa: " + salida.getMatricula());
        System.out.print(" Velocidad " + velocidad);
        System.out.print(" entre " + entrada.getHora());
        System.out.print(" y " + salida.getHora());
        System.out.print(" horas,");
        System.out.print(" entre Kms " + entrada.getKilometro());
        System.out.print(" y " + salida.getKilometro());
        System.out.println();
      }
    }
  }
```

Problema 2

Pregunta 2.1:

```
interface Onda {
    public double getLongitud();
}
```

Pregunta 2.2:

```
interface Corpusculo {
    public double getMasa();
    public double getVelocidad();
}
```

Pregunta 2.3:

```
public class Particula implements Onda, Corpusculo {
    private double longitud = 0.0;
    private double masa = 0.0;
    private double velocidad = 0.0;

    public Particula(double l, double m, double v) {
        longitud = l;
        masa = m;
        velocidad = v;
    }

    public double getMasa() {
        return masa;
    }

    public double getVelocidad() {
        return masa;
    }

    public double getLongitud() {
        return longitud;
    }
}
```

Pregunta 2.4:

```
import java.util.*;

class PruebaParticula {
```

```
    public static void ImprimeOndas(List<Onda> lista) {
        for (Onda onda: lista) {
            System.out.println(onda.getLongitud());
        }
    }
```

```
    public static void ImprimeOndas(Vector<Onda> lista) {
        for (Enumeration<Onda> lenum = lista.elements();
             lenum.hasMoreElements(); ) {
            Onda onda = lenum.nextElement();

            System.out.println(onda.getLongitud());
        }
    }
```

```
public static void main(String[] args) {
        List <Onda> lista = new ArrayList <Onda>();

        lista.add(new Particula(1, 10, 100));
        lista.add(new Particula(2, 20, 200));
        lista.add(new Particula(3, 30, 300));
        lista.add(new Particula(4, 40, 400));
        lista.add(new Particula(5, 50, 500));

        ImprimeOndas(lista);

        Vector<Onda> vector = new Vector<Onda>();

        vector.addElement(new Particula(1, 10, 100));
        vector.addElement(new Particula(2, 20, 200));
        vector.addElement(new Particula(3, 30, 300));
        vector.addElement(new Particula(4, 40, 400));
        vector.addElement(new Particula(5, 50, 500));

        ImprimeOndas(vector);
    }
}
```

Problema 3

Pregunta 3.1:

```
    private List<Punto> puntos;
    public Trayectoria() {
        puntos = new ArrayList<Punto>();
    }
```

```
    private Vector<Punto> puntos;
    public Trayectoria() {
        puntos = new Vector<Punto> ();
    }
```

Pregunta 3.2:

```
    public void addPunto(Punto p) {
        Punto ultimo = null;
        for (Punto aux : puntos) {
            ultimo = aux;
        }
        if ((ultimo != null) &&
            (ultimo.distancia(p) < DISTANCIA_MIN)) {
            return;
        }
        Punto nuevoPunto = new Punto(p.getX(), p.getY(), p.getZ());
        if ((ultimo != null) &&
            Math.abs(ultimo.getZ() - nuevoPunto.getZ()) > ALTURA_MAX) {
            nuevoPunto.setZ(ultimo.getZ());
        }
        puntos.add(nuevoPunto);
    }
```

Pregunta 3.3:

```
    public double getPendiente(Punto p) {
        double alturas = p.getZ() - this.getZ();
        double dx = p.getX() - this.getX();
        double dy = p.getY() - this.getY();
        double distanciasxy = Math.sqrt(dx * dx + dy * dy);
        return Math.abs(Math.atan2(alturas, distanciasxy));
    }
```

Pregunta 3.4:

```
    public double getPendienteMaxima() throws Exception {
        double maxima = 0.0;
        if (puntos.size() < 2) {
            throw new Exception("No hay puntos para el cálculo");
        }
        Punto anterior = null;
        for (Punto p: puntos) {
            if (anterior != null) {
                double pendiente = anterior.getPendiente(p);

                if (pendiente > maxima) {
                    maxima = pendiente;
                }
            }
            anterior = p;
        }
        return maxima;
    }
```

Fundamentos de Programación

9 de febrero de 2008

Patrón

Dadas estas clases java:

```
 1 public abstract class Pieza {
 2   private int fila, columna;
 3
 4   public Pieza(int fila, int columna) {
 5     this.fila= fila;
 6     int columna= columna;
 7   }
 8
 9   public int getFila() { return fila; }
10
11   public int getColumna() { return columna; }
12
13   public void setPosicion(int f, int c) {
14     fila= f;
15     this.columna= c;
16   }
17
18   /**
19    * Averigua si la pieza puede moverse a una cierta posicin.
20    * @param fila a la que queremos movernos.
21    * @param columna a la que queremos movernos.
22    * @return true si puede moverse ah.
23    */
24   public abstract boolean sePuede(int fila, int columna);
25 }
```

