## 

**FACULTAD DE CIENCIAS E INGENIERÍA**

**ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**

**TEMA:**

Patrones Factory

**CURSO:**

Programación I

**PROFESOR:**

Coronel Castillo Eric Gustavo

**AUTORES:**

Aguila Ruiz Bengy Código: 16201005

Vicente Rosas Noe Código: 16201040

Acosta Espinoza Max Código: 16201003

**FECHA DE ENTREGA:**

06-10-2017

**Los Olivos – Perú**

**2017-II**

**INDICE**

1. **Resumen**
2. **ANTECEDENTES** 
   1. **Identificación del problema**
   2. **Problema**
   3. **Como se va a implementar la solución**
   4. **Planteamiento de solución**
   5. **Justificación de la investigación**
   6. **Objetivo general**
   7. **Objetivo especifico**
3. **MARCO TEORICO**
   1. **Patrón Factory**
   2. **Fabricas**
   3. **Los patrones** 
      1. **Simple Factory**
      2. **Factory method**
      3. **Abstract factory**
4. **CASOS**
5. **CONCLUCIONES**
6. **RECOMENDACIONES**
7. **REFERENCIAS**

**RESUMEN**

En este trabajo analizaremos los patrones de fabricación más conocidos. Llamamos patrones de fabricación a aquellos patrones que involucran algún tipo de factoría o fábrica (factory, en inglés) de objetos. Estos patrones entran en la categoría de patrones de creación, la cual comparten con otros patrones tales como el Singleton, Builder y Prototype.

Los objetos de fabricación (fábricas) tienen la responsabilidad de crear instancias de objetos de otras clases. Tienen además la responsabilidad y el conocimiento necesario para encapsular la forma en que se crean determinados tipos de objetos en una aplicación.

Existe una gran cantidad de patrones de diseño. Pero en este trabajo trataremos los patrones Factory más conocidos como Abstract Factory, Factory Method, y Simple Factory.

**IDENTIFICACION DEL PROBLEMA**

Hoy en día observamos mucho desorden al momento de programar, debió a que los programadores se enfocan más en que el software funcione sin importarles la forma en como fue desarrollado, de modo que, con el tiempo se manifiestan problemas, que difícilmente se podrían localizar por culpa de un código mal escrito y mal estructurado,

dificultando el trabajo a la hora de darle mantenimiento al software, debido que al modificar o cambiar algunas partes del código se manifiestan gran variedad de errores, a causa de un código sucio, provocando la pérdida de tiempo al solucionar el problema.

En ese sentido el uso de patrones Factory nos facilita al entendimiento de códigos, debido a que tiene una arquitectura establecida y fácil de entender.

**PROBLEMA**

¿De qué manera podemos implementar y derivar el uso de patrones Factory en nuestro código y como nos beneficia?

**PLANTAMIENTO DE SOLUCION**

* Aplicando métodos que ayuden a separar las instancias y poder corregir posibles errores que se den.
* Aplicando un orden de clases que ayude al mejor entendimiento del código.
* Una clase derivada toma la decisión sobre qué clase instanciar y cómo instanciarla
* Una clase necesita instanciar otra clase derivada de una tercera clase, pero no sabe cuál. Factory Method permite a la clase derivada tomar esta decisión.
* El patrón Abstract Factory proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

**COMO SE VA A IMPLEMENTAR LA SOLUCION**

Aplicando Patrón de Diseño Factory (patrones de fabricación)

**JUSTIFICACIÓN DE LA INVESTIGACIÓN**

En la actualidad al momento de programar se observa mucho “código sucio”, de modo que dificulta su comprensión, y es propenso a fallos que difícilmente podemos localizar para corregir, en ese caso hemos tenido la oportunidad de Investigar y dar a conocer el uso de patrones Factory, dándonos la facilidad de entender el código y poder desarrollar un mejor software.

**OBJETIVO GENERAL**

Analizar el uso de Patrones Factory para poder aplicarlo en los códigos que realicemos.

**OBJETIVOS ESPECÍFICOS**

* Enseñar el uso de Patrones Factory como una opción para programar de manera ordenada.
* Identificar en qué casos se podría implementar la utilización de Patrones Factory.
* Mostrar en qué casos utilizaremos los patrones Factory.
* mostrar ejemplos con el uso de Patrones Factory.

**MARCO TEÓRICO**

**PATRON FACTORY**

El patrón de diseño “Factory” se enfoca en la creación de una interface que facilite la creación de objetos que se organizan por diferentes subclases. Esto ocurre con frecuencia cuando se hace uso de la herencia. Una clase abstracta se genera conteniendo los atributos generales, y después se crean clases para objetos específicos. Para evitar llamar constructores específicos se deben crear interfaces que nos ayuden en estas tareas. ("Patrón de diseño Factory - michelletorres.mx", 2017)

Los objetos de fabricación (fábricas) tienen la responsabilidad de crear instancias de objetos de otras clases. Tienen además la responsabilidad y el conocimiento necesario para encapsular la forma en que se crean determinados tipos de objetos en una aplicación. ("Patrones de Fabricación: Fábricas de Objetos", 2017)

Existen diferentes patrones Factory en este caso trataremos tres, Abstract Factory, Factory Method, y Simple Factory los cuales desarrollaremos en el trascurso

**FÁBRICAS**

Para comenzar con nuestro estudio debemos definir claramente qué es una factoría o fábrica. De la misma forma que sucede con muchos conceptos en la Ingeniería del Software, existen varias definiciones para este término (en la entrega anterior hemos experimentado esta situación para el concepto de patrón); por tanto, es fundamental clarificar este concepto para sentar las bases de nuestro estudio posterior.

El término **Factory** (o fábrica) adolece de ser sobre utilizado e impreciso. Mucha gente se refiere a Factory para indicar una implementación de **Factory Method** o de **Abstract Factory** (patrones definidos en el libro del GoF). Otros, sin embargo, se refieren a un objeto que crea instancias de otros, aunque no sigue las reglas de los patrones mencionados anteriormente.

La falta de consenso sobre el término dificulta la comunicación que, como ya hemos visto anteriormente, es uno de los objetivos de los patrones (poder referirse en forma unívoca a una abstracción de mayor nivel que la clase individual, incrementando así el vocabulario de los ingenieros de software) ("Patrones de Fabricación: Fábricas de Objetos", 2017).

**LOS PATRONES**

Los **Patrones** permiten establecer un vocabulario común de diseño, cambiando el nivel de abstracción a colaboraciones entre clases y permitiendo comunicar experiencia sobre dichos problemas y soluciones. Son también un gran mecanismo de comunicación para transmitir la experiencia de los ingenieros y diseñadores experimentados a los más nóveles, convirtiéndose en unas de las vías para la gestión del conocimiento. Por tanto, es muy importante establecer un significado claro para este término, a los efectos de construir este vocabulario común.

Llamaremos fábrica, factoría o Factory a una clase que implemente uno o más métodos de creación, que son los métodos que se encargan de crear instancias de objetos (estas instancias pueden ser de esta misma clase o de otras). Esta clase tiene entre sus responsabilidades la creación de instancias de objetos, pero puede tener también otras responsabilidades adicionales. Los métodos de creación pueden ser estáticos. ("Patrones de Fabricación: Fábricas de Objetos", 2017).

**FACTORY PATTERN (****SIMPLE FACTORY)**

Una fábrica Simple normalmente es llamada por el cliente mediante un método estático, y devuelve uno de varios objetos que heredan / implementan el mismo padre.

La manera más fácil para recordar esto es que una fábrica simple es llamada directamente por la clase que quiere crear un objeto (la clase llamante se conoce como el "cliente"). La fábrica simple devuelve una de muchas clases diferentes. Todas las clases que una fábrica simple puede devolver o heredar de la misma clase padre o implementar la misma interfaz.

**Paso uno:** llama a un método en la fábrica. Aquí tiene sentido utilizar un método estático. Los parámetros de su llamada indican a la fábrica qué clase crear.

**Segundo paso:** la fábrica crea su objeto. La única cosa a tener en cuenta es que de todos los objetos que puede crear, los objetos tienen la misma clase padre, o implementar la misma interfaz.

**Paso tres:** la fábrica devuelve el objeto, y es por eso por lo que el paso dos tiene sentido. Dado que el cliente no sabía lo que iba a ser devuelto, el cliente está esperando un tipo que coincida con la clase / interfaz principal. (Jain, 2014)

**MÉTODO FACTORY (****FACTORY METHOD)**

* Define una interfaz para crear un objeto, pero dejando en manos de las subclases la decisión de qué clase concreta instanciar
* Permite que una clase delegue en sus subclases la creación de objetos
* También conocido como virtual constructor

**ESTE PATRÓN DEBE SER UTILIZADO CUANDO:**

* Una clase no puede prever la clase de objetos que tiene que crear
* Una clase quiere que sus subclases decidan qué objetos crean
* Las clases delegan responsabilidades a una de entre varias subclases auxiliares, y queremos localizar en qué subclase concreta se ha delegado.

