En este documento se presentan una serie de modificaciones al Proyecto de las Casas Rurales que implica el re-diseño de la aplicación. Deberás plantearte cómo realizar cada modificación tanto a nivel de diseño como sus implicaciones en el código.

Patrón Factory Method

La aplicación de las casas rurales está diseñada en un arquitectura de 3 niveles, donde la presentación puede acceder a un objeto de la lógica de negocio ubicado bien de forma local o bien a un objeto distribuido a través de servicios web. En ambos casos, si bien los objetos comparten la misma interfaz (*ApplicationFacadeInterface*), la implementación varia notablemente.
En la aplicación actual, la clase *StartWindow*, decide cual de las 2 implementaciones utilizar. Concretamente en el método *main()*, teniendo en cuenta el valor de la variable *isLocal*, se asigna a la variable de la lógica de negocio *facadeInterface* cual de las implementaciones a utilizar.

Modifica la aplicación para que la obtención del objeto de la lógica de negocio se centralice en un objeto factoría, y sean las clases de la presentación las que decidan cual de las implementaciones de la lógica de negocio utilizar. Diseña e implementa la solución indicando qué clases juegan el papel de Creator, Product y ConcreteProduct.

Patrón Iterator

Queremos modificar el método de la clase FacadeImplementation

```
public Vector<RuralHouse> getAllRuralHouses()
```

por la siguiente signatura:

```
public ExtendedIterator<RuralHouse> ruralHouseIterator();
```

de manera que en vez de devolvernos un Vector de casas Rurales, nos devuelva un Iterador de las casas Rurales. Sin embargo, este nuevo "Iterador extendido", además de poder recorrer los elementos de la forma tradicional(secuencialmente hacia adelante), puede ir secuencialmente hacia atrás, o bien posicionarse en el primer o último elemento. La signatura de la interfaz ExtendedIterador es la siguiente:

```
public interface ExtendedIterator extends Iterator {

    //devuelve el elemento actual y pasa al anterior
    public Object previous();

    //true si existe el elemento anterior
    public boolean hasPrevious();

    //Se posiciona en el primer elemento
    public void goFirst();

    //Se posiciona en el último elemento
    public void goLast();
}
```

Un ejemplo de ejecución donde recorremos todas las casas rurales en secuencia inversa, y a continuación en secuencia tradicional seria:

```
public static void main(String[] args) {
   boolean isLocal=true;
   //obtener el objeto fachada
   ApplicationFacadeInterface facadeInterface=RuralHouseApp.getApplicationFacade(isLocal);
   ExtendedIterator<RuralHouse> i=facadeInterface.ruralHousesIterator();
   RuralHouse rh;
   i.goLast();
   while (i.hasPrevious()){
      rh=i.previous();
      rh.print();
   }
   //Aunque suponemos que hemos llegado al principio, realizamos la operación
   i.goFirst();
   while (i.hasNext())()){
      rh=i.next();
      rh.print();
   }
```

Se pide:

Implementa el Iterador extendido, y realiza un programa similar al del ejemplo que visualice todas las casas rurales en ese orden.

Patrón Adapter

Crear una nueva ventana donde aparezcan los datos de todas las casas rurales de un Propietario en un JTable. Nota: No se puede modificar la clase Owner. Diseña e implementa la solución.

Patrón Observer

Queremos extender el sistema con 2 nuevos tipos de usuarios como potenciales clientes de las casas rurales: UsuarioParticular y AgenciaDeViajes.
Estos nuevos clientes quieren hacer uso de la nueva funcionalidad del sistema ActivarAlertaCasaRural, que les permita subscribirse a una casa rural, de manera que cada vez que se añada una nueva oferta a la casa donde están subscritos se les notifique. Diseña e implementa la solución.