```
 1 public class Rey extends Pieza {
 2
 3   public Rey(int fila, int columna) {
 4     super(fila, columna);
 5   }
 6
 7   public boolean sePuede(int f, int c) {
 8     if (f < 0 || f > 7 || c < 0 || c > 7)
 9       throw new Exception("fuera de rango");
10     int dx= Math.abs(f - fila);
11     int dy= Math.abs(c - getColumna());
12     return dx = 1 || dy = 1;
13   }
14 }
```

Indique si son ciertas o falsas las siguientes afirmaciones:

Pregunta 1 [1 punto]
La línea 5 de Pieza es correcta.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 2 [1 punto]
La línea 6 de Pieza es correcta.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 3 [1 punto]
La línea 14 de Pieza es correcta.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 4 [1 punto]
La línea 15 de Pieza es correcta.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 5 [1 punto]
La línea 4 de Rey es correcta.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 6 [1 punto]

La línea 9 de Rey es correcta.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 7 [1 punto]
La línea 10 de Rey es correcta.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 8 [1 punto]
La línea 11 de Rey es correcta.

1 [+1,00] cierto
2 [-1,00] falso

Dadas las siguientes definiciones y sentencias java:

```
class A implements I {
 void m() { }
}
interface I {
  void m();
}
```

```
1: I i = new A();
2: I[] i2 = new I[2];
3: i2[0] = new I();
4: i2[0] = i;
5: i2[1].m();
6: i.m();
```

Indique si son ciertas o falsas las siguientes afirmaciones.

Pregunta 9 [1 punto]
La línea 1 compila y ejecuta correctamente.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 10 [1 punto]
La línea 2 compila y ejecuta correctamente.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 11 [1 punto]
La línea 3 compila y ejecuta correctamente.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 12 [1 punto]
La línea 4 compila y ejecuta correctamente.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 13 [1 punto]
La línea 5 compila y ejecuta correctamente.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 14 [1 punto]
La línea 6 compila y ejecuta correctamente.

1 [+1,00] cierto
2 [-1,00] falso

Dado este interfaz y las dos clases,

```
interface I1 {        class A {            class B extends A
   void m1(A a);         public int i;        implements I1 {
}                     }                    }
```

indique qué asignaciones se pueden hacer, y cuales no de las siguientes:

```
1: public void m(I1 p1, A p2, B p3) {
2:   p1 = p2;
3:   p2 = p3;
4:   p3 = p1;
5: }
```

Pregunta 15 [1 punto]
La línea 1 compila y ejecuta correctamente.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 16 [1 punto]
La línea 2 compila y ejecuta correctamente.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 17 [1 punto]
La línea 3 compila y ejecuta correctamente.

1 [+1,00] cierto
2 [-1,00] falso

Pregunta 18 [1 punto]
La línea 4 compila y ejecuta correctamente.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 19 [1 punto]
Una variable NO marcada como *static* se puede usar en métodos marcados como *static*.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 20 [1 punto]
¿Qué método se ejecuta en la línea 3?

```
1:  class Uno {                    7:  class Dos extends Uno {
2:    public void m(Uno a) {       8:    public void m(Uno a) {
3:      n(a);                      9:      n(a);
4:    }                           10:    }
5:    public void n(Uno a) {}     11:    public void n(Uno a) {}
6:  }                             12:  }
```

1 [+1,00] Faltan datos para responder.
2 [-0,33] Al método "n()" de la línea 5.
3 [-0,33] Al método "n()" de la línea 11.
4 [-0,33] El código es erróneo y ni compila.

Pregunta 21 [1 punto]
Una clase puede heredar de varias clases.

1 [+1,00] falso
2 [-1,00] cierto

Pregunta 22 [1 punto]

Indique qué escribe el siguiente fragmento de programa:

```
int x = 0;
try {
   System.out.print("1");
   if (x <= 0) throw new IllegalArgumentExceptionException();
   System.out.print("2");
} catch (Exception e) {
   System.out.print("3");
   return;
} finally {
   System.out.print("4");
}
```

1 [+1,00] 1 3 4
2 [-0,50] 1 4
3 [-0,50] 1

Pregunta 23 [1 punto]

Dada la siguiente clase Punto:

```
class Punto {
   private double x, y;
   Punto(double x, double y) { this.x = x; this.y = y; }
   public boolean equals(Punto p) {
      return this.x == p.x && this.y == p.y;
   }
}
```

Indique qué se imprime al ejecutar:

```
Punto p = new Punto(1, 2);
Punto q = new Punto(1, 2);
System.out.print(p.equals(q))
```

1 [+1,00] true
2 [-1,00] false

Pregunta 24 [1 punto]

¿Qué resulta de la llamada "recursivo(2)" usando el siguiente método?

```
public int recursivo(int p) {
    if (p < 1)
       return 1;
    else
       return p * recursivo(p--);
}
```

1 [+1,00] Es un bucle infinito que provoca un error de ejecución por falta de memoria.
2 [-0,50] Calcula el factorial del numero p.
3 [-0,50] Devuelve siempre 1.

Pregunta 25 [1 punto]

Dadas las siguientes clases:

```
 1:  interface A {
 2:    void a1();
 3:  }
 4:  interface B {
 5:    void b1();
 6:  }
 7:  class C implements A, B {
 8:    void a1() { }
 9:    void b1() { }
10:  }
```

Qué pasará si ...

```
21:  B x = new C();
22:  x.b1();
```

1 [+1,00] Se ejecuta el método b1() de la línea 9.
2 [-0,50] Se ejecuta el método b1() de la línea 5.

3 [-0,50] Error de compilación en la declaración de C.

Pregunta 26 [1 punto]

¿Qué resulta de la ejecución del siguiente código?

