Programación Orientada a Objetos, 3er curso

Ejercicios resueltos

1. ¿Cómo es posible crear objetos de una clase cuyos constructores son todos privados?

   a) Definiendo un método estático público en la clase que cree un objeto de la clase y lo devuelva.
   b) Definiendo una subclase y declarando públicos los constructores heredados.
   c) Definiendo una superclase con constructores públicos.
   d) No es posible.

2. ¿Cuál de las siguientes afirmaciones es falsa en Java?

   a) Es posible definir arrays bidimensionales de la forma a[i][j], donde cada fila a[i] puede tener distinta longitud.
   b) No es posible declarar arrays con memoria estática.
   c) "Hola" es un array.
   d) Un array es un objeto.

3. Dado el siguiente programa:

```
class A {
   static int x[ ] = {1, 2};
   int y[ ];
   void f (int z[ ]) { z[1]+=2; }
   void g ( ) {
      A a = new A ( );
      a.x[0]++;
   }
}
class B {
   public static void main (String args[ ]) {
      A b = new A ( );
      b.y = b.x;
      b.f (b.y);
      b.g ( );
      System.out.println (b.x[0] + " " + b.x[1]);      (1)
      System.out.println (b.y[0] + " " + b.y[1]);      (2)
   }
}
```

   ¿Cuál es la salida de la línea (1)?

   a) 1 2
   b) 1 4
   c) 2 2
   d) 2 4

4. En el programa anterior, ¿cuál es la salida de la línea (2)?

   a) 1 2
   b) 1 4
   c) 2 2
   d) 2 4

5. ¿Cuál es la salida en pantalla del siguiente programa?

```
class A {
   static int n = 0;
   String f (A x) { return "AA" + (x.n++); }
   String f (B x) { return "AB" + (x.n--); }
}
```

```
class B extends A {
   String f (B x) { return "BB" + (x.n--); }
   public static void main (String args[]) {
          A a = new A ();
          A b = new B ();
          System.out.println (a.f (b));
          System.out.println (b.f (b));
          System.out.println (a.f ((B) b));
          System.out.println (b.f ((B) b));
   }
}
```

Salida:

```
AA0
AA1
AB2
BB1
```

6. El siguiente método ordena una lista de números decimales por el método de la burbuja:

```
static void ordenar (double lista[]) {                                    1
   for (int i = 0; i < lista.length; i++)                                 2
       for (int j = lista.length-1; j > i; j--)                           3
           if (lista[j] < lista[j-1]) intercambiar (lista, j, j-1);       4
}
```

   a) Generalizar la función ordenar para que ordene listas de cualquier tipo de datos sobre los que tenga sentido definir una relación de orden. Para ello, introducir una mínima modificación en las líneas 1 y 4, y definir las clases y/o interfaces adicionales que sean necesarias.

```
static void ordenar (Ordenable lista[]) {
   for (int i = 0; i < lista.length; i++)
       for (int j = lista.length-1; j > i; j--)
          if (lista[j].menor (lista[j-1])) intercambiar (lista, j, j-1);
}

interface Ordenable {
       boolean menor (Ordenable obj);
}
```

   b) Basándose en el diseño del apartado anterior, definir las clases Rectangulo, Circulo y Figura, de tal manera que sea posible ordenar listas de figuras por su área. Definir en estas clases todos los métodos y variables necesarios para ello, siempre al nivel más alto posible de la jerarquía de clases.

   Nota: para simplificar, se permite suponer que los lados de un `Rectangulo` son paralelos a los ejes de coordenadas.

```
abstract class Figura implements Ordenable {
       public boolean menor (Ordenable obj) {
              return obj instanceof Figura
                     && area () < ((Figura) obj) .area ();
       }
       abstract double area ();
}

class Rectangulo extends Figura {
       double left, top, width, height;
       public double area () { return width * height; }
}

class Circulo extends Figura {
       double centerX, centerY, radius;
       public double area () { return 2 * Math.PI * radius; }
}
```

7. a) Definir una clase `Conjunto` que contenga:
   - Un array de valores de cualquier tipo.
   - Un método `interseccion` que tome como argumento otro conjunto, y devuelva un nuevo conjunto con la intersección de los dos, es decir, los elementos de la primera lista que son `equal` a algún elemento de la segunda.

```
class Conjunto {
	Object elementos[];
	Conjunto (Vector elems) {
		elementos = new Object [elems.size ()];
		for (int i = 0; i < elementos.length; i++)
			elementos [i] = elems.elementAt (i);

	}
	Conjunto interseccion (Conjunto conj) {
		Vector inter = new Vector ();
		for (int i = 0; i < elementos.length; i++)
			for (int j = 0; j < conj.elementos.length; j++)
				if (elementos[i].equals (conj.elementos[j])) {
					inter.addElement (elementos[i]);
					break;
				}
		return new Conjunto (inter);
	}
}
```