**ESTE PATRÓN ESTÁ COMPUESTO POR:**

* **Product (Documento):** define la interfaz de los objetos que crea el método factoría
* **Concrete Product (Mi Documentó):** implementa la interfaz de Product
* **Creator (Aplicación):**
* Declara el método factoría que devuelve un objeto de tipo Product. Puede definir una implementación por defecto de dicho método, que devuelva un objeto de algún producto concreto Concrete Product.
* Puede llamar al método factoría para crear un objeto de tipo Product
* **Concrete Creator (Mi Aplicación):** sobrescribe el método factoría para devolver un objeto de algún Concrete Product

**CONSECUENCIAS**

* Elimina la necesidad de ligar clases específicas de una aplicación al código, que trata con la interfaz Product y funciona con cualquier clase Concrete Product.
* El cliente debe crear clases hijas de Creator para cada tipo de producto concreto
* Proporciona enganches para las subclases.
* La creación de objetos con métodos factoría es más flexible
* Las subclases pueden dar una versión extendida del código padre

**VENTAJAS**

* Se gana en flexibilidad, pues crear los objetos dentro de una clase con un "Método de Fábrica" es siempre más flexible que hacerlo directamente, debido a que se elimina la necesidad de atar nuestra aplicación unas clases de productos concretos.
* Se facilitan futuras ampliaciones, puesto que se ofrece las subclases la posibilidad de proporcionar una versión extendida de un objeto, con sólo aplicar en los Productos la misma idea del "Método de Fábrica".
* Se facilita, en cuanto a que se hace natural, la conexión entre jerarquías de clases paralelas, que son aquellas que se generan cuando una clase delega algunas de sus responsabilidades en una clase aparte. Ambas jerarquías de clases paralelas son creadas por un mismo cliente y el patrón Método de Fábrica establece la relación entre parejas de subclases concretas en las mismas. ("Factory Method - EcuRed", n.d.)

**INCONVENIENTES**

* Se obliga al cliente a definir subclases de la clase "Creador" sólo para crear un producto concreto y esto puede no ser apropiado siempre.

### **ABSTRACT FACTORY**

El patrón Abstract Factory nos permite crear, mediante una interfaz, conjuntos o familias de objetos (denominados productos) que dependen mutuamente y todo esto sin especificar cuál es el objeto concreto.

**Este patrón se puede aplicar cuando:**

* Un sistema debe ser independiente de cómo sus objetos son creados.
* Un sistema debe ser 'configurado' con una cierta familia de productos.
* Se necesita reforzar la noción de dependencia mutua entre ciertos objetos.

### **Elementos del Patrón Abstract Factory**

**Fabrica Abstracta**: Define un conjunto de métodos (interfaz) para la creación de productos abstractos.

* **Fabrica Concreta 1/2**: Implementa la interfaz de la Fabrica Abstracta para la creación de los distintos productos concretos.
* **Producto Abstracto A/B**: Define la interfaz de los objetos de tipo Producto A/B.
* **Producto Concreto A1/A2/B1/B2**: Implementan su respectiva interfaz representando un producto concreto.

### **Pros/Contras del Patrón Abstract Factory**

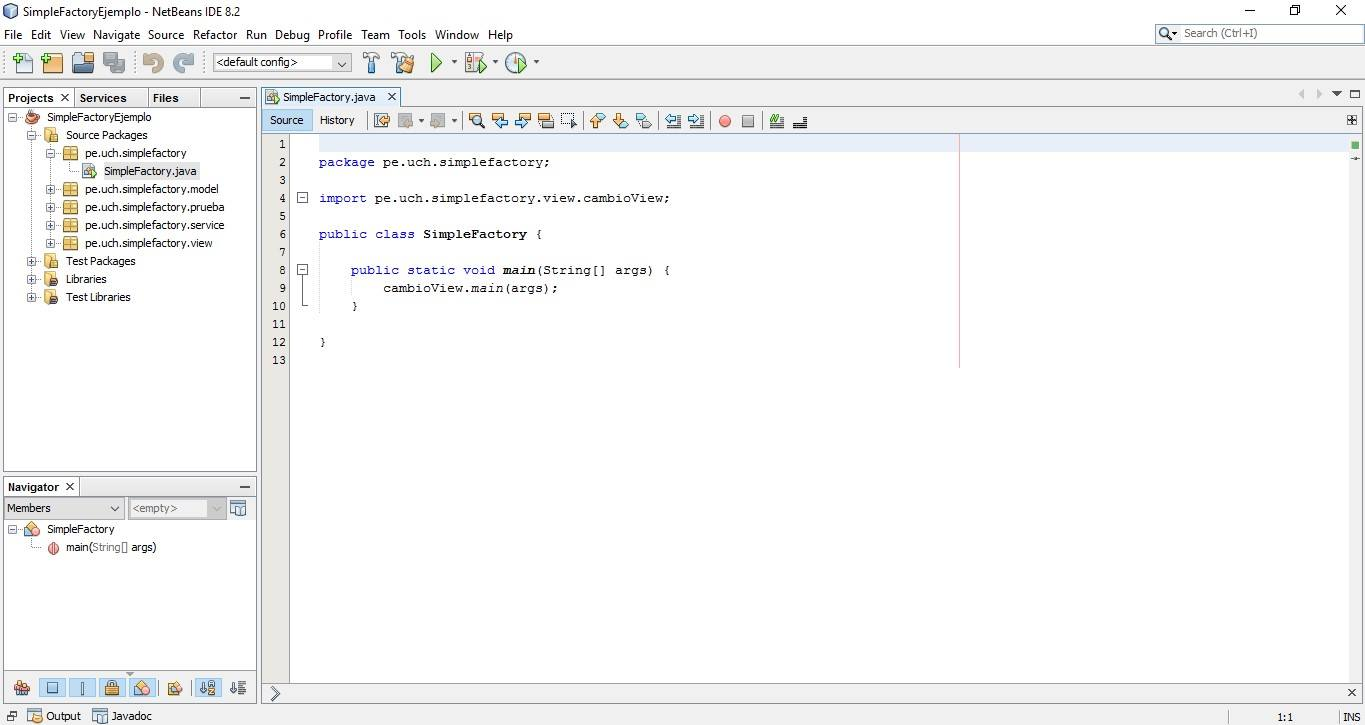
* + Brinda flexibilidad al aislar a las clases concretas.
* + Facilita cambiar las familias de productos.
* - Para agregar nuevos productos se deben modificar tanto las fabricas abstractas como las concretas. ("Patrón Abstract Factory - Línea de Código", 2013)