```
int A []= new int[100];

for(int i : A) {
  i = 7;
}
```

1 [+1,00] Se genera un array de 100 números enteros inicializados a 0
2 [-0,50] Se genera un array de 100 números enteros inicializados a 7
3 [-0,50] Error de compilación, no se puede usar así el "*for*".

Pregunta 27 [1 punto]

Dado el siguiente método:

```
String p (int c) {
    if (c > 0) {
        return c + "" + p (c-1);
    }
    return c + "";
}
```

Indique cual de las siguientes afirmaciones es FALSA:

1 [-0,50] La ejecución de p(...) puede no acabar nunca.
2 [-0,50] La ejecución de p(3) devuelve 3210.
3 [+1,00] La ejecución de p(3) devuelve 6.

Pregunta 28 [1 punto]

Indique qué imprime por pantalla al ejecutar *p('C')*

```
char p (char c) {
    if (c > 'A') {
        System.out.print(c + " " + p ((char)(c-1)) );
    }
    return c;
}
```

1 [+1,00] B AC B
2 [-0,50] C B A
3 [-0,50] A B C

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN - FEBRERO 2008

Normas de examen:

- Con libros y apuntes.
- Duración: 2 horas.
- Responda a cada problema en hojas separadas
- No se contestará ninguna pregunta durante el examen.

Problema 1 (3 puntos)

El desarrollo en serie de Taylor de la función cos (x) es

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

Nótese que cada término puede calcularse en función del anterior, según:

$$a_n = -a_{n-1} \cdot \frac{x^2}{2n \cdot (2n-1)} \text{ con } a_0=1$$

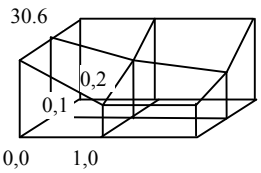
La suma de los n primeros términos calculará el valor del coseno con una precisión determinada "p" cuando $|a_n| < p$.

Se pide codificar en Java un método que calcule de forma iterativa el cos (x) con una cierta precisión p.
La cabecera de dicho método debe ser:

```
public static double calculaCoseno(double x, double p) { . . . }
```

Problema 2 (4 puntos)

Se va a construir una clase para manejar mapas, parte de la cual se proporciona hecha. Un mapa tiene un *array* bidimensional rectangular de números reales, donde el valor de cada casilla representa la altura sobre el nivel del mar de un cierto punto. Así, el *array* cubre una zona rectangular de puntos, con distancia 1 en ejes x e y, y con el punto de coordenadas 0,0 como el primero.



La figura muestra un ejemplo de mapa de 3x3, donde la altura máxima es 30.6.

```
class Mapa {
    private double [][] mapa;

    public Mapa (double [][] mapa) {
        this.mapa = mapa;
    }
    public double alturaMedia (int x, int y) { //…
    }
    public double volumen(){ // …
    }
    public Mapa recortar (double altura) { // …
    }
}
```

1. Escriba el método alturaMedia, para calcular la media de las alturas de los cuatro puntos correspondientes a las coordenadas (x, y), (x+1, y), (x, y+1), (x+1, y+1). Tenga cuidado de no salirse del mapa -en este caso el método devuelve 0.

```
double alturaMedia (int x, int y) { . . . }
```

2. Escriba el método volumen, que devuelve el volumen del mapa. Para simplificar, suponga que se calcula como la suma de los volúmenes de cada uno de los prismas cuadrados cuya base está formada por los cuatro vértices correspondientes a las casillas contiguas y cuya altura es la media calculada con el método anterior.

```
double volumen () { . . . }
```

3. Escriba el cuerpo del método recortar, que devuelve un Mapa construido mediante un mapa que es copia del original, pero la altura máxima es el valor del parámetro altura.

```
Mapa recortar (double altura) { . . . }
```

Problema 3 (3 puntos)

Se definen los tipos Motorizado, Combustible y Vehiculo:

```
interface Motorizado {
  double consumo();
  Combustible tipoCombustible ();
}

enum Combustible   {Gasolina, Gasoil}

class Vehículo {
   private String nombre;
   private int nPasajeros;
   ...
}
```

Se pide

1. Codifique la clase VehiculoAMotor que extiende la clase Vehículo e implementa la interface Motorizado. Proporcione el constructor adecuado a las clase según las definiciones.
2. Codifique la clase Tractor como una extensión de la Clase VehículoAMotor. Todos los tractores son de Gasoil por definición.
3. codifique el método:

```
double consumoMedio  (List <Motorizado> lm, Combustible  c) {
. . .
}
```

   que calcula la media de los consumos de todos los Motorizados en la lista lm que utilizan combustible c. Este método estaría en una clase diferente de las anteriores.

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN - FEBRERO 2008

Problema 1 (3 puntos)

El desarrollo en serie de Taylor de la función cos (x) es

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

Nótese que cada término puede calcularse en función del anterior, según:

$$a_n = -a_{n-1} \cdot \frac{x^2}{2n \cdot (2n-1)} \text{ con } a_0=1$$

La suma de los n primeros términos calculará el valor del coseno con una precisión determinada "p" cuando $|a_n| < p$.

Se pide codificar en Java un método que calcule de forma iterativa el cos (x) con una cierta precisión p.
La cabecera de dicho método debe ser:

```
public static double calculaCoseno(double x, double p) { . . . }
```

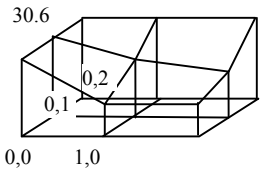
```
public static double calculaCoseno(double x, double p)
{
     double res= 1;
     double term= 1;
     int n= 1;

     do {
        term= - term * x * x / (2*n) / (2*n-1);
        res += term;
        n ++;
     } while (Math.abs(term) > p);

     return res;
}
```

Problema 2 (4 puntos)

Se va a construir una clase para manejar mapas, parte de la cual se proporciona hecha. Un mapa tiene un *array* bidimensional rectangular de números reales, donde el valor de cada casilla representa la altura sobre el nivel del mar de un cierto punto. Así, el *array* cubre una zona rectangular de puntos, con distancia 1 en ejes x e y, y con el punto de coordenadas 0,0 como el primero.



La figura muestra un ejemplo de mapa de 3x3, donde la altura máxima es 30.6.

```
class Mapa {
    private double [][] mapa;

    public Mapa (double [][] mapa) {
        this.mapa = mapa;
    }
    public double alturaMedia (int x, int y) { //…
    }
    public double volumen(){ // …
    }
    public Mapa recortar (double altura) { // …
    }
}
```

1. Escriba el método alturaMedia, para calcular la media de las alturas de los cuatro puntos correspondientes a las coordenadas (x, y), (x+1, y), (x, y+1), (x+1, y+1). Tenga cuidado de no salirse del mapa -en este caso el método devuelve 0.

   ```
   double alturaMedia (int x, int y) { . . . }
   ```

2. Escriba el método volumen, que devuelve el volumen del mapa. Para simplificar, suponga que se calcula como la suma de los volúmenes de cada uno de los prismas cuadrados cuya base está formada por los cuatro vértices correspondientes a las casillas contiguas y cuya altura es la media calculada con el método anterior.

   ```
   double volumen () { . . . }
   ```

3. Escriba el cuerpo del método recortar, que devuelve un Mapa construido mediante un mapa que es copia del original, pero la altura máxima es el valor del parámetro altura.

   ```
   Mapa recortar (double altura) { . . . }
   ```

```
    double alturaMedia(int x, int y) {
        if ((x < 0) || (x >= (mapa.length - 1)) ||
            (y < 0) || (y >= (mapa[0].length - 1))) {
            return 0.0;
        }

        return (mapa[x][y] + mapa[x + 1][y] +
                mapa[x][y + 1] + mapa[x + 1][y + 1]) / 4.0;
    }
```

```
    double volumen() {
        double volumen = 0.0;

        for (int i = 0; i < mapa.length; i++)
            for (int j = 0; j < mapa[0].length; j++)
                volumen += alturaMedia(i, j);

        return volumen;
    }
```

```
    Mapa recortar(double altura) {
        double[][] m2 = new double[mapa.length][mapa[0].length];

        for (int i = 0; i < mapa.length; i++)
            for (int j = 0; j < mapa[0].length; j++)
                if (mapa[i][j] > altura) {
                    m2[i][j] = altura;
                } else {
                    m2[i][j] = mapa[i][j];
                }

        return new Mapa(m2);
    }
```

Problema 3 (3 puntos)

Se definen los tipos Motorizado, Combustible y Vehiculo:

```
interface Motorizado {
  double consumo();
  Combustible tipoCombustible ();
}

enum Combustible   {Gasolina, Gasoil}

class Vehículo {
   private String nombre;
   private int nPasajeros;
   ...
}
```

Se pide

1. Codifique la clase VehiculoAMotor que extiende la clase Vehículo e implementa la interface Motorizado. Proporcione el constructor adecuado a la clase según las definiciones.
2. Codifique la clase Tractor como una extensión de la Clase VehículoAMotor. Todos los tractores son de Gasoil por definición.
3. codifique el método:

   ```
   double consumoMedio  (List <Motorizado> lm, Combustible  c) {
   . . .
   }
   ```

   que calcula la media de los consumos de todos los Motorizados en la lista lm que utilizan combustible c. Este método estaría en una clase diferente de las anteriores.

```
public class VehiculoAMotor
    extends Vehiculo
    implements Motorizado {
  double consumo;          // en litros por cada 100 km
  Combustible combustible; // tipo de combustible

  public VehiculoAMotor (String n, int np, double con, Combustible comb) {
       super (n, np);
       consumo = con;
       combustible = comb;
  }

  public double consumo () { return consumo; }

  public Combustible tipoCombustible () { return combustible; }
}
```

```
public class Tractor
      extends VehiculoAMotor {

   public Tractor (String nombre, int pasajeros, double consumo) {
      super (nombre, pasajeros, consumo, Combustible.Gasoil);
   }
}
```