   b) Definir una clase `Persona` con una variable `dni` y los métodos necesarios para que se pueda hacer la intersección de listas de personas con la clase anterior, considerando que dos personas son la misma cuando tienen el mismo DNI.

```
class Persona {
	String dni;
	public boolean equals (Object obj) {
		return (obj instanceof Persona) && dni.equals (((Persona) obj) .dni);
	}
}
```

   c) Definir un método elements en la clase Conjunto que devuelva una Enumeration para iterar sobre un conjunto. Indicación: Enumeration es una interface con dos métodos: nextElement y hasMoreElements.

```
class Conjunto {
	...
	Enumeration elements () {
		return new ConjEnumeration (this);
	}
}
class ConjEnumeration implements Enumeration {
	Conjunto conjunto;
	int actual = 0;
	ConjEnumeration (Conjunto conj) { conjunto = conj; }
	public Object nextElement () {
		return conjunto.elementos[actual++];
	}
	public boolean hasMoreElements () {
		return actual < conjunto.elementos.length;
	}
}
```

8. Indicar cuál es la salida del siguiente programa y explicar por qué.

```
class A {
    public static void main (String args[]) throws X {
        try { f (); throw new Z (); }
        catch (Y ex) { System.out.println ("Y" + ex); }
        catch (X ex) { System.out.println ("X" + ex); }
    }
```

```
    static f () throws X {
        try { throw new Y (); } catch (X ex) { g (); }
    }
    static g () throws X {
        try { throw new X (); } catch (Y ex) {}
    }
}

class X extends Exception {
    public String toString () { return "X"; }
}

class Y extends X {
    public String toString () { return "Y"; }
}

class Z extends Y {
    public String toString () { return "Z"; }
}
```

Solución...

9. Explica brevemente la diferencia entre un método synchronized y otro que no lo es.

   Dos threads no pueden ejecutar simultáneamente dos métodos synchronized de un mismo objeto. Los métodos no synchronized no tienen esta restricción (se puede simultanear su ejecución con la de otro método, sea éste synchronized o no).

10. Dada la siguiente clase:

```
class Contador extends Thread {
    private int n = 0;
    private boolean running = true;
    private boolean active = true;
    public void run () {
        while (running) {
            if (active) System.out.println (n++);
        }
    }
}
```

Añadir a la clase `Contador` los métodos `detener`, `reanudar`, `reiniciar` y `terminar` que hagan respectivamente que la cuenta hacia delante se detenga, se reanude donde se detuvo, vuelva a 0 (esté o no detenida), y termine definitivamente. Indicación: no utilizar métodos como `interrupt`, `suspend`, `resume`, `stop`, etc., heredados de la clase `Thread`.

Solución...

11. Dado el programa:

```
class Ventana extends Frame implements MouseMotionListener {
   Label text = new Label ();
   Ventana () {
          setBounds (0, 0, 200, 200);
          setLayout (null);                                                    1
          add (text);                                                          2
          text.setBounds (1, 30, 100, 20);
          addMouseMotionListener (this);
   }
   public void mouseDragged (MouseEvent ev) {
          text.setText ("(" + ev.getX () + "," + ev.getY () + ")");
//        text.setLocation (ev.getPoint ());                                   3
   }
   public void mouseMoved (MouseEvent ev) {}
   static public void main (String args[]) {
          new Ventana () .setVisible (true);
```

```
    }
}
```

a) Describe la interfaz que se define y su funcionamiento cuando se ejecuta el programa.

Al arrancar se abre una ventana en blanco. Al arrastrar sobre la ventana el ratón pulsado, aparecen en la esquina superior izquierda de la ventana las coordenadas de la posición del ratón según se sigue moviendo. Cuando el ratón se deciene o se suelta el botón, las coordenadas no cambian.

b) ¿Que ocurriría si se quita el comentario en la línea 3? ¿Y si se elimina la línea 1 y se substituye 2 por "`add("Center",text);`"? Razonar las respuestas.

Si se quita el comentario de la 4, el texto que muestra las coordenadas se mueve siguiendo el ratón.

Si se hace el otro cambio, el Label text ocupa toda la ventana tapando todo el área de ésta, de forma que los eventos de ratón los emite el Label y no la ventana. Por lo tanto la ventana no recibe estos eventos y permanece en blanco todo el tiempo.

