Instituto Tecnológico de Costa Rica

Escuela de Computación

Ingeniería en computación

Diseño de Software

Grupo 1

Profesor: Luis Chavarría

Sistema de Amortización

Josué Arrieta Salas, 2014008153

Fernando Molina

Cartago

Lunes 30 de mayo

Índice

Contenido

[SOLID 3](#_Toc449260759)

[SRP – Single Responsability Principle 3](#_Toc449260760)

[OCP – Open/Close Principle 4](#_Toc449260761)

[LSP – Liskov Substitution Principle 5](#_Toc449260762)

[ISP – Interface Segragation Principle 7](#_Toc449260763)

[DIP – Dependency Inversion Principle 8](#_Toc449260764)

[Patrones de diseño 10](#_Toc449260765)

[MVC 10](#_Toc449260766)

[Strategy 11](#_Toc449260767)

[DTO 13](#_Toc449260768)

[Factory (creacional) 14](#_Toc449260769)

[Ejecución de funcionalidades 16](#_Toc449260770)

[Caso 1 : Funcionalidad de inversión tradicional 16](#_Toc449260771)

[Modo GUI 16](#_Toc449260772)

[Modo Consola 17](#_Toc449260773)

[Caso 2 : Funcionalidad de inversión pactada 18](#_Toc449260774)

[Modo GUI 18](#_Toc449260775)

[Modo Consola 19](#_Toc449260776)

[Caso 3 : Funcionalidad de inversión certificada 20](#_Toc449260777)

[Modo GUI 20](#_Toc449260778)

[Modo Consola 21](#_Toc449260779)

[Bitácoras 22](#_Toc449260780)

[XML 22](#_Toc449260781)

[CSV 23](#_Toc449260782)

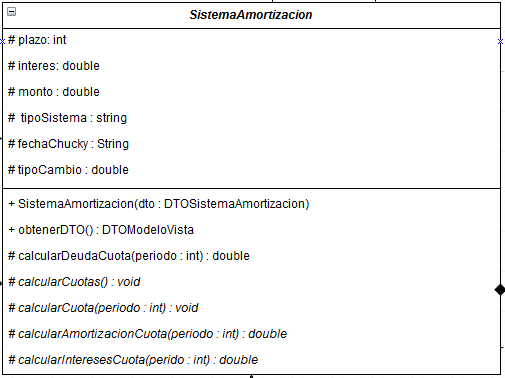
# SOLID

Para cada letra se debe:

* Screenshots con partes del diseño UML donde se cumple el principio.
* Segmentos de código donde se cumple el principio.
* Explicación o estrategias tomadas en consideración para el cumplimiento del principio.

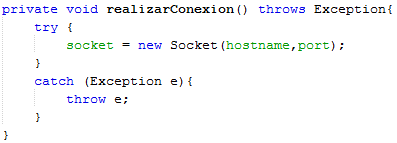
## SRP – Single Responsability Principle

Una entidad solo tiene una responsabilidad. Se deben agrupar responsabilidades similares. En el siguiente diseño se tiene la clase: SistemaArmotizacion.



SRP se ve cumplido a nivel de clase ya que esta clase solo tiene una responsabilidad: la lógica de negocios de un sistema de amortización. Solo posee atributos relacionados con esta lógica y realiza cálculos para un sistema de armotización. Solo posee una razón para cambiar ya que no mezcla responsabilidades.

A nivel de código se tiene el siguiente método:



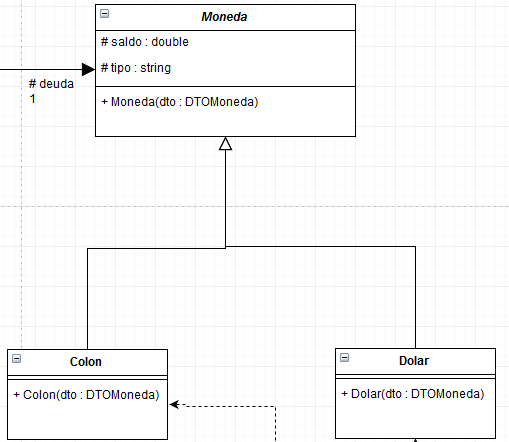
En este método SRP se ve cumplido ya que sólo posee 1 responsabilidad: realizar la conexión por medio de un socket a un back-end local. Solo posee una razón para cambiar.

Para cumplir SRP se trataron de seguir los siguientes puntos como estrategias:

* Promover siempre la alta cohesión para agrupar responsabilidades similares.
* Evitar por completo una clase Utilitaria enorme con un grado de acoplamiento altísimo.
* Asegurarse que la clase solo tiene una razón para existir, así para cambiar.
* Si un método o una clase se está haciendo excesivamente, es probable que tenga más de una responsabilidad (aunque no siempre).