```
double consumoMedio (List<Motorizado> lm, Combustible c) {
    double motorizados = 0.0;
    double consumomotorizados = 0.0;
    double res;
    for (Motorizado m : lm) {
        if (m.tipoCombustible () == c) {
            motorizados ++;
            consumomotorizados += m.consumo();
      }
    }
    if (motorizados == 0)
        return 0.0;
    return consumomotorizados / motorizados;
}
```

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN - SEPTIEMBRE 2008

Normas de examen:

- Con libros y apuntes.
- Duración: 2 horas y media (2h 30m).
- Responda a cada problema en hojas separadas
- No se contestará ninguna pregunta durante el examen.

Problema 1 (2,5 puntos)

Se pide especificar e implementar un método que tenga dos parámetros cuyo tipo es el interfaz *List* de la biblioteca Java, y que devuelva un *boolean*. Los elementos de los dos parámetros son todos *String*. El método se encargará de comparar las dos listas, y comprobar que todos los string del primer parámetro tienen una copia igual en el segundo. El método devolverá *true* si todos los string tienen al menos una copia en el segundo conjunto, y *false* si alguno no está en el segundo conjunto. Si el primer parámetro es la lista vacía, el método devolverá *true*. Si alguno de los dos parámetros es *null* lanzará una *Exception*.

Para comparar dos objetos *String* podemos emplear el método *public boolean equals(Object anObject)* de la clase *String*. Este método devuelve *true,* si y solo si el parámetro no es *null* y es un objeto *String* que representa la misma secuencia de caracteres que el objeto sobre el que se ejecuta *equals*.

Problema 2 (2,5 puntos)

Queremos desarrollar un método que calcule la potencia enésima de un número real. La especificación del método será:

public static real elevar(real base, int exponente)

donde *exponente* será mayor o igual a 0.

Para calcular la potencia usaremos el siguiente algoritmo:

1. si *exponente* es par entonces $base^{exponente} = (base^{exponente/2})^2$
2. si *exponente* es impar entonces $base^{exponente} = (base^{exponente-1}) * base$
3. si exponente = 0 entonces $base^{exponente} = 1$

Problema 3 (2,5 puntos)

Sea la siguiente clase Punto que permite gestionar puntos del plano

```
public class Punto {
  private double x;
  private double y;

  public Punto (double x, double y) {
    this.x = x;
    this.y = y;
  }

  public double getX () {
    return x;
  }

  public double getY () {
    return y;
  }

  public double distancia () {
    return Math.sqrt(x*x+y*y);
  }
}
```

Y sea la siguiente clase Datos, que contiene una lista de puntos del plano:

```
import java.util.*;

public class Datos {
  private ArrayList <Punto> datos;

  public Datos () {
    datos = new ArrayList <Punto> ();
  }

  public Datos (ArrayList <Punto> lista) {
    this.datos = lista;
  }

  public void add (Punto p) {
    datos.add(p);
  }
}
```

Se pide escribir un método *distanciaMedia ()*, de la clase *Datos*, que devuelva la distancia media de todos los puntos al origen del plano. El resultado es de tipo *double*. Si la lista está vacía, lance una *Exception*.

Problema 4 (2,5 puntos)

Partiendo de las mismas clases que en el Problema 3, se pide cambiar la clase *Datos* para que sea una clase instanciable que implementa el interfaz *DatosDeCirculo*:

```
interface DatosDeCirculo {
    Punto[] enCirculo (double r);
}
```

El método *enCirculo* devuelve un array (subconjunto) de todos los puntos de la lista original que se encuentran a una distancia menor que *r* del origen.

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN - SEPTIEMBRE 2008

Soluciones

Problema 1 (2,5 puntos)

```
import java.util.List;

public class ComparaListas {

    public static boolean listaContenida(List<String> subconjunto,
            List<String> superconjunto) throws Exception {
        if (subconjunto == null || superconjunto == null)
            throw new Exception("subconjunto o superjunto son null");
        for (String elemento: subconjunto) {
            boolean encontrado = false;
            for (String elementoSuper: superconjunto) {
                if (elemento == null)
                    throw new Exception("Un elemento es null");
                if (elemento.equals(elementoSuper)) {
                    encontrado=true;
                    break;
                }
            }
            if (!encontrado)
                return false;
        }
        return true;
    }
}
```

Problema 2 (2,5 puntos)

```
public class Potencias {

    public static double elevar(double base, int exponente) {
        double resultado = 0;
        if (exponente == 0)
            resultado = 1;
        else if (exponente % 2 == 0) {
            double exponenteDiv2 = elevar(base, exponente / 2);
            resultado = exponenteDiv2 * exponenteDiv2;
        } else {
            double exponenteMenos1 = elevar(base, exponente - 1);
            resultado = exponenteMenos1 * base;
        }
        return resultado;
    }
}
```

Problema 3 (2,5 puntos)

```
    public double distanciaMedia () throws Exception {
        double distancia = 0;
        int numeroPuntos = 0;
        if (datos.size() == 0) {
            throw new Exception ();
        }
        for (Punto p: datos) {
            distancia = distancia + p.distancia();
            numeroPuntos++;
        }
        return distancia/numeroPuntos;
    }
```

Problema 4 (2,5 puntos)

```
public class Datos implements DatosDeCirculo {

..

    public Punto[] enCirculo (double r) {
        ArrayList <Punto> datosTemp = new ArrayList <Punto> ();

        for (Punto p: datos) {
            if (p.distancia() < r) {
                datosTemp.add(p);
            }
        }

        int i = 0;
        Punto[] puntosEnCirculo = new Punto[datosTemp.size()];
        for (Punto p: datosTemp) {
            puntosEnCirculo[i++] = p;
        }

        return puntosEnCirculo;
    }
```

```
public class Datos implements DatosDeCirculo {

..

