T4: Patternbox

1. CAMBIAR EL FORMATO DE LAS FUENTES 1 Y 2 A "ROMAN-BASELINE" Y "TRUETYPE-FONT" RESPECTIVAMENTE Y VOLVER A EJECUTAR LA APLICACIÓN.

Para modificar las fuentes lo único que hay que hacer es sustituir el primer parámetro que se pasa en las clases Bold e Italic por Font.ROMAN_BASELINE y Font.TRUETYPE_FONT.

```
public class Bold implements IFontStyle {
    @Override
    public Font getFont() { return new Font("ROMAN-BASELINE", Font.ROMAN_BASELINE, 24); }
}
```

```
public class Italic implements IFontStyle {
    @Override
    public Font getFont() {
        return new Font("TRUETYPE-FONT", Font.ITALIC, 24);
    }
}
```

2. CÓMO AÑADIRÍAS UN NUEVO BOTÓN DE FORMATO A LA VENTANA?

Para añadir un nuevo botón que de formato a la ventana simplemente bastaría con crear una función que añada al frame este botón. El formato se añadiría a traves de la clase Factoria pasandole el parametro que identifique a un nuevo formato, en este caso el 4.

```
    private JButton getJButtonBig(){
        if(jButtonBig == null){
            jButtonBig = new JButton();
            jButtonBig.setText("G");
            IFontStyle f = FontFactory.createFont(4);
            jButtonBig.setFont(f.getFont());
            jButtonBig.addActionListener(new ActionListener(){

                @Override
                public void actionPerformed(ActionEvent e) {
                    jTextArea.setFont(FontFactory.createFont(4).getFont());
                }
            });
            jButtonBig.setBounds(new Rectangle(195, 151, 65, 43));

        }
        return jButtonBig;
    }
```

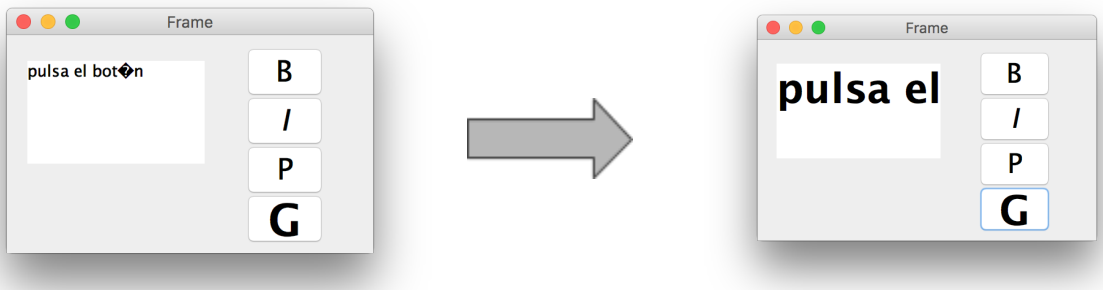
Se modifica el método de la clase FontFactory para que gestione un nuevo caso y cree un nuevo objeto de la clase que da el nuevo formato.

```java
public class FontFactory {

    public static IFontStyle createFont(int n){
        switch (n){
            case 1:
                return new Bold();
            case 2:
                return new Italic();
            case 3:
                return new Plain();
            case 4:
                return new BigFont();
            default:
                return null;
        }
    }
}
```

Esta clase que da formato tiene que implementar la interfaz IFontStyle como lo han hecho Bold, Italic y Plain.

```java
public class BigFont implements IFontStyle{
    @Override
    public Font getFont() {
        return new Font("Bold", Font.BOLD, 36);
    }
}
```



Patrón Adapter

1- CREAR UN PROGRAMA PRINCIPAL, CON UN VECTOR DE OBJETOS DE TIPO PERSON (NOMBRE, CIUDAD) Y LOS VISUALICE ORDENADOS POR NOMBRE, Y A CONTINUACIÓN POR CIUDAD.

Primero creo dos clases que implementen las interfaces Iterator y Comparator para adaptar Person a los requisitos de la función *Sorting.sortedIterator(Iterator itr, Comparator c)*.

```
8   public class PersonIterator implements Iterator {
9
10      private Enumeration<Person> lista;
11
12      public PersonIterator(Vector<Person> lista){
13              this.lista = lista.elements();
14      }
15
16      @Override
17      public boolean hasNext() {
18          return lista.hasMoreElements();
19      }
20
21      @Override
22      public Person next() {
23          return lista.nextElement();
24      }
25  }
```

```
public class PersonCityComparator implements Comparator<Person> {
    @Override
    public int compare(Person o1, Person o2) {
        return o1.getCiudad().compareTo(o2.getCiudad());
    }
}
```

```
public class PersonNameComparator implements Comparator<Person>{

    @Override
    public int compare(Person o1, Person o2) {
        return o1.getNombre().compareTo(o2.getNombre());
    }
}
```

Para imprimir un vector de forma ordenada solamente hay que llamar al método estático sortedIterator de la clase Sorting y pasarle como parámetros una instancia de PersonIterator y otra del comparador que se necesite en función del criterio de ordenación.
El resultado devuelto será un objeto del tipo Iterator que tendrá Personas ordenadas.