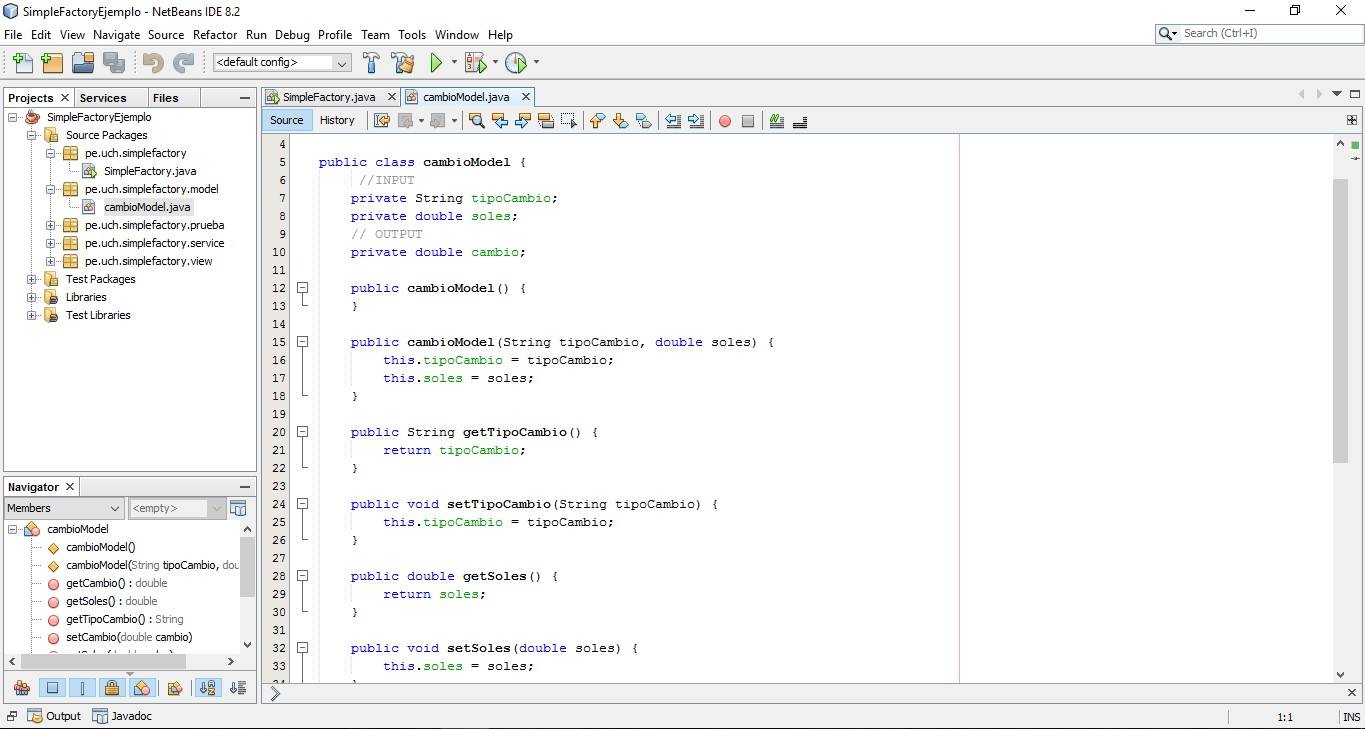
### **IMPLEMENTACION DE SOLUCION**

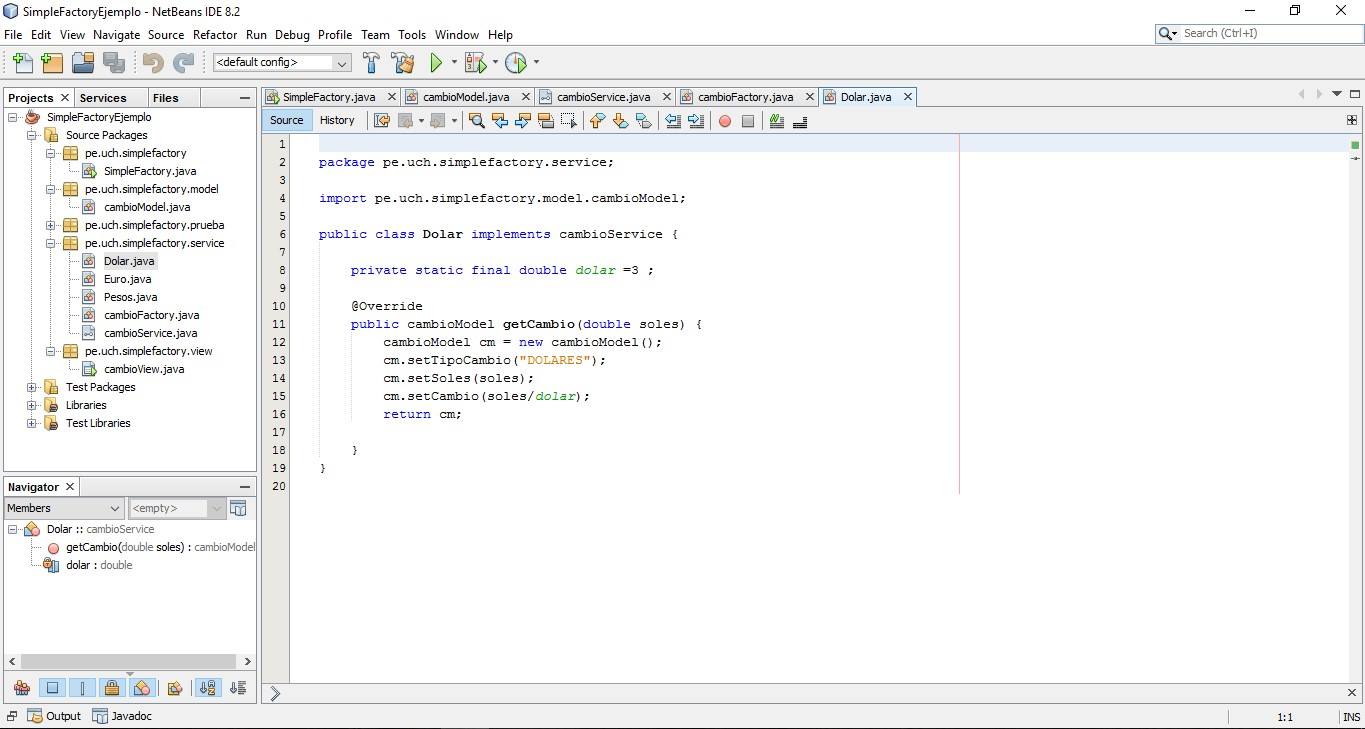
Dar a conocer a todos nuestros compañeros sobre los patrones Factory (Patrones de diseño de fabrica) y como puedes implementar en el código que programen de manera que tengan una buena estructura de código, que este bien definido y establecido para que cualquier otro programador pueda comprender nuestro código.