    public Punto[] enCirculo (double r) {
        int i = 0;
        Punto[] puntosEnCirculo = new Punto[datos.size()];
        for (Punto p: datos) {
            if (p.distancia() < r) {
                puntosEnCirculo[i++] = p;
            }
        }
        return puntosEnCirculo;
    }
```

Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid

FUNDAMENTOS DE PROGRAMACIÓN - FEBRERO 2009

Normas del examen:

- Con libros y apuntes.
- Duración: 2 horas y media (2h 30m).
- Responda a cada problema en hojas separadas
- No se contestará ninguna pregunta durante el examen.

Fechas:

- notas provisionales: 20.2.2009
- revisión: 25.2.2009
- notas finales: 9.3.2009

Problema 1 (3 puntos)

Se va a construir un programa para comprobar el grado de afinidad (las posibilidades de amistad) entre usuarios. Cada usuario se representará mediante un objeto, que contiene su dirección de correo electrónico y un *array* de 20 números reales entre 0.0 y 1.0, que indican el interés del usuario respecto a 20 aficiones diferentes.

Pregunta 1. Escriba la clase *Usuario*, con los atributos mencionados, incluya un constructor cuyo único parámetro sea la dirección de correo electrónico (de tipo *String*), un método accesor (*getter*) para la dirección, y dos métodos uno para acceder (*getter*) y otro para modificar (*setter*) el valor de cada casilla del *array* de aficiones, dado su índice. Estos dos métodos tienen las cabeceras que se indican a continuación, y lanzan una excepción si el valor del índice que se pasa como parámetro no corresponde a ninguna casilla del *array* de aficiones:

```
public void setAficion (double aficion, int indice) throws Exception

public double getAficion (int indice) throws Exception
```

Pregunta 2. Para calcular el grado de afinidad entre dos usuarios se usa la aproximación del coseno (coseno del ángulo que formarían los dos arrays de aficiones considerados como vectores en un espacio de 20 dimensiones), variando entre 0 (coincidencia total o afinidad máxima) y 1 (afinidad mínima).

Siendo x_i el valor de interés en la afición i para el usuario x, e y_i el correspondiente para el otro usuario, el grado de afinidad se calcula así:

$$\cos(\vec{x},\vec{y}) = \frac{\sum_{i=0..19} x_i y_i}{\sqrt{\sum_{i=0..19} x_i^2 \sum_{i=0..19} y_i^2}}$$

Escriba el método que calcula el grado de afinidad entre dos usuarios en la clase *Usuario*. Lanza excepciones si el denominador es cero o si alguna llamada a otro método las lanzase. La cabecera es:

```
public double afinidad (Usuario usuario) throws Exception
```

Pregunta 3. Escriba un método para calcular el usuario medio del array de usuarios que se pasa como parámetro. El usuario medio es uno con dirección de correo vacía, en el que cada casilla de afición contiene el valor medio de las casillas de aficiones correspondientes de los usuarios que se pasan en el parámetro. Si no hay usuarios, las casillas de afición son cero. La cabecera es:

```
public static Usuario getUsuarioMedio (Usuario[] usuarios)
```

Problema 2 (4 puntos)

Queremos definir una nueva colección de datos que será una lista de *String* ordenada por prioridades. En esa lista varios string pueden tener la misma prioridad, y se insertan y extraen según sea su prioridad. La prioridad es un número natural mayor o igual a 0, y menor que un nivel máximo que se fija en la construcción. Lo que no podemos fijar es el número máximo de strings que tendremos para cada prioridad. Un *array* puede representar todas las prioridades, y un elemento del *array* permitir referenciar todos los elementos de esa prioridad (el elemento del *array* puede ser una lista que contiene los elementos de esa prioridad). Los elementos de igual prioridad se ordenan según su orden de llegada, como se hace en el tipo *List*. La lista con prioridad tiene el siguiente interfaz:

```
interface ListaConPrioridades {
  boolean add(int prioridad, String valor)
      throws IllegalArgumentException, NullPointerException;

  boolean remove(int prioridad, String valor)
      throws IllegalArgumentException, NullPointerException;

  List<String> nivelPrioridad(int prioridad)
      throws IllegalArgumentException;

  int maxPrioridad();
}
```

`boolean add(int prioridad, String valor)`

Añadimos un string "valor" como último elemento de prioridad "prioridad". Devolvemos true si se ha podido insertar, sino false. Se puede lanzar la excepción *IllegalArgumentException* cuando la prioridad "prioridad" no existe o *NullPointerException* cuando "valor" es *null*

`boolean remove(int prioridad, String valor)`

Retiramos la primera ocurrencia del string "valor" en el nivel de prioridad "prioridad". Devolvemos true si se ha podido borrar; si no, false. Se puede levantar la excepción *IllegalArgumentException* cuando la prioridad "prioridad" no existe o *NullPointerException* cuando "valor" es *null*

`List<String> nivelPrioridad(int prioridad)`

Devolvemos todos los string de nivel de prioridad "prioridad". Se puede levantar la excepción *IllegalArgumentException* cuando la prioridad "prioridad" no existe

`int maxPrioridad()`

Devolvemos el nivel de prioridad mayor.

Se pide crear la clase *ListaConPrioridadBasadaEnList* que implementa el interfaz *ListaConPrioridades*, basándose en datos de tipo *List* (y cualquier clase que lo implemente), *arrays* y tipos primitivos *int* y *boolean*. Del interfaz *List* es importante tener en cuenta los métodos:

```
public boolean add(Object e); // inserta un nuevo elemento

public boolean remove(Object e); // extrae y retira un elemento
```

Pregunta 1. Definir los campos de esa clase y el constructor:

```
public ListaConPrioridadBasadaEnList(int maxPrioridad)
```

Que construye la lista con un rango de prioridades de 0 a "maxPrioridad-1" ambas inclusive.

Pregunta 2. Implementar los métodos *add* y *remove* del interfaz *ListaConPrioridades*.

Pregunta 3. Implementar el método *nivelPrioridad* del interfaz *ListaConPrioridades*. Si no hay ningún string de la prioridad que indica el parámetro, se devolverá una lista vacía.

Problema 3 (3 puntos)

Se desea definir una jerarquía de clases para gestionar una biblioteca. La definición de esta estructura se basará en las clases *Publicacion* y *Articulo*.

La clase *Publicacion* tiene como información privada el título (*String*) de la publicación y el número de páginas (número entero). A partir de esta clase (como clases que extienden a la clase *Publicacion*), se definen las clases *Libro* y *Revista*.

La clase *Libro* tiene como información privada el autor (*String*) y el ISBN (número entero), mientras que la clase *Revista* tiene una lista de artículos y un ISSN (número entero). La lista de *Artículos* dentro de la revista se define como una *List<Articulo>*.

La clase *Artículo* está Implementada como sigue:

```
public class Articulo {
   private String titulo;
   private String autor;
   private int np;

   public Articulo (String t, String a, int n) {
     titulo = t;     autor = a;     np = n;
   }

   public void   setTitulo  (String t) { titulo = t;   }
   public void   setAutor   (String a) { autor  = a;   }
   public void   setNPaginas(int n)    { np     = n;   }
   public String getTitulo  ()         { return titulo;}
   public String getAutor   ()         { return autor;   }
   public int    getNPaginas()         { return np;  }
 }
```

El número de páginas de la *Revista* será igual a la suma del número de páginas de los artículos contenidos más 4 (pastas).

Se muestra un ejemplo de uso de estas clases.

```
import java.util.*;

public class TestPublicacion {
  public static void main (String [] args){
      Articulo p1 = new Articulo ("Artículo 1", "Tomas", 10);
      Articulo p2 = new Articulo ("Artículo 2", "Enrique", 15);
      Articulo p3 = new Articulo ("Artículo 3", "Juan", 5);

      List <Articulo> arts = new ArrayList <Articulo> ();
      arts.add (p1); arts.add (p2); arts.add (p3);

      Revista r = new Revista ("Articulos Enero 2009", arts, 1111);
      System.out.println ("Revista=> " + r);
  }
}
```

Pregunta 1. Codifique la clase *Publicacion*

```
public class Publicacion { . . . }
```

Pregunta 2. Codifique la clase *Libro*

```
public class Libro extends Publicacion { . . . }
```

Pregunta 3. Codifique la clase *Revista*. Recuerde que el número de páginas de la *Revista* se calcula como la suma del número de páginas de los artículos que la componen + 4.

```
public class Revista extends Publicacion { . . . }
```

```
/**
 * Examen 9-Feb-2009. Problema 1.
 *
 * @author Fundamentos de Programación
 * @version 21-feb-2009
 */
public class Usuario {
    private static final int NUMERO_AFICIONES = 20;

    private String email;
    private double[] aficiones;

    public Usuario(String email) {
        this.email = email;
        aficiones = new double[NUMERO_AFICIONES];
    }

    public String getEmail() {
        return this.email;
    }

    public void setAficion(double aficion, int indice) throws Exception {
        if ((indice < 0) || (indice >= aficiones.length))
            throw new Exception();
        aficiones[indice] = aficion;
    }

    public double getAficion(int indice) throws Exception {
        if ((indice < 0) || (indice >= aficiones.length))
            throw new Exception();
        return aficiones[indice];
    }

    public double afinidad(Usuario usuario) throws Exception {
        double num = 0.0;
        double d1 = 0.0;
        double d2 = 0.0;
        for (int i = 0; i < this.aficiones.length; i++) {
            num += usuario.getAficion(i) * this.getAficion(i);
            d1 += usuario.getAficion(i) * usuario.getAficion(i);
            d2 += this.getAficion(i) * this.getAficion(i);
        }
        if ((d1 == 0.0) || (d2 == 0.0))
            throw new Exception();
        return num / Math.sqrt(d1 * d2);
    }

    public static Usuario getUsuarioMedio(Usuario[] usuarios) {
        Usuario usuarioMedio = new Usuario("");
        for (int i = 0; i < NUMERO_AFICIONES; i++) {
            try {
                double d = 0.0;
                for (int j = 0; j < usuarios.length; j++) {
                    d += usuarios[j].getAficion(i);
```

```
                }
                if (usuarios.length > 0)
                    usuarioMedio.setAficion(d / usuarios.length, i);
            } catch (Exception e) {
            }
        }
        return usuarioMedio;
    }
}
```

```
import java.util.List;

/**
 * Examen 9-Feb-2009. Enunciado 2.
 *
 * @author Fundamentos de Programación
 * @version 21-feb-2009
 */
public interface ListaConPrioridades {
    boolean add(int prioridad, String valor)
            throws IllegalArgumentException, NullPointerException;

    boolean remove(int prioridad, String valor)
            throws IllegalArgumentException, NullPointerException;