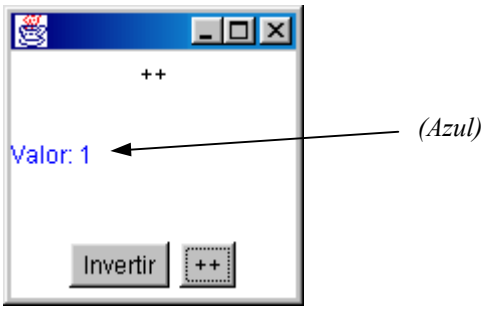
c) ¿Qué ocurriría si se inserta la sentencia "`throw new Exception();`" inmediatamente después de la línea 3? ¿Y si se inserta "`throw new NullPointerException();`"? Razonar las respuestas.

Con throw new Exception ( ) habría un error de compilación ya que el método no declara throws Exception , cosa que de todas formas no sería posible al ser un método heredado de la interfaz MouseMotionListener, en la que no se declara ninguna excepción para el método.

Con throw new NullPointerException ( ) el programa compilaría, ya que no hace falta declarar la excepción por ser subclase de RuntimeException. Al pulsar y arrastrar el ratón sobre la ventana y ejecutarse el método, saltaría una excepción con cada evento de drag generado. Como el método se ejecuta en el thread de la cola de eventos, el thread Main no se interrumpe y el programa funciona igual que antes (salvo por la emisión de excepciones).

12. a) Esboza la interfaz resultante nada más arrancar el programa Ventana, incluido al final de estas hojas, indicando los colores que difieran de blanco y negro.

b) Dibuja la interfaz después de pulsar los botones "Invertir" y "++", por este orden.



(Azul)

c) ¿Sobra alguna de las llamadas a repaint? Explica tu respuesta, y explica de forma muy breve cuándo debe llamarse a repaint en general.

Sobran las llamadas en mostrar e invertir, porque los métodos setText y setForeground forman parte del API de cambio de propiedades de las componentes predefinidas, y se ocupan ellos mismos de que se repinte la componente.

Se debe llamar a repaint cuando, habiendo redefinido el método paint en una componente, se modifica el estado interno de la componente de forma que afecte al resultado de paint.

13. En el mismo programa:

    a) ¿Cambiaría algo si el método incrementar fuese private? ¿Y si fuese protected?

    Si fuese private, error de compilación ya que no se podría invocar desde la clase Ventana. Si fuese protected no cambia nada, sigue siendo accesible dentro del package desde otras clases.

    b) ¿Cambiaría algo si la variable valor fuese final? En general, ¿tiene sentido que un constructor sea final?

    Si valor fuese final, error de compilación ya que en incrementar se cambia su valor, lo que no está permitido.

    Un constructor no puede ser final (da un error de compilación). Además, final significa que un método no puede sobreescribirse, pero los constructores no se sobreescriben nunca porque para empezar no se heredan.

    c) Cuando se arrancase dos veces el programa simultáneamente, ¿cambiaría algo si la variable estado fuese static? ¿Cambiaría algo que el método invertir fuese synchronized?

    No, a ambas preguntas. Cuando el programa se arranca dos veces se crean dos procesos diferentes con la máquina virtual y no comparten datos entre sí (a menos que se comuniquen de alguna forma, que no es el caso). En particular, los valores de variables estáticas no se comparten sino que mantienen su estado independientemente y por lo tanto no interfieren entre sí.

    Puesto que no se crean hilos, no influye en nada que el método sea synchronized.

14. Definir una clase de tipo ventana con un Label centrado en la parte superior que muestre en todo momento las coordenadas del ratón cuando éste se mueva sobre la ventana, y el string “Fuera” mientras el ratón se mueva fuera de ella.

```
class Ventana extends Frame implements MouseMotionListener, MouseListener {
    Label coordenadas;
    Ventana () {
            coordenadas = new Label ("", Label.CENTER);
            add ("North", coordenadas);
            addMouseMotionListener (this);
            addMouseListener (this);
    }
    public void mouseMoved (MouseEvent e) {
            coordenadas.setText (e.getPoint () .toString ());
    }
    public void mouseDragged (MouseEvent e) {}

    public void mouseEntered (MouseEvent e) {}
    public void mouseExited (MouseEvent e) {
            coordenadas.setText ("Fuera");
    }
    public void mousePressed (MouseEvent e) {}
    public void mouseReleased (MouseEvent e) {}
    public void mouseClicked (MouseEvent e) {}
}
```

15. ¿Cuál de las siguientes funcionalidades no es posible en un applet?

    a) Utilizar botones dentro del applet.
    b) Seguir el movimiento del ratón sobre el applet.
    c) Dibujar una gráfica.
    d) Cargar en el navegador un documento html localizado en el cliente.

16. ¿Cuál de las siguientes funcionalidades no es posible en un applet?

    a) Declarar una excepción en el método start.
    b) Implementar la interfaz Runnable y el método run.
    c) Arrancar un thread en el cliente.
    d) Cargar en el navegador un documento html localizado en el servidor.

