Examen de Programación II
(Ingeniería Informática)

Junio 2010

1) Lenguaje C (2 puntos)

1.a) Escribir el módulo "reemplaza_chars" (ficheros `reemplaza_chars.h` y `reemplaza_chars.c`) que defina una única función (`reemplaza`) que retorna un string que es igual a otro pero con todas las ocurrencias de un carácter reemplazadas por otro. Por ejemplo, para el string `"hola adios"` si se pide reemplazar las `'a'` por `'e'` la función retornaría `"hole edios"`.

Una descripción más detallada de la función `reemplaza` sería la siguiente:

- `orig` (parámetro de entrada): string original (en el ejemplo `"hola adios"`).
- `old` (parámetro de entrada): carácter a reemplazar (en el ejemplo la `'a'`).
- `new` (parámetro de entrada): carácter a poner en el lugar del "old" (en el ejemplo la `'e'`).
- `num_cambios` (parámetro de salida): devuelve el número de cambios realizados (en el ejemplo 2).
- valor de retorno: nuevo vector con los caracteres reemplazados (en el ejemplo `"hole edios"`).

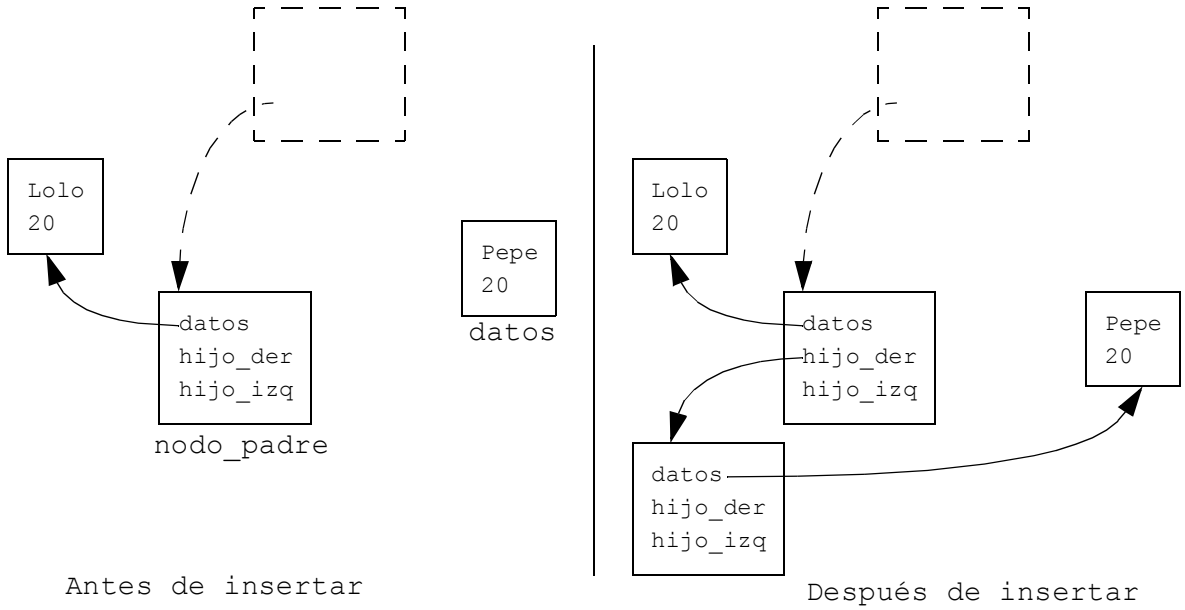
Escribir un programa principal muy sencillo (que no pida datos al usuario) que utilice la función `reemplaza`.

Indicar cual sería el comando que permitiría compilar y enlazar el programa principal desarrollado y crear un ejecutable llamado "`pru_reemplaza.exe`".

