PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 Programación Avanzada (II/2017)

Tarea 1

1. Objetivos

- Aplicar conceptos de programación orientada a objetos (OOP) para modelar y resolver un problema.
- Trabajar con archivos de texto para leer y procesar datos.
- Desarrollar algoritmos para la resolución de problemas complejos.
- Diseñar e implementar protocolos para el intercambio de información.

2. Introducción

En los últimos años han surgido una gran cantidad de monedas no convencionales: las *criptomonedas*. De esta manera, nació una nueva forma de hacer compras y/o transacciones de manera segura y descentralizada, pudiendo prescindir de cualquier tipo de institución malvada que se apodera de nuestro dinero (más conocidas como intermediarios).

La empresa `DCCapital` quiere controlar todos los mercados y para ello necesita crear un software que permita realizar intercambios entre distintos mercados de criptomonedas y monedas convencionales para almacenar, manejar y transar dinero entre clientes. Cada mercado cuenta con usuarios que pueden realizar acciones para vender o comprar monedas.

Para realizar este trabajo, `DCCapital` había decidido pedirle ayuda al malvado Dr. Mavrakis, pero éste pereció de forma trágica, por lo que la ~~compasiva~~ Reina Barrios tomó su puesto. Ella, al no tener tiempo, le delegó el trabajo a los alumnos del curso Programación Avanzada, quienes deberán crear una aplicación llamada `DCCapital Exchange`. Para desarrollar lo anterior, los alumnos deberán aplicar OOP, es decir, el uso de **clases** y la relación correcta entre ellas.

3. DCCapital Exchange

La aplicación `DCCapital Exchange` debe permitirles a los usuarios ejecutar órdenes y transacciones en múltiples mercados. Un mercado representa un espacio virtual en donde se intercambian **exactamente dos divisas distintas**. Una de estas divisas es aquella que es comprada y vendida en el mercado, de la misma forma en que un producto es vendido y comprado; y otra divisa es aquella que se usa como moneda de cambio para comprar y vender.

Estas transacciones funcionan a partir de órdenes, que son acciones que un usuario realiza indicando que desea vender o comprar una cierta cantidad de dinero. Cuando los intereses de compra de un usuario corresponden con los intereses de venta de otro, se produce un *match*. Es importante notar que **los usuarios no intercambian directamente sus bienes, sino que dejan ofertas y es el sistema de cada mercado**

que se encarga de igualar estas ofertas. De esta forma, cuando un *match* ocurre, el sistema debe realizar automáticamente la transacción que satisfaga los intereses de los usuarios que generaron la orden.

Además, cada vez que se realiza una transacción, el mercado le cobra una comisión a ambas partes (comprador y vendedor). Cada mercado tiene una tasa asociada de comisión, que es un número real entre 0 y 1, la cual también puede ser vista como un porcentaje. La comisión cobrada se calcula multiplicando la tasa del mercado por el dinero que el usuario recibiría si no se le cobrara comisión.

Al abrir la aplicación, los usuarios tendrán dos opciones: registrarse con una cuenta nueva o iniciar sesión en una cuenta creada anteriormente. Una vez que un usuario ha iniciado sesión, se deberá desplegar una lista ordenada de acciones posibles. El usuario deberá poder seleccionar alguna de éstas y el software debe ser capaz de realizarlas. Una vez realizada alguna de estas acciones, el usuario deberá poder volver al menú para realizar más acciones si así lo quisiera.

Además, se debe permitir que un usuario cierre sesión y luego otro usuario inicie sesión sin tener que reiniciar la aplicación. Cuando esto ocurra, la aplicación debe tomar en cuenta las acciones realizadas por usuarios anteriores. En otras palabras, todo lo hecho por usuarios que han usado la aplicación puede afectar el estado de los mercados y esto debe ser considerado cuando un nuevo usuario inicie sesión.

La interacción con el sistema será a través de la consola, por lo que deberás desarrollar un menú amigable para los clientes. Todas las monedas son representadas por un símbolo y un nombre.