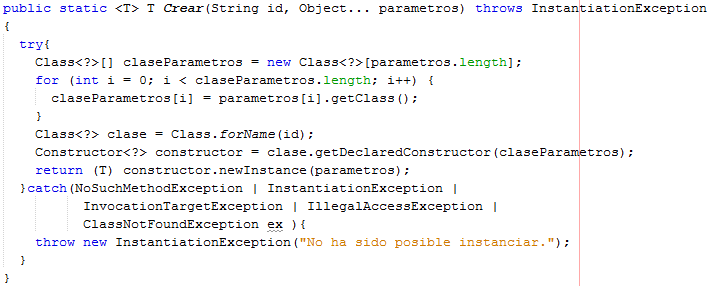
## OCP – Open/Close Principle

Las entidades deben ser abiertas para extensión y cerradas para modificación. Básicamente OCP se puede definir como: nunca modificar código existente y funcional. A nivel de diseño se tiene el siguiente ejemplo de OCP:



Digamos que se quiere extender la funcionalidad y crear un nuevo tipo de moneda, además de los ya existentes: Colon y Dolar. No es necesario cambiar el código en ninguna parte. Si no que está abierto para extensión: se agrega una nueva clase que herede de *Moneda* y se agrega el constructor particular de la nueva clase agregada.

A nivel de código de código se tiene el siguiente ejemplo donde se cumple OCP (es el método de creación de las fábricas):



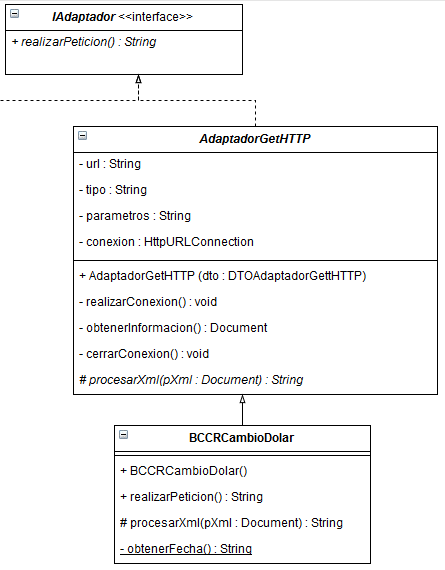
En el patrón de diseño Factory debe recibir un string o tipo, para indicar qué tipo de objeto se debe instanciar. Muchas veces se utiliza una estructura switch o if – else para la creación de objetos. Sin embargo este tipo de estructuras, si se necesita agregar la creación de un nuevo objeto, es necesario agregar un nuevo if o un nuevo case del switch. Esto viola OCP ya que se modifica código existente y dificulta el mantenimiento. Sin embargo si se utiliza la reflexión (como es este caso) tales estructuras if y switch no son necesarias, y en un futuro el código queda totalmente abierto para extensión y OCP se cumple.

Para cumplir OCP se siguieron los siguientes principios como estrategia:

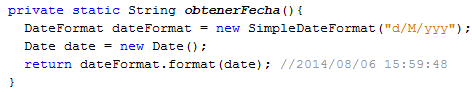
* No usar del todo estructura switch.
* No usar más de 2 if´s seguidos.
* Recurrir a la reflexión y polimorfismo.
* Agregar nuevas clases a las jerarquías para agregar nuevas funcionalidades.
* Sobrecargar sobrescribir métodos como medida para que el código este abierto a la extensión.

## LSP – Liskov Substitution Principle

Subclases deben poder ser tratadas como sus superclases. Estas deben poseer exactamente la misma interfaz. A nivel de diseño se tiene el siguiente ejemplo donde se cumple LSP (la jerarquía de adaptador para protocolo de transporte http, donde TODAS las sublcases se pueden tratar sin problema como su superclase: *AdaptadorGETHTTP*; ya que tienen la misma interfaz (conjunto de métodos públicos)):



Para el ejemplo de código se tomará un método de la misma jerarquía. Este solo está en la clase BCCRCamboiDolar y no se incumple LSP ya que el método no es público. El método se llama obtenerFecha(). Se tiene el siguiente código:



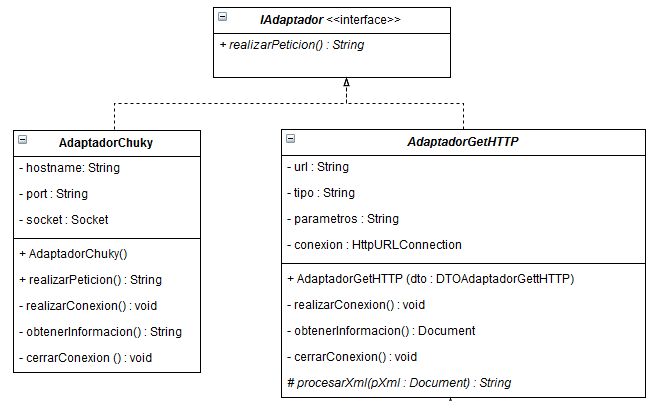
El truco en es que el método es privado, de tal manera que no puede ser accedido desde afuera y LSP se cumple. Tal método solo es llamado por adentro.

