**UNIVERSIDAD PONTIFICIA DE SALAMANCA EN MADRID**
**Facultad de Informática**
**Escuela Universitaria de Informática**
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software

***Cuadernillo de examen***

| | | | |
|---|---|---|---|
| ASIGNATURA | **Laboratorio de Sistemas Operativos Abiertos (Java)** | CÓDIGO | **321** |
| CONVOCATORIA | **Ordinaria de Junio de 2003** | PLAN DE ESTUDIOS | **1996** |
| ESPECIALIDAD | **Sistemas** | CURSO | **2002/2003** |
| TURNO | **Mañana** | CURSO ACADÉMICO | **3º** |
| CARÁCTER | **Anual** | PROGRAMA | **Ingeniería Técnica** |
| DURACIÓN APROXIMADA | **2 horas** | | |

# Soluciones propuestas

## Preguntas teórico-prácticas

**1.** La Máquina Virtual Java. Funcionalidad y características. Concepto de bytecode.

**2.** Clases de envoltura. Funcionalidad. Tipos de clases de envoltura. Constructores y métodos de las clases de envoltura.

**3.** Las clases derivadas de `Throwable`. Explique brevemente las clases derivadas más importantes de la clase `Throwable`.

**4.** Gestores de posicionamiento. Explique las características principales de las clases de la interfaz `LayoutManager`. Para cada una de ellas ponga un ejemplo de la disposición de componentes que realizan.

**5.** Ciclo de vida de un applet. Explique la utilidad de los distintos métodos que se ejecutan desde que se crea un applet hasta su destrucción.

*Documentación aportada en clase por el profesor.*

**Puntuación: 1 punto cada pregunta**

## Preguntas prácticas

**1.** Una empresa mantiene sus productos en un almacen. Cada producto tiene como atributos su identificador, la descripción y el precio. Algunos de los productos están en oferta en cuyo caso tendrán además el porcentaje de descuento.

- Codifique la clase `Almacen`. Dicha clase sólo contendrá un array de productos con el tamaño que el alumno desee. La clase tendrá un método buscarProducto al que se le pasará el código de un producto y devuelva la posición del array donde está o -1 en el caso de que no se encuentre.
- Codifique la clase `Producto`. Dicha clase tendrá un constructor que permita dar valores a todos sus atributos. También tendrá un método toString que permita mostrar una representación de un objeto `Producto` en forma de cadena.
- Codifique la clase `Oferta` derivada de `Producto`. Codifique en dicha clase un constructor que permita dar valor a todos sus atributos.
- Las clases `Producto` y `Oferta` tendrán un método polimórfico `calcularPrecio` que devuelva el precio del producto. En el caso de los productos en oferta devolverá el precio aplicando el descuento correspondiente.
- Codifique una clase `Factura`. En la factura sólo podrá ir un número determinado de productos, determinado por el atributo `numProductosMáximo` cuyo valor se pasará como argumento al constructor de la clase. Cada producto se almacenará en un elemento de un array. Será necesario guardar un campo con el número de productos incluidos en la factura. La clase `Factura` tendrá los siguientes métodos:
  - Un método `añadirProducto` que permita añadir un nuevo producto a la factura (el código de producto se pasará como argumento al método).
  - Un método `listarFactura` que liste todos los productos incluidos.
  - Un método `calcularTotal` que calcule el total de la factura.

**Puntuación: 2 puntos**

```
class Almacen{
    Producto existencias[];
    final int numProductos;

    Almacen(int n){
        numProductos = n;
        existencias = new Producto[n];
        llenarAlmacen();
    }

    void llenarAlmacen(){
        //Llena el almacen de distintos productos
        //En una aplicación real los productos se cargaría de una
        //base de datos u otro tipo de almacenamiento
        existencias[0] = new Producto(12345,"Bolígrafo BIC", 0.75);
        existencias[1] = new Producto(23456,"Paquete 500 DIN A4", 5.25);
        existencias[2] = new Oferta(34567,"Carpeta", 2.34, 10.0);
        existencias[3] = new Producto(45678,"Rotulador permanente", 1.5);
        existencias[4] = new Oferta(56789,"Caja disketes", 6.0,15.0);
    }

    int buscarProducto(long cod){
        int i = 0;
        while(i < numProductos -1){
            if(cod == existencias[i].idProd)
                break;
            i++;
        }
        if(cod == existencias[i].idProd)
            return i;
        else
            return -1;
    }
}

class Producto{
    long idProd = 0;
    String descr = "SinNombre";
    double precio = 0.0;

    Producto(long id, String d, double pre){
        idProd = id;
        descr = d;
        precio = pre;
    }

    public String toString(){
        return idProd + " " + descr + " " + precio;
    }

    double calcularPrecio(){
        return precio;
    }
}

class Oferta extends Producto{
    double descuento = 0.0;

    Oferta(long id, String d, double pre, double des){
        super(id,d,pre);
        descuento = des;
    }

    double calcularPrecio(){
        return precio - precio * descuento / 100;
    }
}

class Factura{
    Producto items[];
```

```
    int numItemsMax;
    static int numItems = 0;

    Factura(int n){
        numItemsMax = n;
        items = new Producto[n];
    }

    void añadirProducto(Almacen alm,long cod){
        int i = alm.buscarProducto(cod);
        if(i >= 0)
            items[numItems] = alm.existencias[i];
        numItems++;
    }

    void listarItems(){
        for(int i=0;i < numItemsMax;i++)
            System.out.println(items[i]);
    }

    double calcularTotal(){
        double total = 0;
        for(int i=0;i < numItemsMax;i++)
            total += items[i].calcularPrecio();
        return total;
    }
}
```