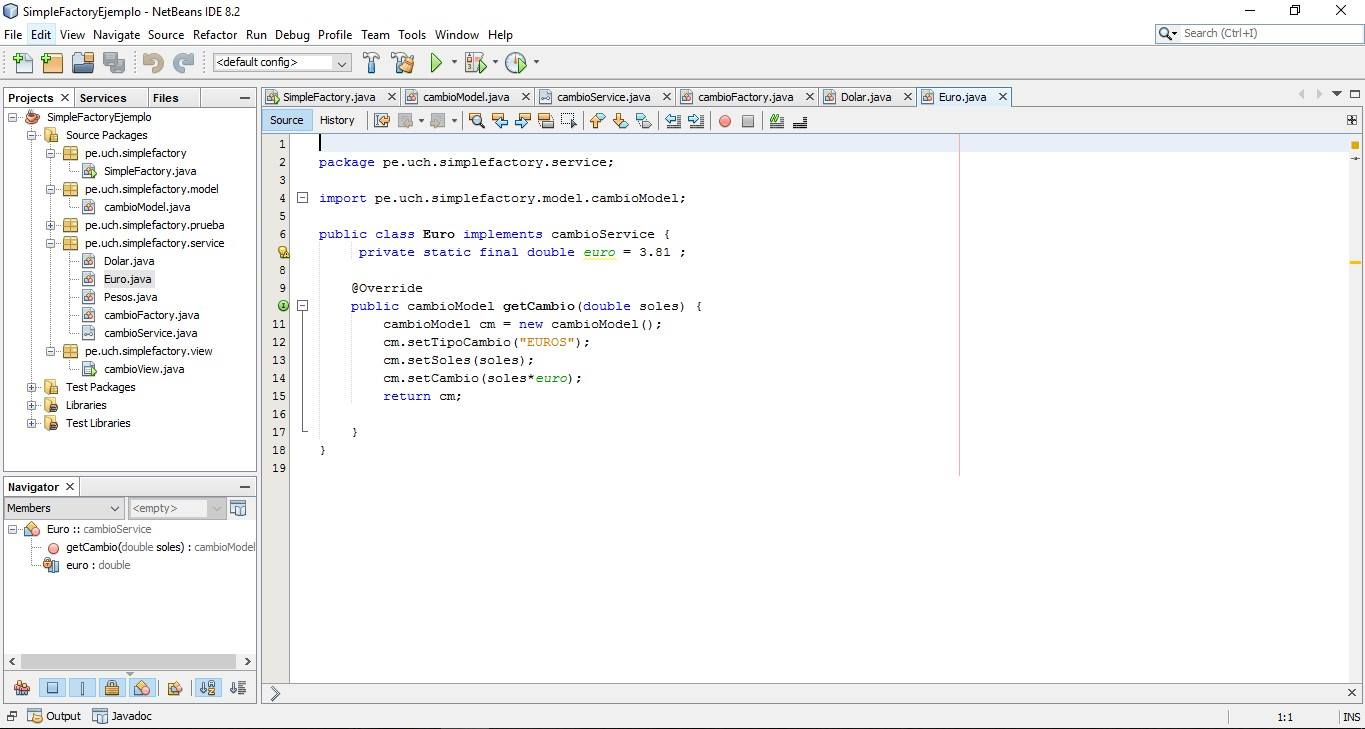
**CASOS**

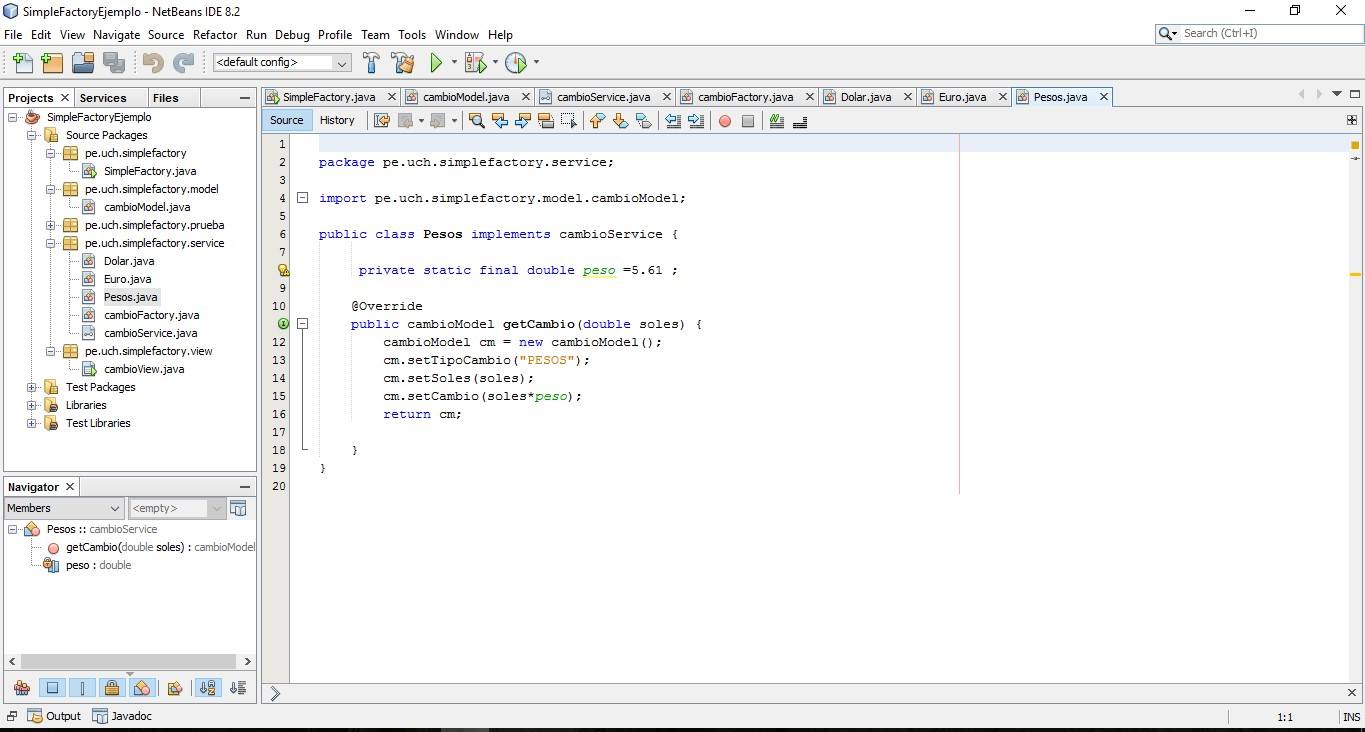
**Ejemplo 1: utilizando simple Factory en un proyecto de cambio de soles a dólares, euros y peso.**