17. ¿Puede un botón AWT ser un objeto remoto CORBA? En caso afirmativo, explicar brevemente qué clases sería necesario definir (sin escribir el código de las clases), y en caso contrario explicar por qué.

    No puede, porque los objetos CORBA se definen como subclases de las clases que se generan automáticamente a partir de la interfaz en IDL, y una subclase de java.awt.Button ya no puede derivar de otra clase.

18. En RMI:

    a) ¿Las excepciones se transmiten entre procesos por valor o por referencia?

       Por valor, ya que no se definen como objetos remotos y por otra parte la clase Exception implementa la interfaz java.io.Serializable. De hecho es imposible definir una clase de excepción como objeto remoto. Si por ejemplo queremos que un método remoto f de una interfaz X lance una excepción RemoteEx como objeto remoto, tendríamos algo como:

```
interface X extends Remote {
    void f () throws Remote, RemoteEx;
}
```

       donde RemoteEx fuese una interfaz remota pero al mismo tiempo extendiera Throwable, lo cual es imposible para una interface.

    b) ¿Es posible transmitir excepciones distintas de RemoteException entre servidor y cliente?

       Sí, cualquier excepción se puede declarar en una interfaz remota. Cuando se emite dentro de un método remoto y no se captura en el servidor, la excepción llega al cliente.

19. Escribir las interfaces y clases necesarias para crear una aplicación RMI en la que el usuario del cliente puede introducir texto por teclado, de tal manera que al pulsar Enter, el texto tecleado aparece en la consola de todos los clientes que estén conectados al mismo servidor.

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.io.*;

interface RemoteServer extends Remote {
    void checkin (RemoteClient clt) throws RemoteException;
    void checkout (RemoteClient clt) throws RemoteException;
    void broadcast (String msg) throws RemoteException;
}

interface RemoteClient extends Remote {
    void display (String msg) throws RemoteException;
}

class Server extends UnicastRemoteObject implements RemoteServer {
    Vector clients = new Vector ();
    Server (String name) throws Exception {
        Naming.rebind (name, this);
    }
    public void checkin (RemoteClient clt) throws RemoteException {
        clients.addElement (clt);
    }
    public void checkout (RemoteClient clt) throws RemoteException {
        clients.removeElement (clt);
    }
```

```
    public void broadcast (String msg) throws RemoteException {
        Enumeration enum = clients.elements ();
        while (enum.hasMoreElements ())
            ((RemoteClient) enum.nextElement ()) .display (msg);
    }
    static void main (String args[]) throws Exception {
        new Server ("Server");
    }
}

class Client extends UnicastRemoteObject implements RemoteClient {
    RemoteServer server;
    Client (RemoteServer srv) throws Exception {
        server = srv;
        server.checkin (this);
    }
    public void display (String msg) throws RemoteException {
        System.out.println (msg);
    }
    static void main (String args[]) throws Exception {
        RemoteServer srv = (RemoteServer) Naming.lookup ("Server");
        Client clt = new Client (srv);
        BufferedReader reader = new BufferedReader (new InputStreamReader (System.in));
        while (true) srv.broadcast (reader.readLine ());
    }
}
```

```
import java.awt.*;
import java.awt.event.*;

class Ventana extends Frame implements ActionListener {
	private Label etiqueta = new Label ("Accion", Label.CENTER);
	private Area area = new Area ();
	Ventana () {
		setBounds (0, 0, 150, 150);
		add ("North", etiqueta);
		add ("Center", area);
		Button boton1 = new Button ("Invertir");
		Button boton2 = new Button ("++");
		Panel botones = new Panel ();
		botones.add (boton1);
		botones.add (boton2);
		add ("South", botones);
		boton1.addActionListener (this);
		boton2.addActionListener (this);
	}
	public void actionPerformed (ActionEvent ev) {
		mostrar (ev.getActionCommand ());
		if (ev.getActionCommand () .equals ("++"))
			area.incrementar ();
		else area.invertir ();
	}
	public void mostrar (String texto) {
		etiqueta.setText (texto);
		repaint ();
	}
	public static void main (String a[]) {
		new Ventana () .setVisible (true);
	}
}

class Area extends Canvas {
	private boolean estado = false;
	private int valor = 0;
	public void invertir () {
		estado = !estado;
		if (estado) { setForeground (Color.blue); }
		else { setForeground (Color.black); }
		repaint ();
	}
	public void incrementar () {
		valor++;
		repaint ();
	}
	public void paint (Graphics g) {
		g.drawString ("Valor: " + valor, 0, getHeight()/2);
	}
}
```