Para cumplir LSP se siguieron los siguientes puntos como estrategia:

* En una jerarquía de clases asegurarse que la superclase defina los métodos públicos de las subclases.
* Nunca definir un método públic para una subclase ya que las superclase no lo posee y se podrá violar Liskov si esto se hace.
* Si hay que usar estructuras switch o if else para identificar el tipo de datos con el que se esté trabajando es posible que se esté violando LSP.
* Nunca debe existir el caso en que una subclase es tratada diferente como las demás subclases en una jerarquía.

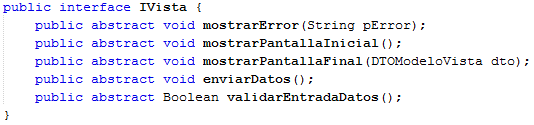
## ISP – Interface Segragation Principle

Una interfaz grande de propósito general no sirve, es mejor tener múltiples interfaces especializadas. Las interfaces tienen que ser lo más pequeñas posibles. Se tiene el siguiente diseño donde se cumple ISP:



La interface al solo poseer la firma de un método (la interface corresponde al protocolo de comportamiento para realizar peticiones específicas para el cumplimiento del patrón adaptador) cumple ISP automáticamente. Es una interfaz altamente especializada.

A nivel de código se tiene el siguiente ejemplo:



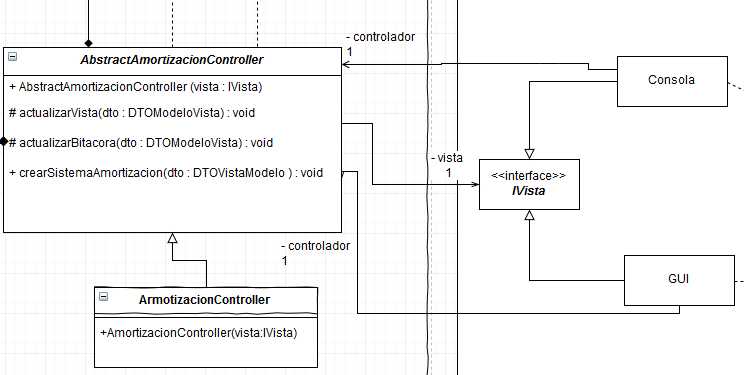
Sería la interfaz usada para el protocolo de comportamiento para las vistas. Es altamente especializada ya que solo posee firmas de métodos para la interacción con los usuarios (y validación de datos).

Para cumplir ISP se utilizaron los siguientes puntos como estrategia:

* Buscar interfaces pequeñas especializadas.
* Si hay una interfaz grande, es posible que ISP se esté violando.
* Para deshacer una interfaz grande, se puede aplicar herencia de interfaces.
* Firmas de métodos de interfaces deben estar relacionados y deben ir juntos en la misma de interfaz.

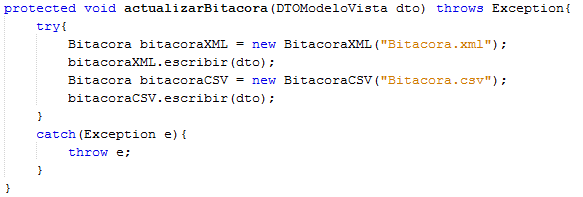
## DIP – Dependency Inversion Principle

Depender en abstracciones y no en objetos concretos. Se debe “desconectar” clases abstractas por medio de una capa de abstracción. Se tiene el siguiente diseño donde se cumple DIP:



Se usó DIP entre capas. Las interfaces no tienen referencia a un controlador concreto (y lo mismo que el controlador no tiene referencia a vistas concretas). Se depende en abstracciones (ya sea una clase abstracta o interfaz). En este caso 2 clases concretas no están relacionadas.

A nivel de código se tiene el siguiente ejemplo (dentro de la clase *AbstractArmotizacionController*):



El controlador no tiene referencia a una clase concreta, si no que a una clase *Bitacora* abstracta, de tal manera que DIP se ve cumplido. No se relacionan 2 clases concretas.