2- DADA LA CLASE ADDRESSBOOK CREAR UN PROGRAMA PRINCIPAL, QUE DE IGUAL FORMA QUE EL PUNTO ANTERIOR, VISUALICE SUS CONTACTOS ORDENADOS POR NOMBRE, Y A CONTINUACIÓN POR CIUDAD. PARA DESARROLLAR ESTE APARTADO, NO SE PUEDE MODIFICAR EL CÓDIGO DE LA CLASE ADDRESSBOOK.

Para mostrar la lista de contactos ordenada sin modificar la clase original se crea una clase Adapter que recibe en el constructor un objeto del tipo AddressBook, almacena en una Enumeration todos los contactos. Este adapter implementa la interfaz iterator. Como en el punto anterior se le pasa al método sortedIterator esta clase como primer parámetro y como segundo uno de los comparadores utilizados anteriormente.

```java
public class AddressBookAdapter implements Iterator{

    Enumeration<Person> lista;
    public AddressBookAdapter(AddressBook ab){
        loadEnumeration(ab);
    }

    private void loadEnumeration(AddressBook ab){
        int size = ab.getSize();
        Vector<Person> vPersons = new Vector<>();

        for (int i = 0; i < size; i++) {
           vPersons.add(ab.getPerson(i));
        }

        lista = vPersons.elements();
    }


    @Override
    public boolean hasNext() {
        return lista.hasMoreElements();
    }

    @Override
    public Person next() {
        return lista.nextElement();
    }
}
```

```java
public static void main(String[] args){

    Iterator<Person> itr = Sorting.sortedIterator(new AddressBookAdapter(new AddressBook()), new PersonNameComparator());
    Iterator<Person> itr2 = Sorting.sortedIterator(new AddressBookAdapter(new AddressBook()), new PersonCityComparator());

    while (itr.hasNext()){
        System.out.println("Nombre: " + itr.next().getNombre());
    }
    while (itr2.hasNext()){
        System.out.println("Ciudad: " + itr2.next().getCiudad());
    }
}
```

Patrón Observer

1. QUEREMOS AÑADIR UNA NUEVA VENTANA QUE ÚNICAMENTE NOS INDIQUE CUÁL ES COLOR SELECCIONADO TAL Y COMO SE MUESTRA EN LA SIGUIENTE FIGURA

Para añadir una nueva ventana basta con crear una clase *PantallaTexto* que herede de JFrame e implemente la interfaz Vista que hace de observador.
En el constructor se pasará como parámetro un sujeto concreto, en este caso será de la clase *UnColor* que hereda de *Modelo*. En el mismo constructor se añade a la lista de observables del color el objeto desde que se llama y se crea la interfaz.
El otro método que se añade es el de *update*() que es el que viene definido por la interfaz, esta función es la encargada de actualizar el texto de la interfaz cada vez que es invocada, para ello obtiene un string con el nuevo valor llamando al método *getColor* del atributo color de la clase.

```
public class PantallaTexto extends JFrame implements Vista {

    private UnColor color;
    private JTextField text;

    public PantallaTexto(UnColor color){
        this.color = color;
        this.color.attach(this);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setPreferredSize(new Dimension(100, 100));
        this.pack();
        this.setLocationRelativeTo(null);
        this.add(addText());
        this.setVisible(true);

    }

    private JTextField addText(){
        if(text == null){
            text = new JTextField("No hay ningún color seleccionado");
        }
        return text;
    }


    @Override
    public void update() {
        String newColor = color.getColor();
        text.setText("El color actual es: " + newColor);
    }
}
```

El otro cambio que se realiza es en la clase Principal que inicializa el programa, se crea un nuevo objeto del tipo *PantallaTexto* al que se le pasa el modelo.

```
public static void main(String args[]){
    UnColor modelo=new UnColor();
    Vista pc=new PantallaColor(modelo);
    Vista pt = new PantallaTexto(modelo);
    new DesplegableFrame(modelo);
}
```

2. CREAR UNA APLICACIÓN QUE TENGA A LA VEZ 3 DESPLEGABLES Y DONDE LAS VISTAS DE LA DERECHA CORRESPONDIESEN A CADA DESPLEGABLE TAL Y COMO SE MUESTRA EN LA SIGUIENTE FIGURA.

Para este último punto es suficiente con crear un modelo diferente para cada grupo de ventanas, de esta forma el comportamiento es independiente y solo observan a un Sujeto.

```
public static void main(String args[]) {
    for (int i = 0; i < 3; i++) {
        crearPantallas();
    }
}

private static void crearPantallas(){
    UnColor modelo = new UnColor();
    new PantallaColor(modelo);
    new PantallaTexto(modelo);
    new DesplegableFrame(modelo);
}
```