**2.** Una aplicación que, utilizando las clases del apartado anterior, permita generar facturas. La aplicación estará basada en una interfaz gráfica con un aspecto similar al siguiente:


Los nombres de los componentes son los que aparecen en la figura. Desarrolle la aplicación correspondiente de forma que:

- Al pulsar sobre el botón `btnNuevo`, se limpiarán todos los campos de texto del formulario.
- Al pulsar sobre el botón `btnAñadir` se añadirá a la factura el producto cuyo código aparece en el campo `txtIDProd`. Al añadir un producto se buscará el producto en el almacen y la descripción del mismo aparecerá en el campo `txtDescr`.
- Al pulsar sobre el botón `btnTotal` aparecerá en el campo `txtTotal` el total de la factura.

**Notas:**

- **Es necesario que el alumno coloque los campos en el contenedor, aunque la distribución de éstos la realizará el alumno a su libre elección.**
- **Se supone que la clase Almacen ya está cargada con los productos al comenzar la aplicación.**

**Puntuación: 2 puntos**

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

public class FacturasSimplificada{
```

```
   public static Almacen alm = new Almacen(5);
    public static Factura f = new Factura(5);

   public static void main(String args[]){

      FrameFactura frm = new FrameFactura();
      frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frm.show();
     }
}

class FrameFactura extends JFrame{
   JTextField txtIDProd = new JTextField("");
   JTextField txtDescr = new JTextField("");
    JTextField txtTotal = new JTextField("");
   JButton btnNuevo = new JButton("Nuevo");
   JButton btnAñadir = new JButton("Añadir");
    JButton btnTotal = new JButton("Total");

   FrameFactura(){
      setTitle("Factura");
      getContentPane().setLayout(new GridLayout(5,3));
      getContentPane().add(new JLabel("ID. producto:"));
      getContentPane().add(txtIDProd);
       getContentPane().add(new JLabel());
       getContentPane().add(new JLabel("Descripción:"));
       txtDescr.setEditable(false);
      getContentPane().add(txtDescr);
       getContentPane().add(new JLabel());
       getContentPane().add(new JLabel("Total:"));
       txtTotal.setEditable(false);
       getContentPane().add(txtTotal);
       getContentPane().add(new JLabel());
      getContentPane().add(btnNuevo);
      getContentPane().add(btnAñadir);
       getContentPane().add(btnTotal);
      pack();

      //Registra los interfaces en los componentes
      btnNuevo.addActionListener(new FacturasActionListener());
      btnAñadir.addActionListener(new FacturasActionListener());
       btnTotal.addActionListener(new FacturasActionListener());
   }
    //Gestión de eventos
    class FacturasActionListener implements ActionListener{
        public void actionPerformed(ActionEvent e){
            //Obtiene el objeto que ha producido el evento
            Object source = e.getSource();
            long id = Long.parseLong(txtIDProd.getText());
                if(source == btnNuevo){
                    txtIDProd.setText("");
                    txtDescr.setText("");
                }
                else if(source==btnAñadir){
                    int i = Facturas.alm.buscarProducto(id);
                    txtDescr.setText(Facturas.alm.existencias[i].descr);
                    Facturas.f.añadirProducto(Facturas.alm, id);
                }
                else if(source == btnTotal){
                    double total = Facturas.f.calcularTotal();
                    txtTotal.setText(String.valueOf(total));
                    Facturas.f.listarItems();
                }
        }
   }
}
```

**3.** Desarrolle una clase para una excepción que informe de que el producto introducido en el campo `txtIDProducto` no existe en el almacen. Indique que modificaciones habría que realizar en el ejercicio anterior para lanzar dicha excepción..

**Puntuación: 1 punto**

### Clase Exception

```
public class NoExisteProductoException extends Exception{
    public NoExisteProductoException() {
        super("El producto no existe en el almacén");
    }
}
```

### Modificaciones en el método buscarProducto de la clase Almacen

(Las modificaciones aparecen en negrita)

```
    int buscarProducto(long cod) throws NoExisteProductoException{
        int i = 0;
        while(i < numProductos -1){
            if(cod == existencias[i].idProd)
                break;
            i++;
        }
        if(cod == existencias[i].idProd)
            return i;
        else
            throw new NoExisteProductoException();
    }
```

### Modificaciones en el método añadirProducto de la clase Factura

(Las modificaciones aparecen en negrita)

```
    void añadirProducto(Almacen alm,long cod){
        try{
           int i = alm.buscarProducto(cod);
           if(i >= 0)
               items[numItems] = alm.existencias[i];
           numItems++;
            } catch (NoExisteProductoException e){};
    }
```

### Modificaciones en el método actionPerformed de la clase FacturasActionListener

(Las modificaciones aparecen en negrita)

```
                …
                else if(source==btnAñadir){
                    try{
                        int i = Facturas.alm.buscarProducto(id);
                        txtDescr.setText(Facturas.alm.existencias[i].descr);
                        Facturas.f.añadirProducto(Facturas.alm, id);
                    } catch (NoExisteProductoException excep){
                        System.out.println(excep.getMessage());
                    }
                }
                …
```