Para cumplir este principio se trataron de seguir los siguientes puntos como estrategia:

* Si se ven 2 clases concretas relacionadas, agregar una abstracción (ya sea una clase abstracta o una interface) entre ellas.
* Si se tiene una clase concreta, siempre es bueno realizar una jerarquía; para crear una superclase abstracta que funcione como abstracción.
* Ninguna variable debe tener referencia a una clase concreta, siempre a una abstracción.

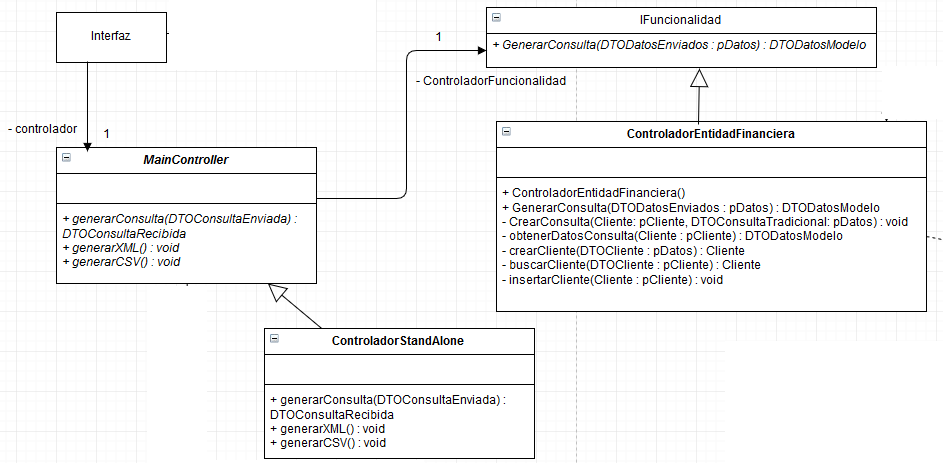
# Patrones de diseño

Para cada patrón se debe:

* Screenshots con partes del diseño UML donde se cumple el patrón.
* Segmentos de código donde se cumple el patrón.
* Explicación o estrategias tomadas en consideración para el cumplimiento del patrón.

## MVC

El patrón supone la separación de la lógica de negocios (modelo) y la interfaz; y que un controlador “amarre” dichas capas. Se tiene el siguiente diseño donde se implementa MVC:



Se puede que hay comunicación entre Interfaz y MainController. A su vez hay comunicación entre MainController y ControladorEntidadFinanciera (un subcontrolador que pertenece a la lógica de dominio y al modelo). MVC se cumple ya que la interfaz no se comunica con el modelo. El modelo es completamente agnóstico al controlador y a la interfaz. Hay una clara distinción de capas.

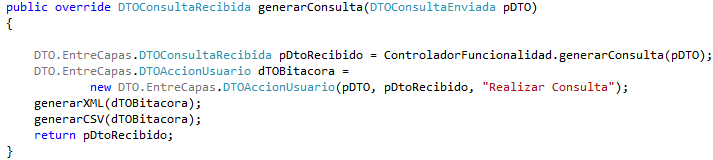
A nivel de código se tiene el siguiente ejemplo:



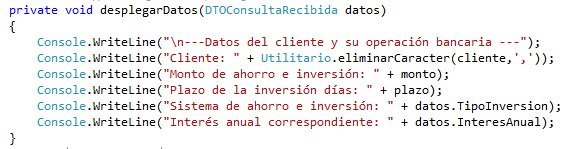
La interfaz tiene una instancia del Controlador. Posteriormente la interfaz le envía solicitudes (por medio de DTOs donde van almacenados los datos introducidos por el usuario):



El controlador procesa la petición y se comunica con el modelo:



El modelo procesa la información de lógica de negocios y devuelve el resultado (nuevamente por medio de un DTO). La vista recoge los datos y pone el resultado en pantalla. Así sería la ejecución completa del MVC:

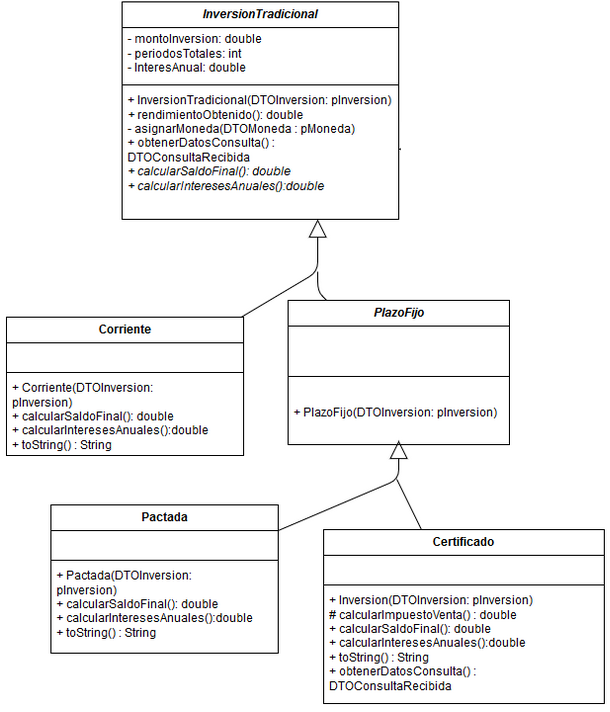


Para cumplir MVC como estrategia se tomaron los siguientes puntos:

* La capa de vista solo debe poseer la lógica de presentación.
* La vista debe enviar solicitudes al controlador.
* El controlador siempre debe determinar la naturaleza de la solicitud.
* El controlador siempre debe retornar la respuesta a la vista.
* El modelo solo debe implementar la lógica de negocios.
* La capa del modelo debe ser completamente agnóstica a las demás.

## Strategy

El patrón de diseño Strategy permite que el comportamiento del algoritmo a usar pueda ser seleccionado en tiempo de ejecución (polimorfismo). Establece una familia de algoritmos, los encapsula, y luego estos pueden ser “intercambiables”. Es necesario cumplir Liskov para que Strategy funcione. Se tiene el siguiente ejemplo de diseño:



La familia de algoritmos corresponde a la de la realización de diferentes tipos de inversiones. Debido a que se cumple LSP poseen la misma interfaz. Si se declara un objeto de la superclase *InversionTradicional* se pueden realizar diferentes tipos de inversiones (corrientes, pactadas, certificadas). Estas son “intercambiables” y son definidas en tiempo de ejecución. Inversion.calcularSaldoFinal() y dependiendo de la inversión tendrá comportamientos polimórficos. A nivel de código se tiene el siguiente ejemplo del uso del Strategy:



Dependiendo de la consulta, se procesará de manera diferente, y se obtendrán Datos Diferentes al aplicar .obtenerDatosConsulta(); produciendo algoritmos diferentes en tiempo de ejecución y polimórficos.

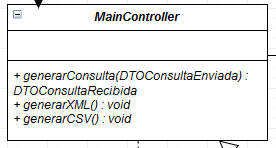
Para cumplir Strategy se siguen los siguientes puntos como estrategia:

* Los algoritmos deben ser siempre totalmente intercambiables.
* Se debe de poder definir el comportamiento deseado en tiempo de ejecución.
* Se debe cumplir Liskov en la jerarquía de algoritmos.

Si se cumplen los puntos anteriores se tendrá un buen patrón Strategy.

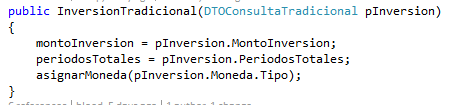
## DTO

Esta patrón supone que la comunicación entre las capas debe ser por medio de DTOs (data transfer object) que se pueden definir como “pequeños repositorios de información”. Esta capa es transversal a las demás capas. A nivel de diseño se tiene el siguiente ejemplo del cumplimiento del patrón:



La clase abstracta MainController posee un método generarConsulta que se puede observar que recibe un DTO de la interfaz, y devuelve otro DTO para la interfaz. La comunicación entre Interfaz y Controlador es por medio de DTOs.

Un ejemplo a nivel de código del uso del patrón DTO es el siguiente:



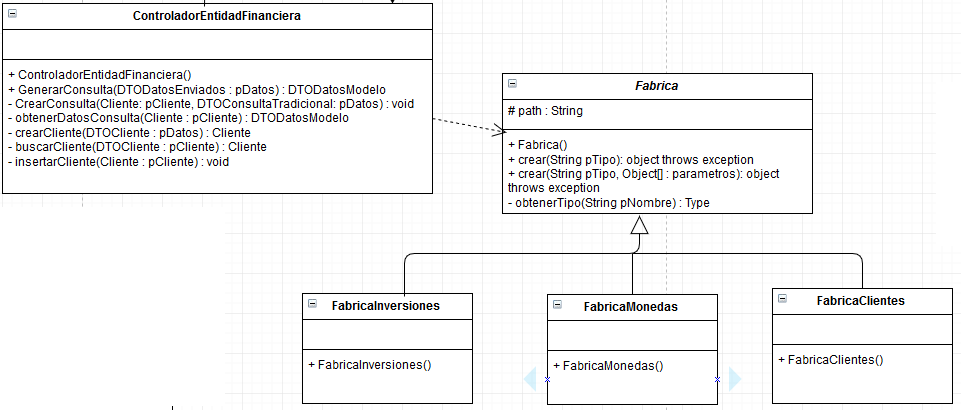
En este caso se utiliza el patrón para ser usado para la comunicación dentro del mismo modelo de dominio. En el ejemplo se tiene el constructor de una clase de dominio, que recibe un DTO y posteriormente se le “sacan” los datos y estos son asignados. Esto favorece el mantenimiento en un futuro.