1.b) Se dispone el siguiente fichero "`arbol.h`" que permite implementar una estructura de datos tipo árbol:

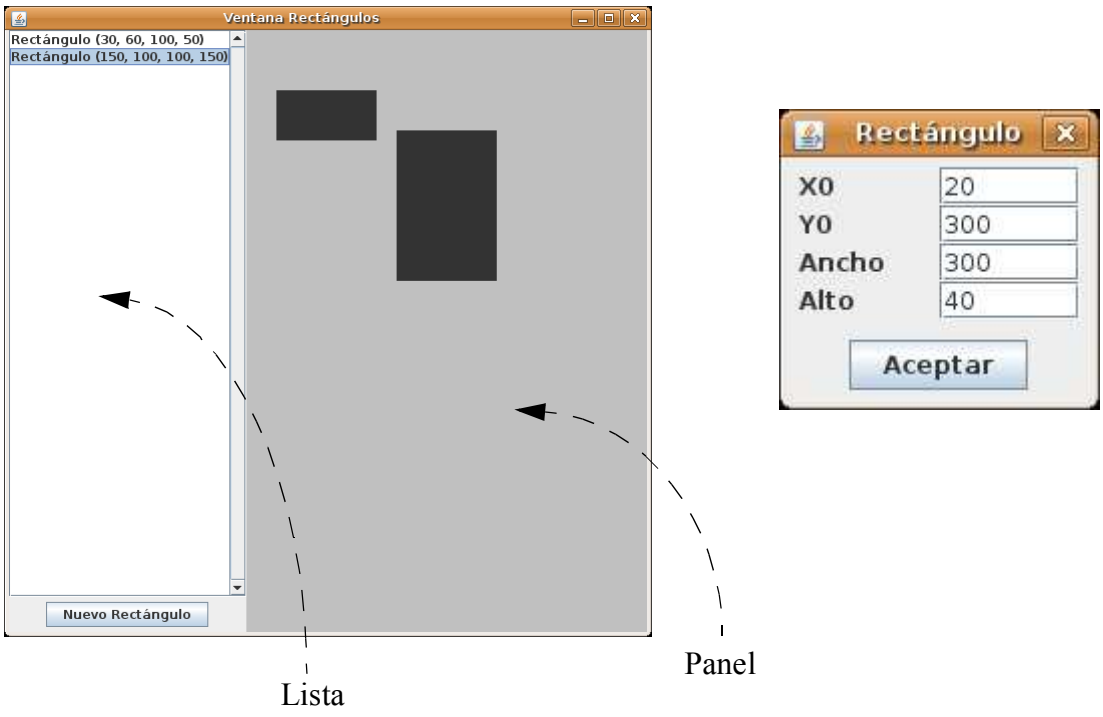
```
// -----   fichero arbol.h  -------

// datos de cada nodo
typedef struct {
  char * nombre;
  int edad;
} datos_t;

// Cada uno de los nodos del árbol
typedef struct nodo {
  datos_t *datos; // puntero a los datos del nodo
  struct nodo *hijo_der; // puntero al hijo derecho
  struct nodo *hijo_izq; // puntero al hijo izquierdo
} nodo_t;

// inserta el hijo derecho de un nodo
void inserta_hijo_derecho(nodo_t *nodo_padre,
                          const datos_t *datos);

// en una implementación real habría muchas otras funciones
```

Escribir el fichero "`arbol.c`" que implemente la función `inserta_hijo_derecho` cuya funcionalidad es la descrita en la figura siguiente:



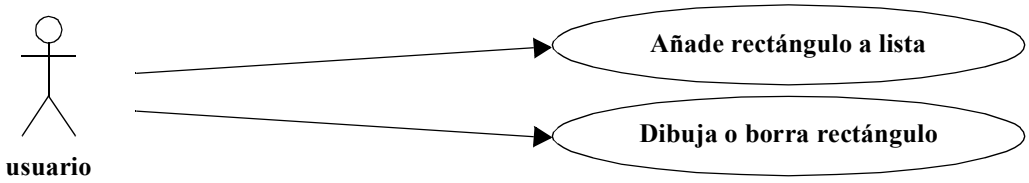
2) Programación dirigida por eventos (3.5 puntos)

Se pretende realizar una aplicación gráfica formada por la ventana `VentanaRectangulos` y el diálogo `DialogoNuevoRectangulo` que tienen la apariencia mostrada a continuación:



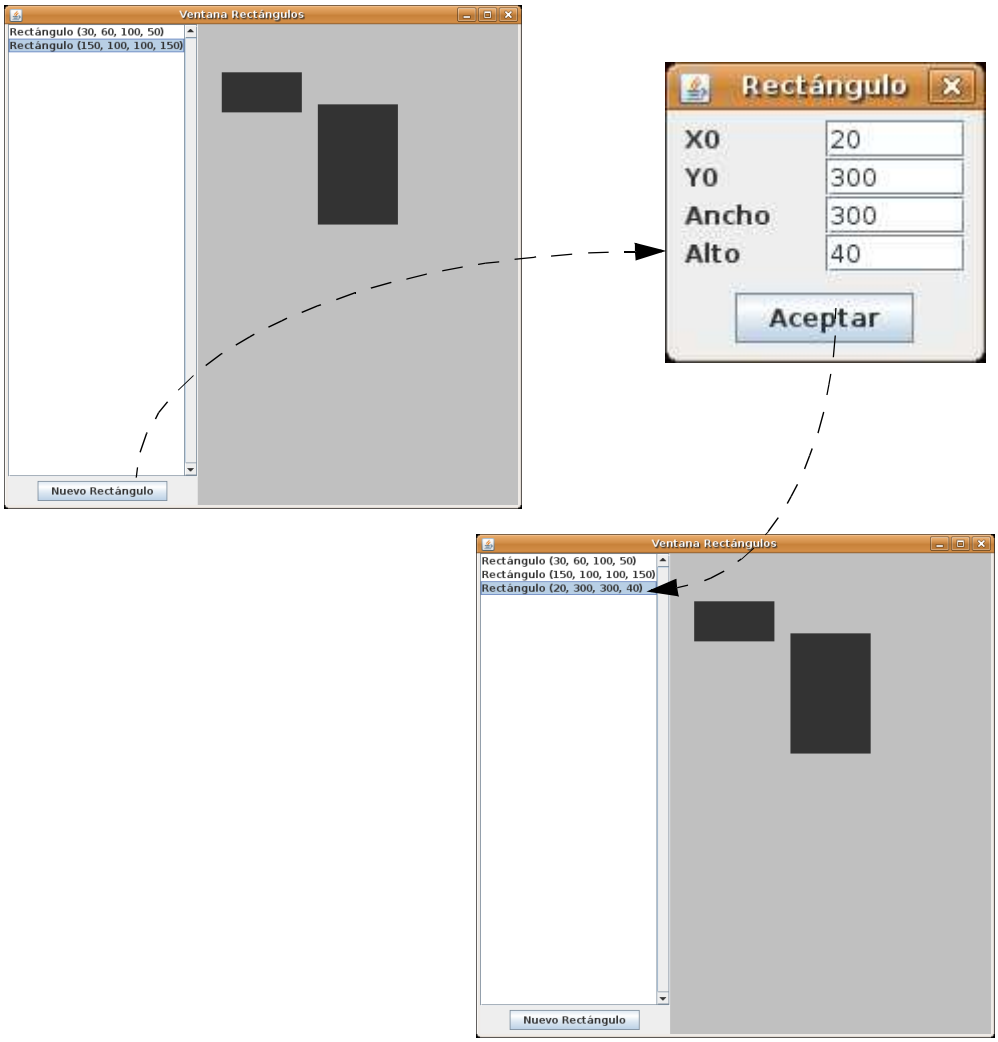
La ventana `VentanaRectangulos` tiene en su parte izquierda una lista con los rectángulos disponibles para dibujar junto con sus dimensiones (x0, y0, ancho, alto). En su parte derecha hay un panel donde se dibujan algunos de los rectángulos existentes en la lista. Debajo de la lista hay un botón que provoca la aparición del diálogo `DialogoNuevoRectangulo` que permite introducir las dimensiones de un nuevo rectángulo que se añadirá a la lista.

La interacción del usuario con la aplicación es la descrita en los siguientes caso de uso:



Caso de uso "Añade rectángulo a lista":