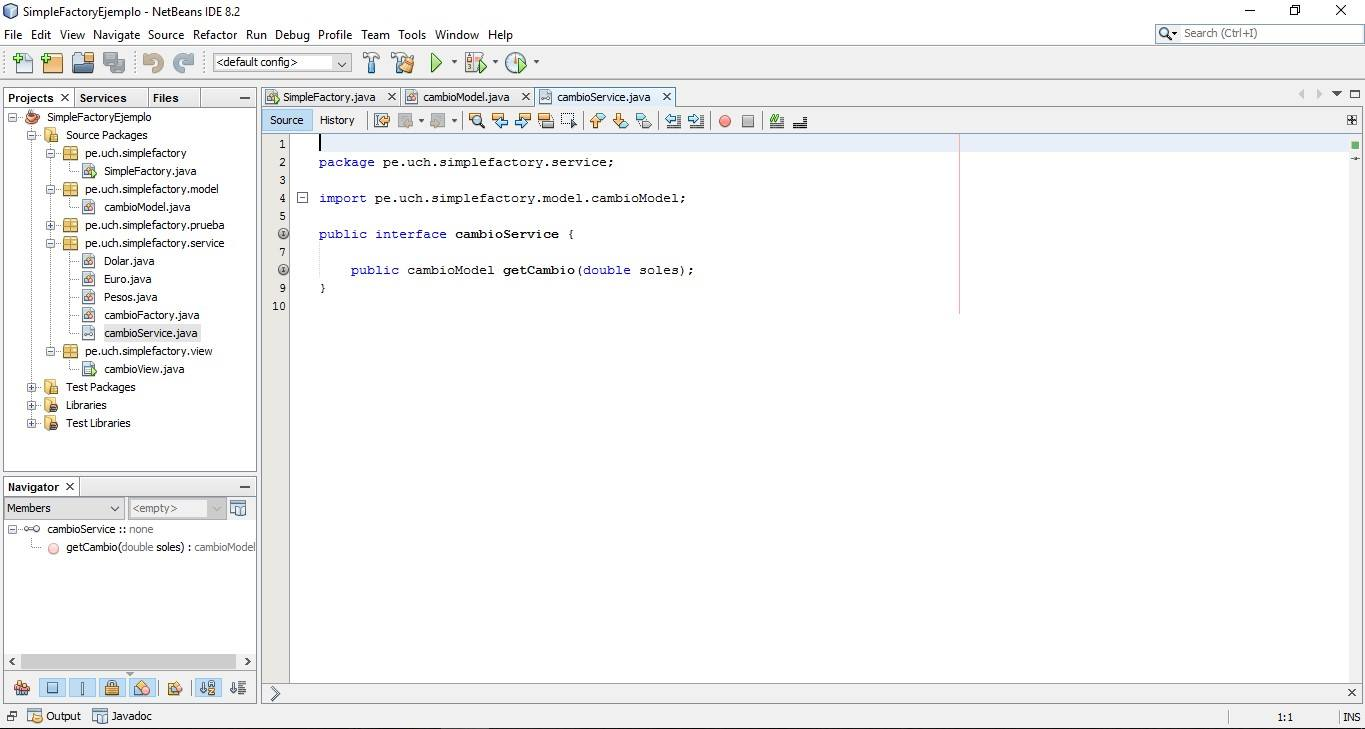


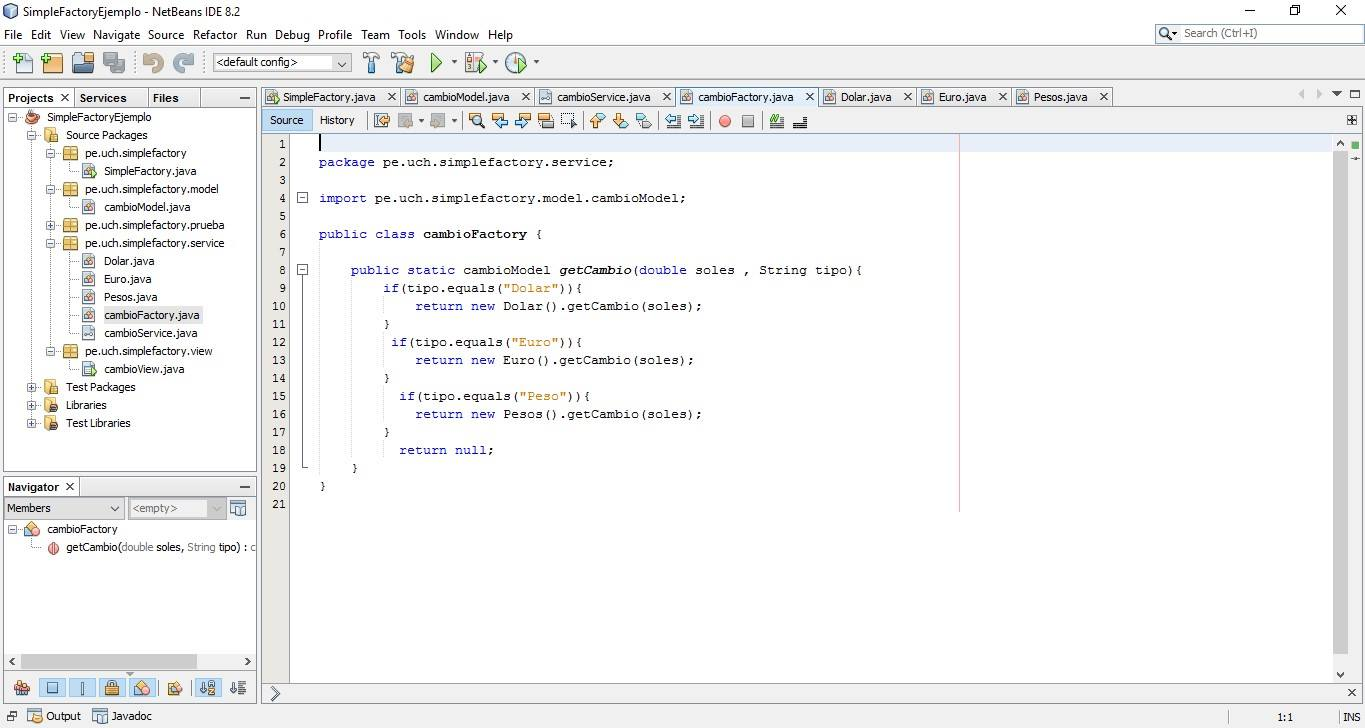






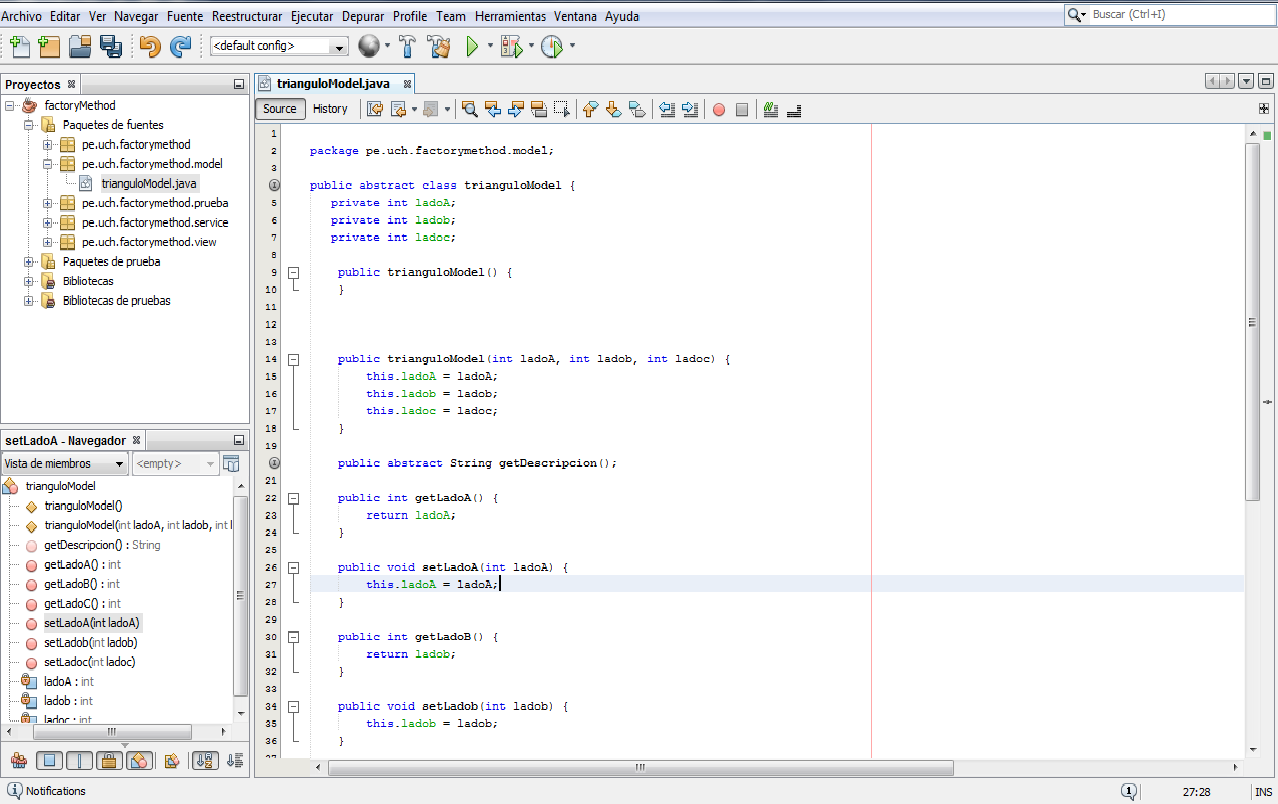


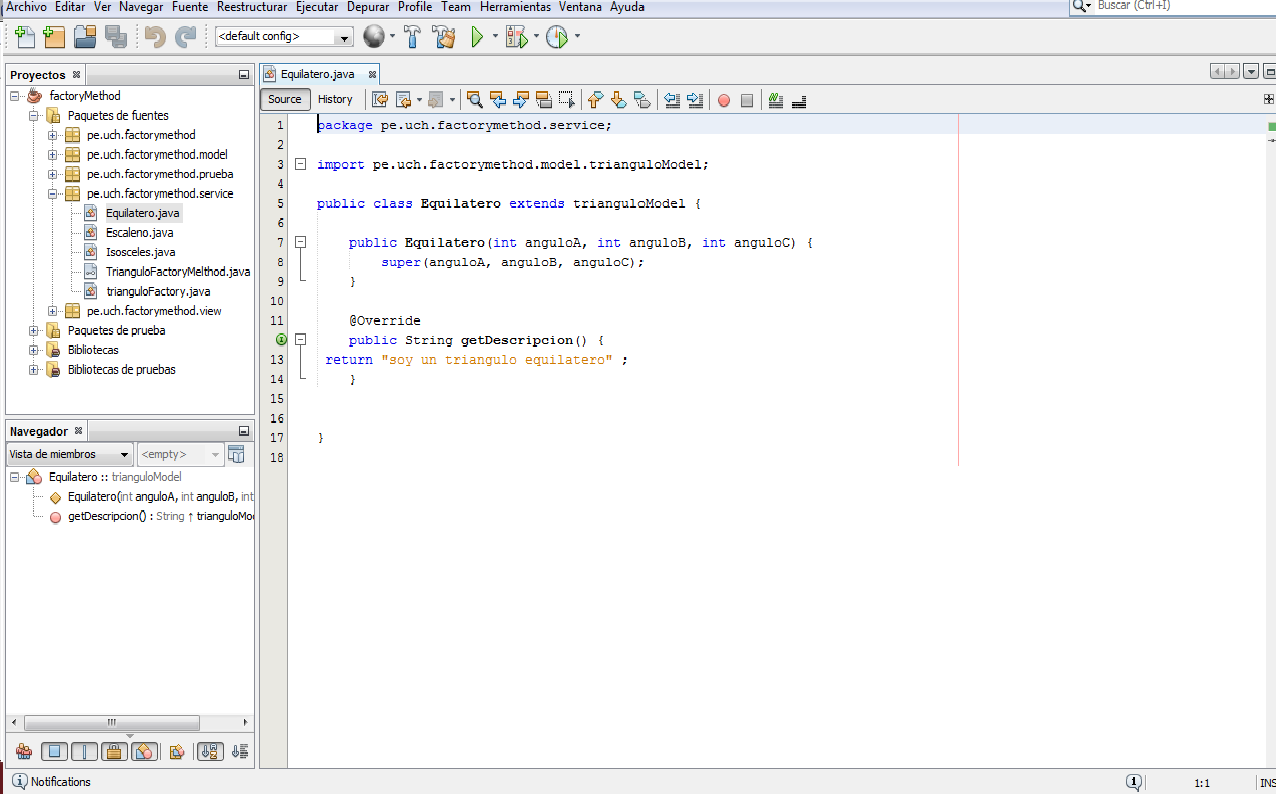


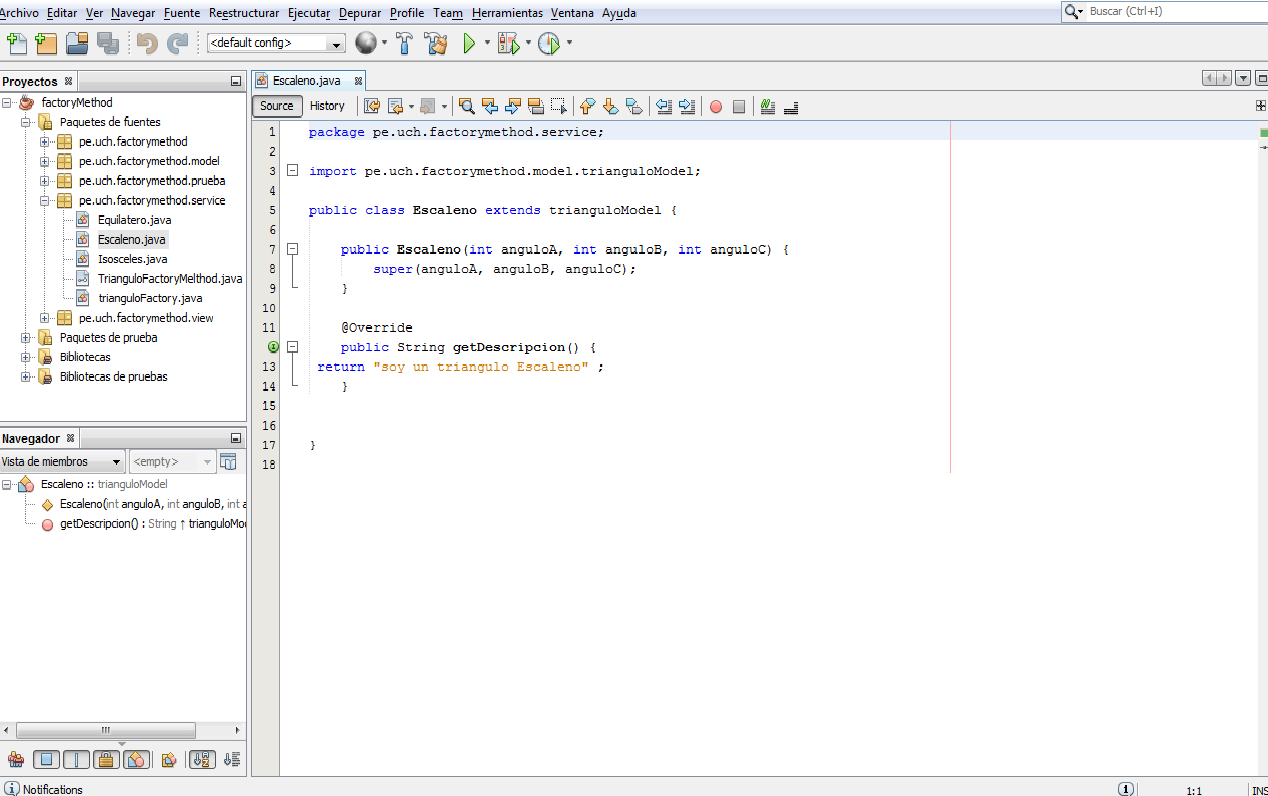