Para cumplir el patrón DTO se tuvo la siguiente estrategia:

* Si un método recibe parámetros, meter un DTO (versión extremista).
* La comunicación entre capas siempre debe ser entre DTOs (ya sea como parámetros o valores de retorno).
* Para los constructores de entidades, utilizar DTO.

## Factory (creacional)

Se utiliza Factory como patrón creacional. Supone la creación de distintos objetos. Aunque hay muchas variaciones del patrón Factory, se utilizó el siguiente:



En este caso el ControladorEntidadFinanciera es el cliente. Se tiene una herencia de fábricas dependiendo de lo que se quiera crear (fabrica objetos dependiendo del atributo path y del string que reciba como parámetro). Se ha de mencionar que la Fabrica no tiene ninguna dependencia hacia alguna entidad ya que no crear objetos de tipos específicos, sino objetos del tipo Object.

A nivel de código se tiene los métodos de creación:



Utiliza reflexión para cumplir el OCP. Devuelve un objeto del tipo correspondiente y posteriormente habrá que hacerle downcasting (que no tiene ningún efecto negativo, ya que al momento de su creación uno sabe que es lo que se quiere instanciar, y el downcasting se hace inmediatamente).

Para cumplir con el patrón de diseño Factory se utilizó la siguiente estrategia:

* En los métodos de creación siempre utilizar reflexión.
* Devolver tipos object genéricos para evitar la dependencia entre el Factory y las otras clases.
* Dividir la jerarquía Factory por medio del path de objetos a crear (Ej.: “SistemasInversiones.Modelo.Clientes.” de esta manera solo se crear clientes en la fábrica clientes).
* Se aprovecha el patrón de diseño cuando se tiene una jerarquías de objetos de los objetos a crear.

# Ejecución de funcionalidades

Se debe incluir:

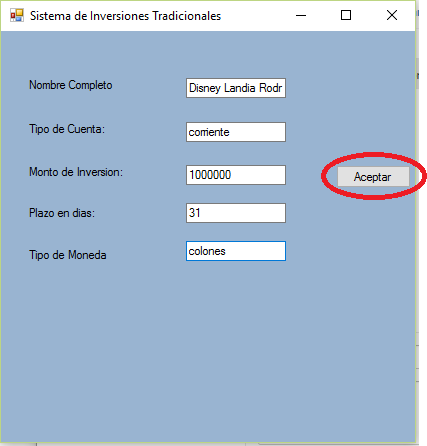
* Screenshot de la funcionalidad que se está ejecutando al usuario.
* Screenshot del resultado de la funcionalidad.
* Se debe hacer para ambas interfaces (GUI y Consola).
* Además se debe mostrar un Screenshot con la información histórica del contenido del XML y otro para la información histórica del CSV.

Se probarán todas las funcionalidades. Estas corresponden a Consultas de inversiones: corrientes, certificadas y pactadas. A continuación se muestran dichas pruebas.

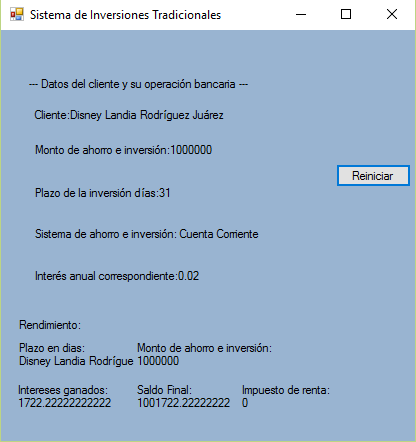
## Caso 1 : Funcionalidad de inversión tradicional

### Modo GUI

Se tienen los siguientes datos de entrada:

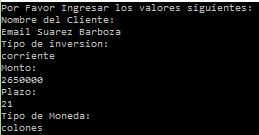


Se obtuvieron los siguientes resultados (correctos):