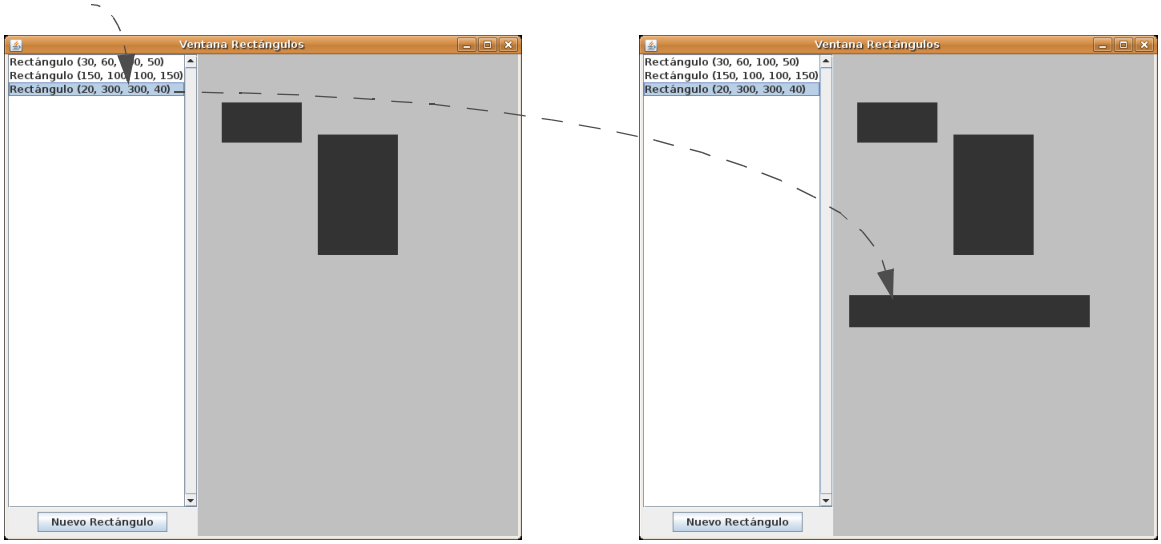
1. El usuario pulsa el botón "Nuevo Rectángulo".
2. La aplicación muestra el diálogo para introducir las dimensiones de un rectángulo.
3. El usuario introduce las dimensiones y pulsa aceptar o cierra el diálogo
   3.a. Si el usuario cierra el diálogo finaliza el caso de uso.
   3.b. En el caso pulsar aceptar, la aplicación muestra el nuevo rectángulo en la lista de la `VentanaRectangulos`.



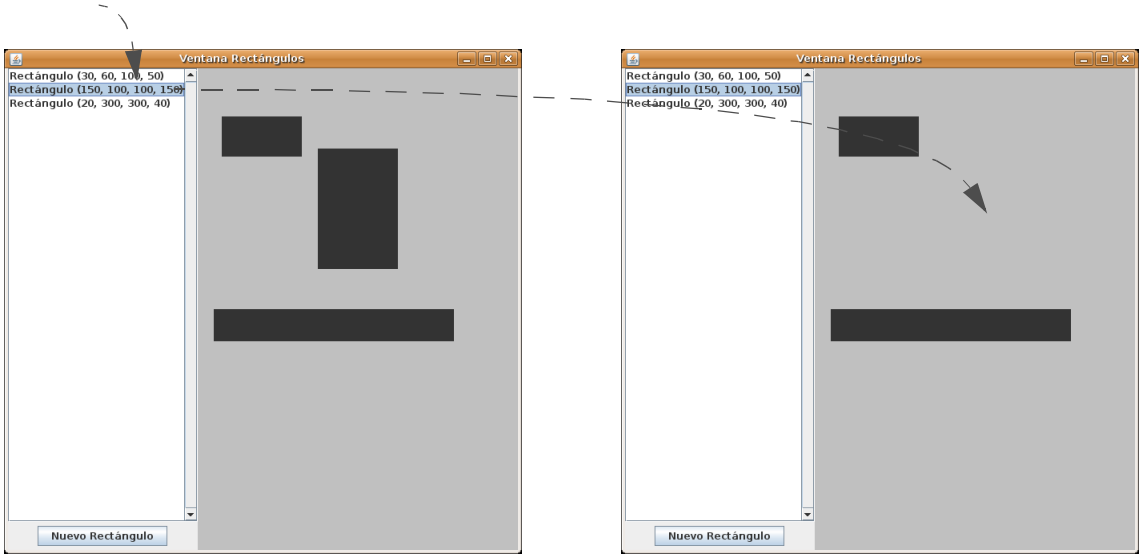
Caso de uso "Dibuja o borra rectángulo":

1. El usuario hace doble "clic" sobre algún rectángulo de la lista.
2. La aplicación dibuja o borra el rectángulo:
   2.a. Si el rectángulo no estaba dibujado en el panel, se dibuja.
   2.b. Si ya estaba dibujado, desaparece.