    List<String> nivelPrioridad(int prioridad)
            throws IllegalArgumentException;

    int maxPrioridad();
}
```

```
import java.util.ArrayList;
import java.util.List;

/**
 * Examen 9-Feb-2009. Problema 2.
 *
 * @author Fundamentos de Programación
 * @version 21-feb-2009
 */
public class ListaConPrioridadBasadaEnList
        implements ListaConPrioridades {
    private List<String>[] elementos;
    int prioridades = 0;

    public ListaConPrioridadBasadaEnList(int prioridades) {
        this.prioridades = prioridades;
        elementos = new List[prioridades];
        for (int i = 0; i < prioridades; i++)
            elementos[i] = new ArrayList<String>();
    }

    /**
     * Añadimos un elemento “valor” como ultimo elemento de prioridad “prioridad”.
     *
     * @return true si se ha podido insertar, sino false.
     * @throws IllegalArgumentException cuando la prioridad “prioridad” no existe.
     * @throws NullPointerException     cuando el valor “valor” es null.
     */
    public boolean add(int prioridad, String valor)
            throws IllegalArgumentException, NullPointerException {
        if (prioridad >= prioridades)
            throw new IllegalArgumentException();
        if (valor == null)
            throw new NullPointerException();
        return elementos[prioridad].add(valor);
    }

    /**
     * Retiramos la primera ocurrencia del elemento “valor” en el nivel de prioridad “prioridad”.
     *
     * @return true si se ha podido borrar, sino false.
     * @throws IllegalArgumentException cuando la prioridad “prioridad” no existe.
     * @throws NullPointerException     cuando el valor “valor” es null.
     */
    public boolean remove(int prioridad, String valor)
            throws IllegalArgumentException, NullPointerException {
        if (prioridad >= prioridades)
            throw new IllegalArgumentException();
        if (valor == null)
            throw new NullPointerException();
        return elementos[prioridad].remove(valor);
    }

    /**
```

```
     * @return devolvemos todos los elementos de nivel de prioridad “prioridad”.
     * @throws IllegalArgumentException cuando la prioridad “prioridad” no existe.
     */
    public List<String> nivelPrioridad(int prioridad)
            throws IllegalArgumentException {
        if (prioridad >= prioridades)
            throw new IllegalArgumentException();
        return elementos[prioridad];
    }

    /**
     * @return el nivel de prioridad mayor.
     */
    public int maxPrioridad() {
        return prioridades;
    }
}
```

```
/**
 * Examen 9-Feb-2009. Problema 3.
 *
 * @author Fundamentos de Programación
 * @version 21-feb-2009
 */
public class Articulo {
    private String titulo;
    private String autor;
    private int nPaginas;

    public Articulo(String titulo, String autor, int nPaginas) {
        this.titulo = titulo;
        this.autor = autor;
        this.nPaginas = nPaginas;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public void setNPaginas(int nPaginas) {
        this.nPaginas = nPaginas;
    }

    public String getTitulo() {
        return titulo;
    }

    public String getAutor() {
        return autor;
    }

    public int getNPaginas() {
        return nPaginas;
    }
}
```

```
/**
 * Examen 9-Feb-2009. Problema 3.
 *
 * @author Fundamentos de Programación
 * @version 21-feb-2009
 */
public class Publicacion {
    private String nombre;
    private int nPaginas;

    public Publicacion(String nombre, int nPaginas) {
        this.nombre = nombre;
        this.nPaginas = nPaginas;
    }

    public void setNPaginas(int nPaginas) {
        this.nPaginas = nPaginas;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getNPaginas() {
        return nPaginas;
    }

    public String getNombre() {
        return nombre;
    }
}
```

```
/**
 * Examen 9-Feb-2009. Problema 3.
 *
 * @author Fundamentos de Programación
 * @version 21-feb-2009
 */
public class Libro
        extends Publicacion {
    private String autor;
    private int isbn;

    public Libro(String nombre, int nPaginas, String autor, int isbn) {
        super(nombre, nPaginas);
        this.autor = autor;
        this.isbn = isbn;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public void setIsbn(int isbn) {
        this.isbn = isbn;
    }

    public String getAutor() {
        return autor;
    }

    public int getIsbn() {
        return isbn;
    }
}
```

```
import java.util.List;

/**
 * Examen 9-Feb-2009. Problema 3.
 *
 * @author Fundamentos de Programación
 * @version 21-feb-2009
 */
public class Revista
        extends Publicacion {
    private List<Articulo> articulos;
    private int issn;

    public Revista(String nombre, List<Articulo> articulos, int issn)
            throws Exception {
        super(nombre, cuenta(articulos));
        this.articulos = articulos;
        this.issn = issn;
    }

    private static int cuenta(List<Articulo> articulos) {
        int np = 0;
        for (Articulo articulo : articulos) {
            np += articulo.getNPaginas();
        }
        return np + 4;
    }

    public void setArticulos(List<Articulo> articulos) {
        this.articulos = articulos;
        setNPaginas(cuenta(articulos));
    }

    public void setIssn(int issn) {
        this.issn = issn;
    }

    public List<Articulo> getArticulos() {
        return articulos;
    }

    public int getIssn() {
        return issn;
    }
}
```