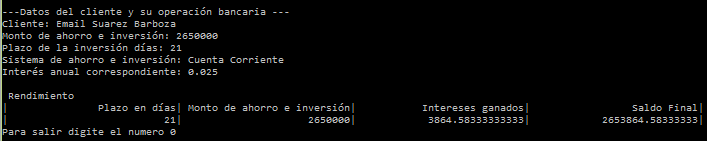


### Modo Consola

Se tienen los siguientes datos de entrada



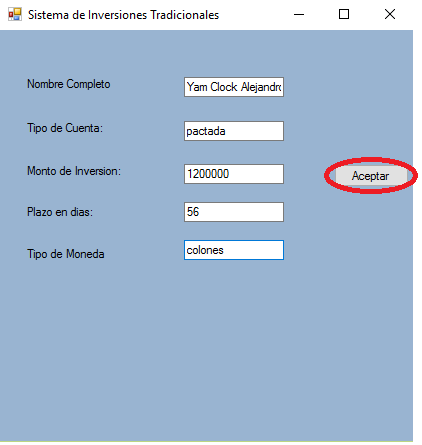
Se obtienen los siguientes resultados (correctos):



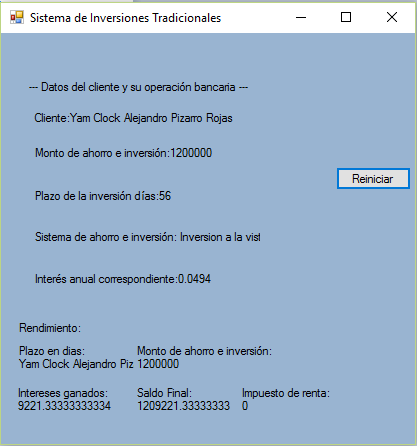
## Caso 2 : Funcionalidad de inversión pactada

### Modo GUI

Se tienen los siguientes datos de entrada:

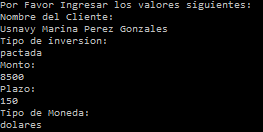


Se obtuvieron los siguientes resultados (correctos):

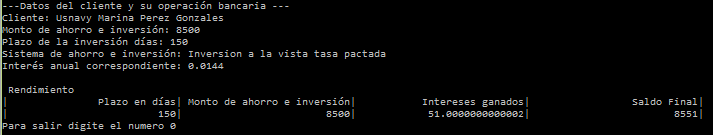


### Modo Consola

Se tienen los siguientes datos de entrada:



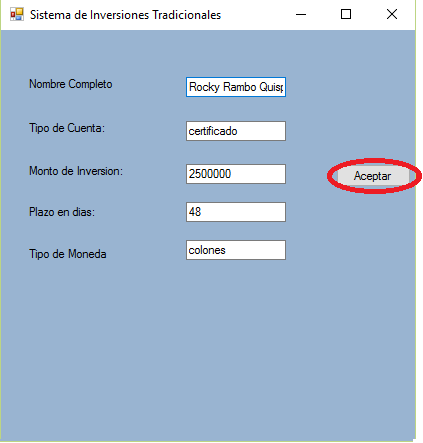
Se obtienen los siguientes resultados (correctos):



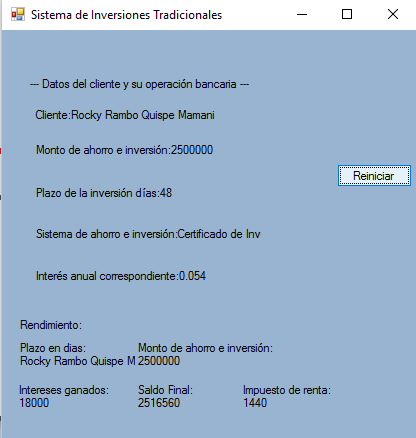
## Caso 3 : Funcionalidad de inversión certificada

### Modo GUI

Se tienen los siguientes datos de entrada:

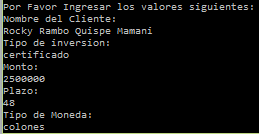


Se obtienen los siguientes resultados (correctos):

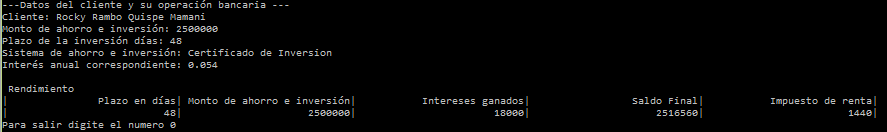


### Modo Consola

Se tienen los siguientes datos de entrada:



Se obtienen los siguientes resultados (correctos):



## Bitácoras

Luego de realizar las pruebas anteriores se tienen las siguientes bitácoras (correctas):