doble "clic"



doble "clic"



Se dispone de la clase `Rectangulo` ya implementada:

```
public class Rectangulo {
  final int x0;
  final int y0;
  final int ancho;
  final int alto;

  public Rectangulo(int x0, int y0, int x1, int y1) {
    this.x0 = x0;
```

```
    this.y0 = y0;
    this.ancho = x1;
    this.alto = y1;
  }

  @Override
  public String toString() {
    return "Rectángulo ("+x0+", "+y0+", "+ancho+", "+alto+")";
  }
}
```

La clase `VentanaRectangulos` se encuentra parcialmente implementada:

```
public class VentanaRectangulos extends JFrame {

  // Botón nuevo rectángulo
  private JButton bNuevoRect = new JButton("Nuevo Rectángulo");

  // Constructor
  public VentanaRectangulos(String title){
    distribución de componentes y ajustes generales de la
    ventana (no hacer por el alumno)

    // asocia manejadores                    Hacer por el alumno
    ...
  }

  public static void main(String[] args) {
      new VentanaRectangulos("Ventana Rectángulos");
  }

  // otros componentes y atributos          Hacer por el alumno
  ...

  // clases manejadoras                     Hacer por el alumno
  ...
}
```

La clase `DialogoNuevoRectangulo` también se encuentra parcialmente implementada:

```
public class DialogoNuevoRectangulo extends JDialog {
  // componentes
  private JTextField tfX0 = new JTextField("");
  private JTextField tfY0 = new JTextField("");
  private JTextField tfAncho = new JTextField("");
  private JTextField tfAlto = new JTextField("");
  private JButton bAceptar = new JButton("Aceptar");

  // constructor
  public DialogoNuevoRectangulo(JFrame owner) {
    distribución de componentes y ajustes generales del
    diálogo (no hacer por el alumno)
```

```
      // asocia manejador
      bAceptar.addActionListener(new ManejadorAceptar());
    }

    // otros atributos y métodos            Hacer por el alumno
    ...

    // clase manejadora ManejadorAceptar    Hacer por el alumno
    ...
}
```

Se pide escribir el código de la aplicación completa salvo las partes que ya están implementadas. Puede ser necesario crear nuevas clases, métodos, atributos, etc. Las partes marcadas como "Hacer por el alumno" se indican a modo de orientación, dependiendo de la implementación elegida por el alumno las partes a modificar podrían ser diferentes.

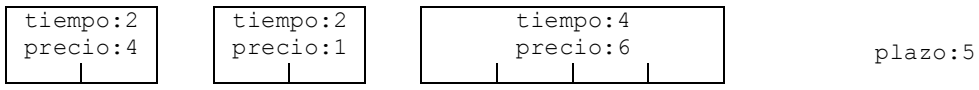
(Indicar claramente a qué parte de cada clase parcialmente implementada corresponde el código escrito).

3) Esquemas algorítmicos

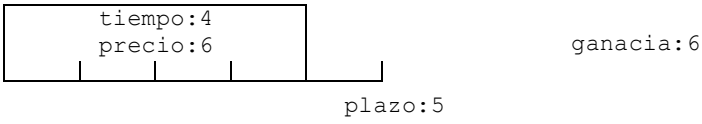
Una fábrica recibe una serie de encargos para fabricar que deben estar terminados en un determinado plazo de tiempo (el mismo para todos los encargos). Cada encargo tarda un tiempo en fabricarse y la empresa cobra su precio al cliente sólo si completa la fabricación dentro del plazo.

Los directivos de la fábrica desean ganar el máximo dinero posible, por lo que les interesa disponer de un algoritmo que les permita obtener el conjunto de encargos que pueda fabricarse en el plazo establecido y que les proporcione una ganancia máxima.

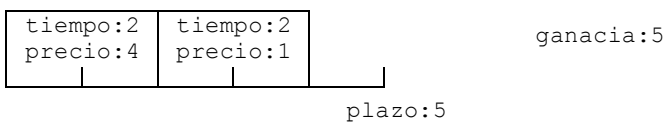
Por ejemplo supongamos que la empresa tiene tres encargos y un plazo de finalización de 5:



La solución óptima a este ejemplar del problema sería hacer sólo el tercer trabajo, lo que produce una ganancia de 6:



Es posible hacer más trabajos (el primero y el segundo) pero la ganancia sería peor, por lo que esta opción constituye una peor solución al problema que la anterior.