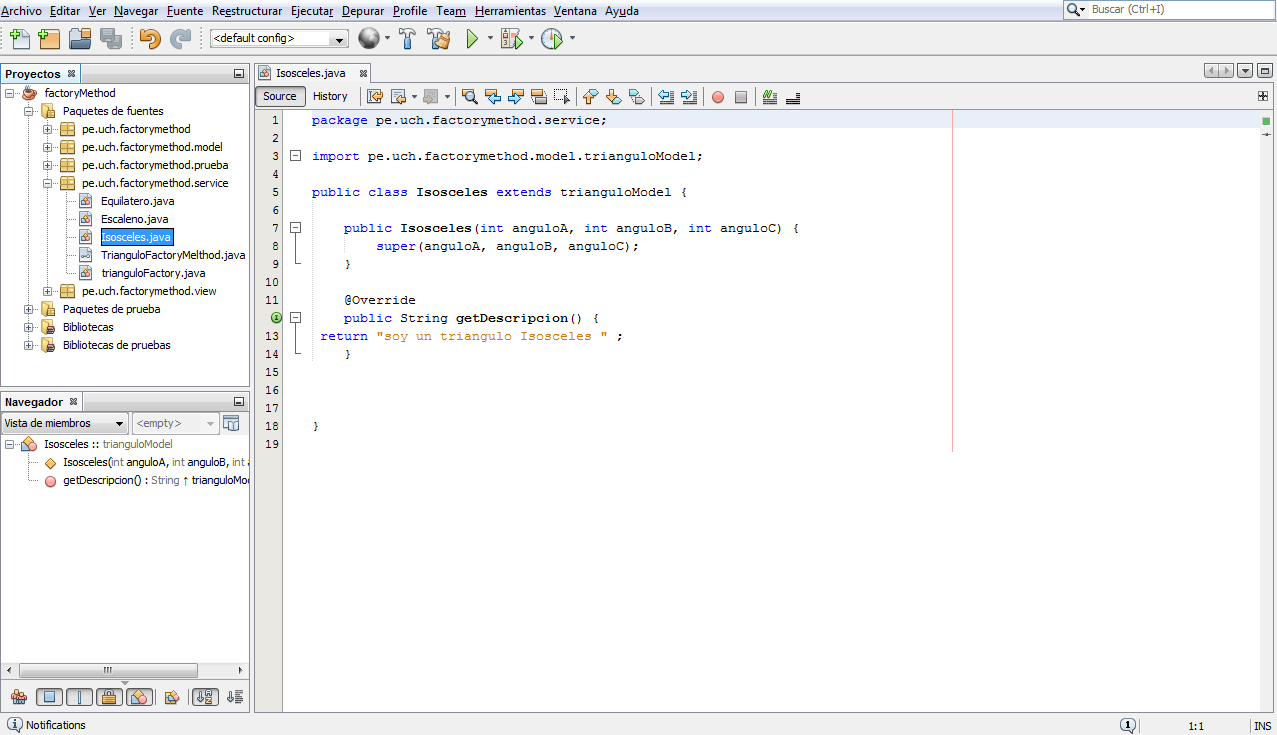
**Ejemplo 1: utilizando MÉTODO FACTORY**

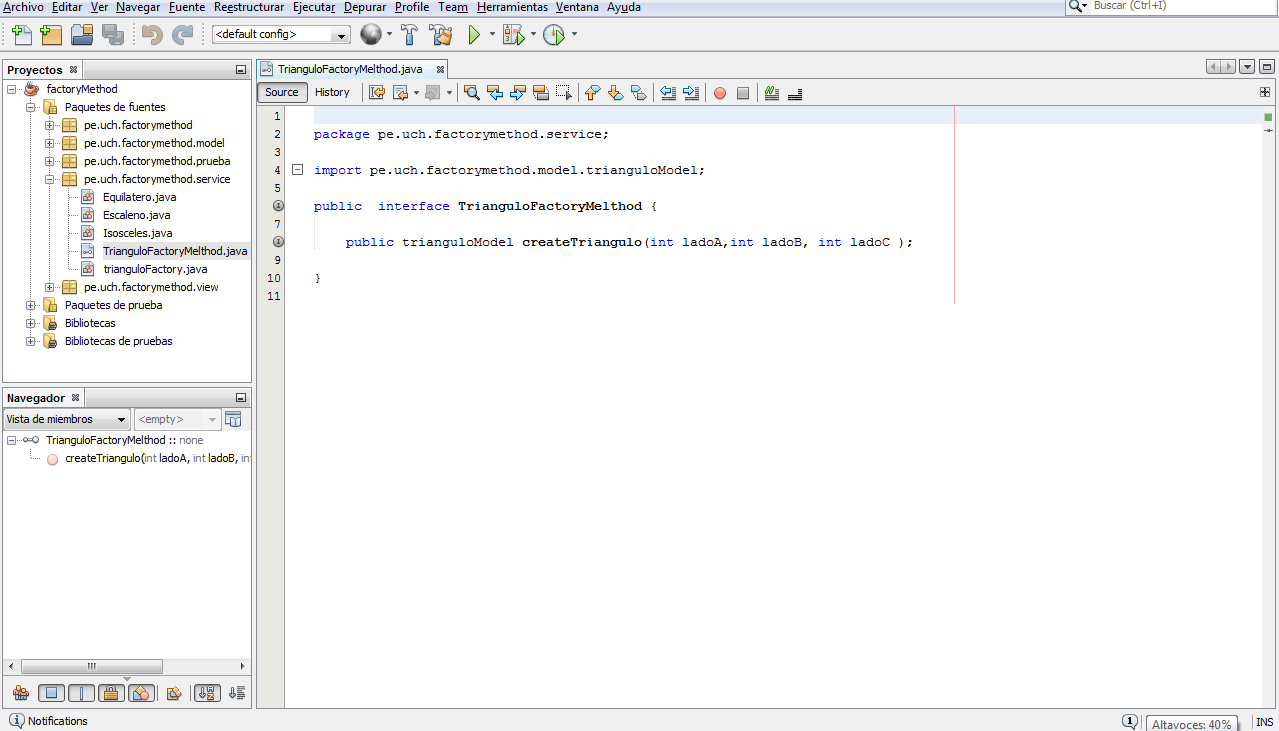
**En nuestro ejemplo tenemos una clase abstracta llamada Triangulo, de la cual heredan los 3 tipos de triangulos conocidos.**