### XML

<?xml version="1.0" encoding="utf-8"?>

<BitacoraFuncionalidadesUsadas>

<Funcionalidad>

<NombreCompleto>Disney Landia Rodriguez Juarez</NombreCompleto>

<TipoCliente>PersonaCorriente</TipoCliente>

<MontoInversion>1000000</MontoInversion>

<PeriodosTotales>31</PeriodosTotales>

<Tipo>Colones</Tipo>

<TipoInversion>Corriente</TipoInversion>

<NombreFuncionalidad>Realizar Consulta</NombreFuncionalidad>

<InteresAnual>0.02</InteresAnual>

<InteresGanado>1722.22222222222</InteresGanado>

<ImpuestoRenta>0</ImpuestoRenta>

<Saldo>1001722.22222222</Saldo>

<TipoInversion>Cuenta Corriente</TipoInversion>

</Funcionalidad>

<Funcionalidad>

<NombreCompleto>Yam Clock Alejandro Pizarro Rojas</NombreCompleto>

<TipoCliente>PersonaCorriente</TipoCliente>

<MontoInversion>1200000</MontoInversion>

<PeriodosTotales>56</PeriodosTotales>

<Tipo>Colones</Tipo>

<TipoInversion>Pactada</TipoInversion>

<NombreFuncionalidad>Realizar Consulta</NombreFuncionalidad>

<InteresAnual>0.0494</InteresAnual>

<InteresGanado>9221.33333333334</InteresGanado>

<ImpuestoRenta>0</ImpuestoRenta>

<Saldo>1209221.33333333</Saldo>

<TipoInversion>Inversion a la vista tasa pactada</TipoInversion>

</Funcionalidad>

<Funcionalidad>

<NombreCompleto>Rocky Rambo Quispe Mamani</NombreCompleto>

<TipoCliente>PersonaCorriente</TipoCliente>

<MontoInversion>2500000</MontoInversion>

<PeriodosTotales>48</PeriodosTotales>

<Tipo>Colones</Tipo>

<TipoInversion>Certificado</TipoInversion>

<NombreFuncionalidad>Realizar Consulta</NombreFuncionalidad>

<InteresAnual>0.054</InteresAnual>

<InteresGanado>18000</InteresGanado>

<ImpuestoRenta>1440</ImpuestoRenta>

<Saldo>2516560</Saldo>

<TipoInversion>Certificado de Inversion</TipoInversion>

</Funcionalidad>

<Funcionalidad>

<NombreCompleto>Email Suarez Barboza</NombreCompleto>

<TipoCliente>PersonaCorriente</TipoCliente>

<MontoInversion>2650000</MontoInversion>

<PeriodosTotales>21</PeriodosTotales>

<Tipo>Colones</Tipo>

<TipoInversion>Corriente</TipoInversion>

<NombreFuncionalidad>Realizar Consulta</NombreFuncionalidad>

<InteresAnual>0.025</InteresAnual>

<InteresGanado>3864.58333333333</InteresGanado>

<ImpuestoRenta>0</ImpuestoRenta>

<Saldo>2653864.58333333</Saldo>

<TipoInversion>Cuenta Corriente</TipoInversion>

</Funcionalidad>

<Funcionalidad>

<NombreCompleto>Usnavy Marina Perez Gonzales</NombreCompleto>

<TipoCliente>PersonaCorriente</TipoCliente>

<MontoInversion>8500</MontoInversion>

<PeriodosTotales>150</PeriodosTotales>

<Tipo>Dolares</Tipo>

<TipoInversion>Pactada</TipoInversion>

<NombreFuncionalidad>Realizar Consulta</NombreFuncionalidad>

<InteresAnual>0.0144</InteresAnual>

<InteresGanado>51.0000000000002</InteresGanado>

<ImpuestoRenta>0</ImpuestoRenta>

<Saldo>8551</Saldo>

<TipoInversion>Inversion a la vista tasa pactada</TipoInversion>

</Funcionalidad>

<Funcionalidad>

<NombreCompleto>Rocky Rambo Quispe Mamani</NombreCompleto>

<TipoCliente>PersonaCorriente</TipoCliente>

<MontoInversion>2500000</MontoInversion>

<PeriodosTotales>48</PeriodosTotales>

<Tipo>Colones</Tipo>

<TipoInversion>Certificado</TipoInversion>

<NombreFuncionalidad>Realizar Consulta</NombreFuncionalidad>

<InteresAnual>0.054</InteresAnual>

<InteresGanado>18000</InteresGanado>

<ImpuestoRenta>1440</ImpuestoRenta>

<Saldo>2516560</Saldo>

<TipoInversion>Certificado de Inversion</TipoInversion>

</Funcionalidad>

</BitacoraFuncionalidadesUsadas>

### CSV