3.a) (1.5 puntos) Implementar un algoritmo heurístico voraz con tiempo de ejecución mejor que $O(n^2)$ que permita encontrar una buena solución para el problema planteado (no hace falta que encuentre la óptima). Detallar también los cambios realizados en la clase `Encargo` (en caso de que sea necesario realizar alguno). Explicar con lenguaje natural el criterio de selección utilizado e indicar cual es el ritmo de crecimiento del algoritmo. El método que implemente el algoritmo deberá tener la siguiente cabecera:

```
/**
 * Selección heurística de los encargos que trata de maximizar
 * las ganancias
 * @param encargos encargos pendientes
 * @param plazo plazo máximo de finalización
 * @return mejor conjunto de encargos encontrado
 */
public static ConjuntoEncargos selecEncargosVoraz(
     Encargo[] encargos, double plazo)
```

3.b) (2.5 puntos) Implementar un algoritmo basado en "Ramificación y Poda" que permita encontrar la solución óptima al problema planteado. Este algoritmo deberá hacer la ramificación más eficiente posible y sacar partido del heurístico anteriormente desarrollado. Escribir también el código correspondiente a los cambios que es necesario realizar en las clases `ConjuntoEncargos` y `Encargo` (en caso de que sea necesario realizar alguno).

3.c) (0.5 puntos) Razonar brevemente si la ordenación de los encargos de acuerdo a algún criterio puede favorecer la eficiencia en el caso promedio del algoritmo basado en "Ramificación y Poda". ¿Cual sería ese criterio de ordenación?.

Para implementar los algoritmos solicitados, se dispone de la clases `Encargo` y `ConjuntoEncargos`. (El alumno podrá hacer las modificaciones que crea conveniente a ambas clases)

Código de la clase `Encargo`:

```
/**
 * Cada uno de los encargos pendientes
 */
public class Encargo {
  // tiempo que tarda en fabricarse el encargo
  public final double tiempo;

  // precio a cobrar por el encargo
  public final double precio;

  // constructor
  public Encargo(double tiempo, double precio) {
    this.tiempo = tiempo;
    this.precio = precio;
  }
}
```

Código de la clase `ConjuntoEncargos`:

```
/**
 * Un conjunto de encargos
 */
public class ConjuntoEncargos {

  // lista con los encargos incluidos
  private LinkedList<Encargo> encargos = new LinkedList<Encargo>();

  // suma de las duraciones de todos los encargos incluidos
  private double durTotal=0;

  // suma de los precios de todos los encargos incluidos
  private double precTotal=0;

  // plazo límite de fabricación de encargos
  private final double plazo;

  /**
   * Constructor
   * @param plazo plazo de finalización de los encargos
   */
  public ConjuntoEncargos(double plazo) {
    this.plazo = plazo;
  }

  /**
   * Retorna la suma de las duraciones de todos los encargos
   * incluidos en la lista
   * @return la suma de las duraciones de todos los encargos
   */
  public double duracionTotal() {
    return durTotal;
  }

  /**
   * Retorna la suma de los precios de todos los encargos
   * incluidos en la lista
   * @return suma de los precios de todos los encargos
   */
  public double precioTotal() {
    return precTotal;
  }

  /**
   * Retorna si se puede incluir un encargo a el conjunto y la
   * duración total sigue estando por debajo del plazo
   * @param e encargo a incluir
   * @return true si es posible realizar el encargo dentro del
   * plazo y false en caso contrario
```

```
     */
    public boolean cabeEncargo(Encargo e) {
      return plazo - durTotal >= e.tiempo;
    }


    /**
     * Comprueba si un encargo ya está en el conjunto
     * @param e encargo a comprobar si ya está en el conjunto
     * @return true si ya está y false en caso contrario
     */
    public boolean encargoYaIncluido(Encargo e) {
      return encargos.contains(e);
    }

    /**
     * Incluye un encargo en el conjunto.
     * No realiza ninguna comprobación (por eficiencia), asume que el
     * encargo cabe y no está incluido aún
     * @param e encargo a incluir.
     */
    public void incluyeEncargo(Encargo e) {
      encargos.add(e);
      durTotal += e.tiempo;
      precTotal += e.precio;
    }
}
```