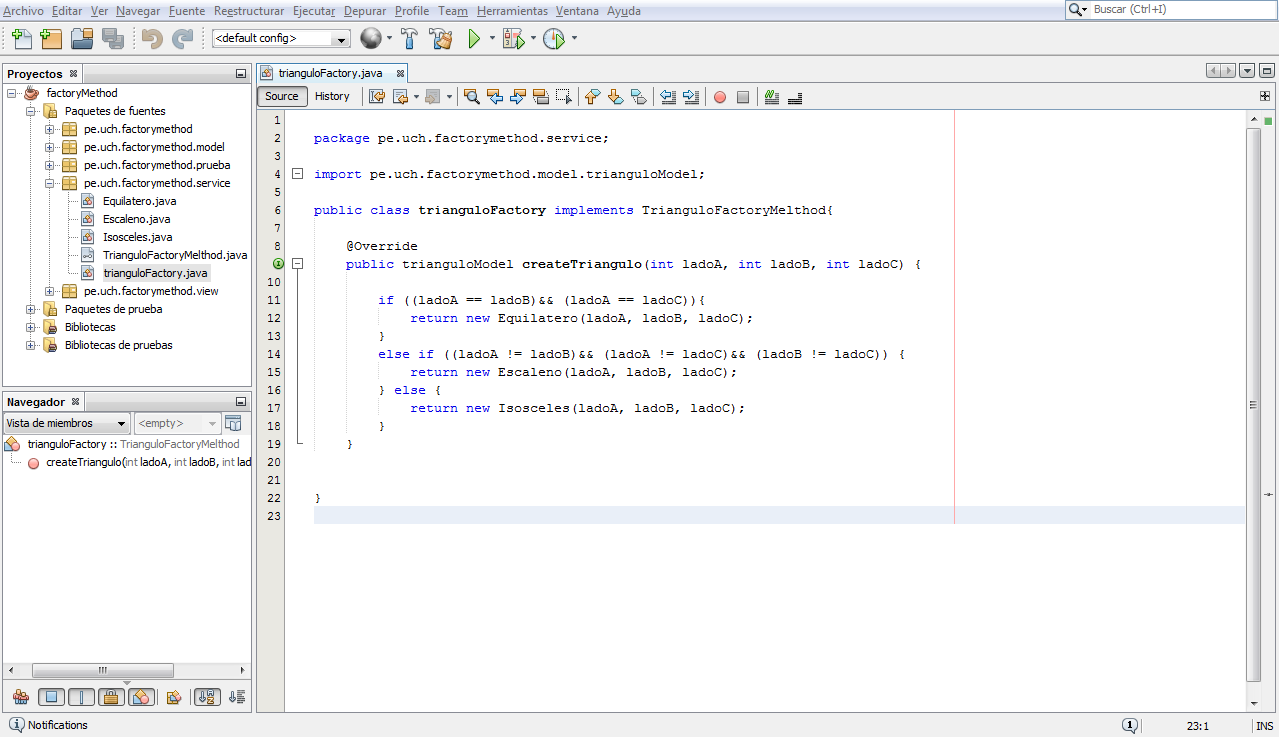
****

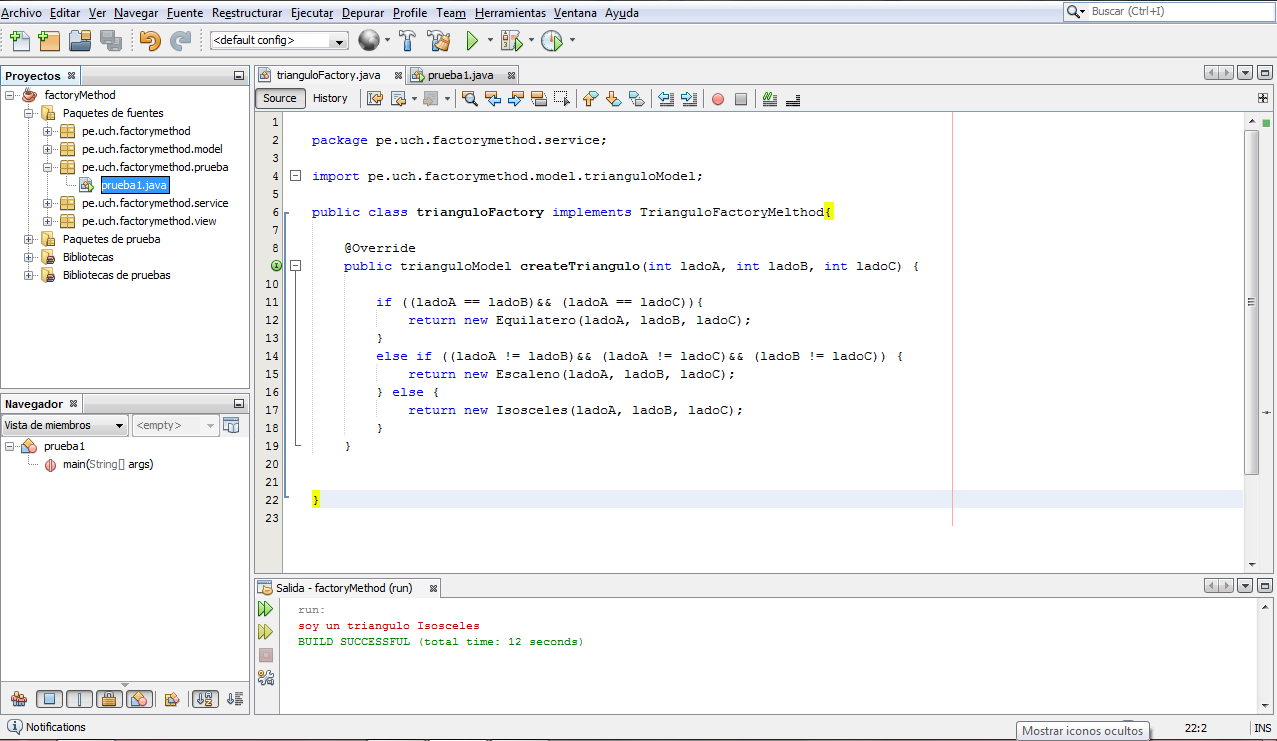


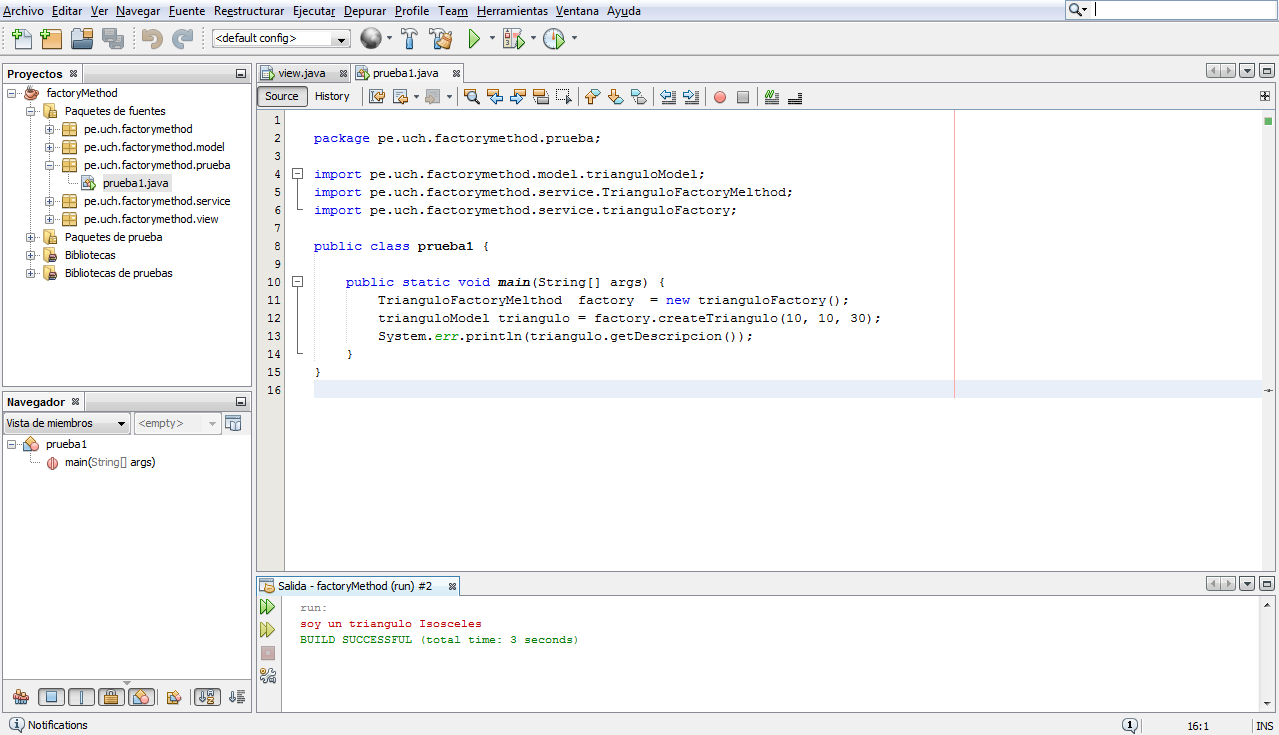


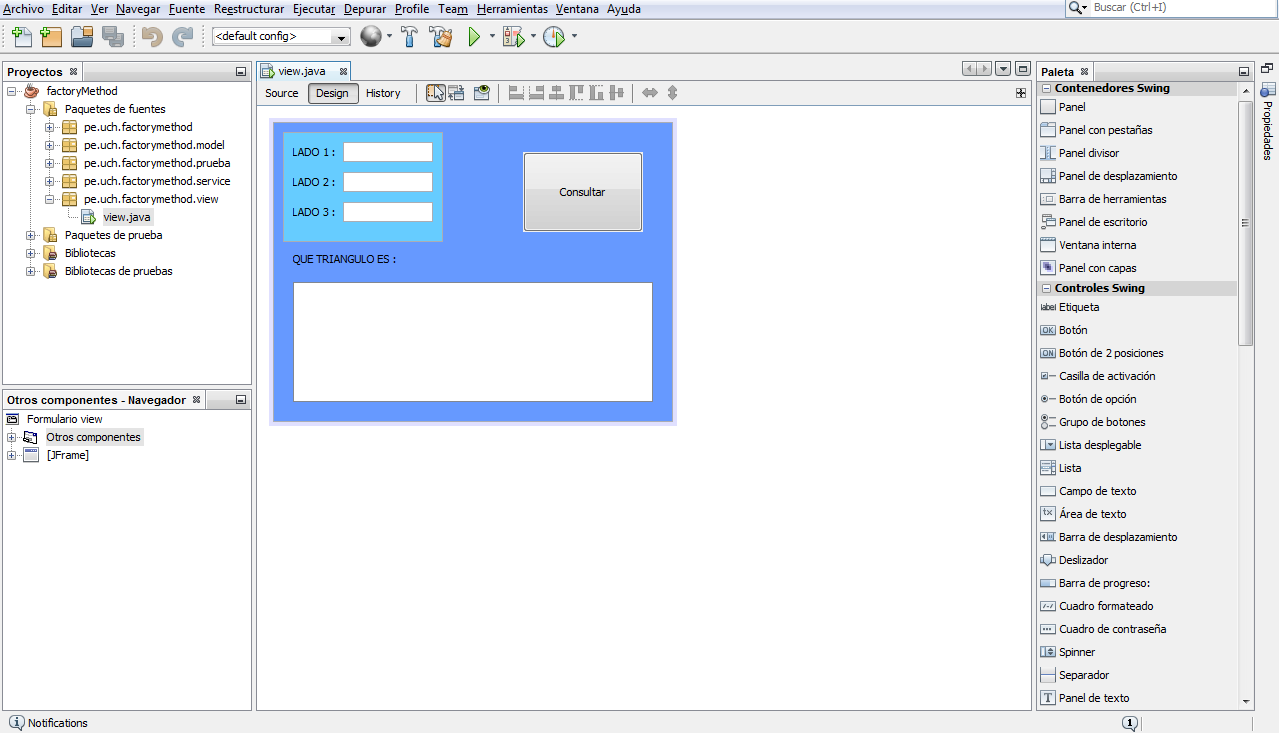


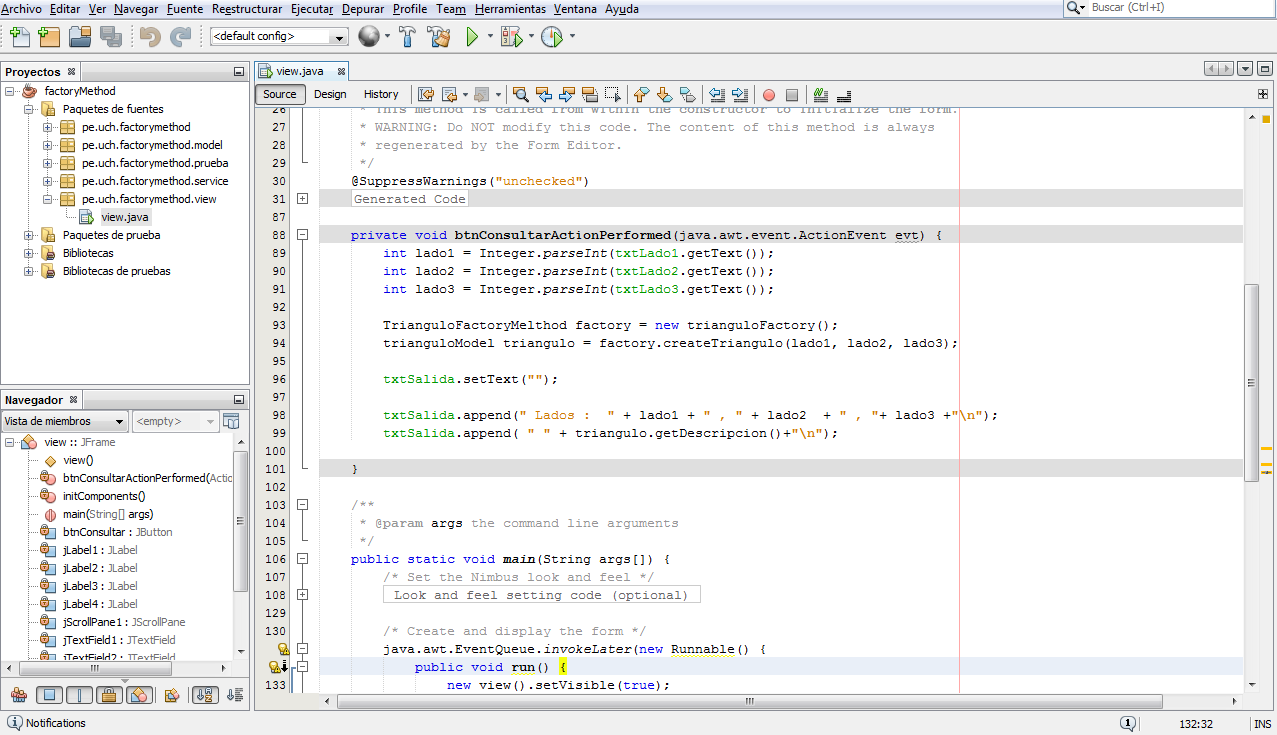


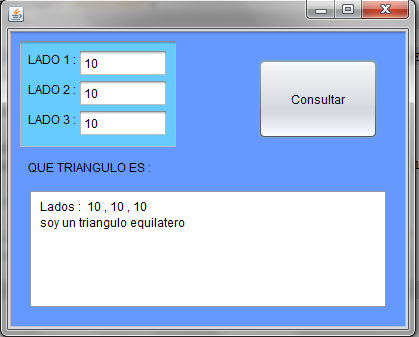












**CONCLUCIONES**

* Los patrones de diseño describen una solución estándar a problemas comunes de diseño de software. También posee una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios y son reutilizables e interaccionan con objetos.
* Simple Factory devuelve instancias de clase que tienen los mismos métodos. Pueden ser instancias de diferentes subclases derivadas, o de hecho pueden ser clases no relacionadas que comparten la misma interfaz. De cualquier manera, los métodos en esta instancia de clase son iguales y se pueden usar de forma intercambiable.
* Method Factory permite encapsular el movimiento referente a la creación de objetos. Además, hace que el diseño sea más adaptable a cambios.
* Abstract Factory promueve una interfaz que nos permite crear familias de objetos que comparten una serie de características comunes.

**RECOMENDADCIONES**

* Recomendamos analizar bien el problema a resolver antes de incluir los patrones Factory.

**REFERENCIAS**

*Factory Method - EcuRed*. *Ecured.cu*. Recuperado 1 octubre 2017, de <https://www.ecured.cu/Factory_Method>

Jain, A. (2014). what's the difference between a simple factory, a factory method design pattern, and an abstract factory? linkedin. Recuperado 1 Octubre 2017, de <https://www.linkedin.com/pulse/20140901184348-90925576-what-s-the-difference-between-a-simple-factory-a-factory-method-design-pattern-and-an-abstract-factory/>

Patrón Abstract Factory - Línea de Código. (2013). Línea de Código. Recuperado 1 octubre 2017, de <http://lineadecodigo.com/patrones/patron-abstract-factory/>

Patrón de diseño Factory - michelletorres.mx. (2017). michelletorres.mx. Recuperado 30 septiembre 2017, de <http://michelletorres.mx/patron-de-diseno-factory/>

Patrones de Fabricación: Fábricas de Objetos. (2017). Msdn.microsoft.com. Recuperado 30 septiembre 2017, de <https://msdn.microsoft.com/es-es/library/bb972258.aspx#EFAA>