4. Glosario [15 %]

- `Order`: Término general para referirse a una intención concreta de venta (*ask*) o de compra (*bid*) realizada por un usuario, en un mercado determinado. Esta orden se mantiene activa hasta que otro usuario decida ofrecer una contraparte en el mercado, en cuyo caso se producirá un *match* entre ambas órdenes. Cuando esto ocurre, diremos que la orden fue *ejecutada*. Además, un usuario podrá cancelar una orden, siempre y cuando ésta no haya sido ejecutada.
- `Ask`: Si imaginamos que `NEC` es una moneda, entonces *ask* es una orden de venta que especifica cuánto `NEC` se quiere vender (e.g. 2 `NEC`) y a cuánto dinero (en otra moneda) por unidad de `NEC`.
- `Bid`: Si imaginamos que `NEC` es una moneda, entonces *bid* es una orden de compra que especifica cuánto `NEC` se quiere comprar (e.g. 3 `NEC`) y cuánto dinero (en otra moneda) por unidad de `NEC`.
- `Match`: Evento que ocurre cuando se iguala un *ask* y un *bid* (es decir, ambos usuarios ofrecen el mismo precio en su oferta), y se realiza el traspaso de monedas-dinero correspondientes hacia ambos *traders*.
- `Symbol`: Símbolo (o identificador) de tres caracteres asociado a una moneda. Por ejemplo, el símbolo de "pesos chilenos" es `CLP` —que viene de ***chi**lean **p**eso*—, mientras que el símbolo de *nebcoins* es `NEC`.
- `DCC`: Símbolo de **D**CC **C**rypto**C**oin, que es la moneda oficial de `DCCapital Exchange`.
- `Ticker`: Identificador del mercado que está compuesto de **dos *currencies***. El primero de ellos representa el *currency* que se transará en ese mercado, mientras que la otra indica cuál es la divisa utilizada para realizar estos intercambios. Por ejemplo: `NECCLP` es posible separarlo en `NEC` y en `CLP`. Por lo tanto, en este mercado, los usuarios venden y compran `NEC`, entregando pesos chilenos a cambio.
- Valor actual (de una divisa): Se define como el último valor para el cual hubo *match* en un mercado. En un mercado, se mide el valor actual de la divisa transada, en términos de la divisa con la cual se transa. Por ejemplo, en el mercado `NECCLP`, el valor actual del `NEC` se mide en `CLP`.

5. Usuarios [9 %]

Todos los usuarios poseen un nombre de usuario **único**, un nombre, un apellido, una fecha de nacimiento, y un balance de sus distintas *currencies*. El *exchange* define tres tipos de usuarios:

- `Underaged`: Corresponden a todos los usuarios que tengan menos de 18 años. A diferencia de los `Traders` e `Investors`, los `Underaged` no pueden realizar *orders*. En otras palabras, no tienen el poder de hacer *ask* o *bid*. Sin embargo, sí pueden registrarse a los mercados, que es necesario para que ellos puedan ver la información de los mercados.
- `Trader`: Es el usuario común de `DCCapital Exchange` que ya ha cumplido la mayoría de edad. Pueden colocar órdenes (*ask* y *bid*) en mercados, y además disfrutan de todos los permisos que los `Underaged` tienen. Los usuarios pertenecientes a esta categoría pueden hacer un máximo de 15 *orders* diarias y pueden tener un máximo de 5 órdenes activas de forma concurrente. Además, pueden acceder a un registro de las *orders* realizadas en los últimos 7 días.
- `Investor`: Estos usuarios, a diferencia de los anteriores, tienen otros beneficios: menor comisión por transacción (la mitad de la tasa), no tener límites diarios de *orders*, no tener límites de *orders* concurrentes y es el ganador en caso de empate en un *match*. También poseen permisos especiales: pueden ver su balance histórico y las *orders* históricos (desde el inicio de los tiempos) de cualquier mercado. Un `Trader` puede cambiar a `Investor` pagando una cuota única de 300.000 `DCC`.

6. Funcionalidades [32 %]

- Antes de iniciar sesión en el sistema:
  - Sistema de ingreso y registro: para poder utilizar la aplicación de *exchange*, es necesario estar registrado en la aplicación. Al registrarse, los usuarios deben indicar su nombre de usuario, nombre, apellido y fecha de nacimiento. Si el nombre de usuario ya existe, el sistema debe informar y no crear el usuario nuevo. Todos los usuarios nuevos que cumplan con la mayoría de edad reciben una cantidad inicial de 100.000 `DCC`, y también 50.000 de las monedas respectivas (*i.e.* las dos divisas que componen un mercado) cuando se ingresa a un mercado específico, obsequio de parte de Nebil, el magnate benevolente.

    Cuando un usuario registrado ingresa al sistema sólo debe escribir su nombre de usuario. El sistema debe informarle sobre el estado de sus *orders* realizadas y de su saldo actual.
- En el sistema: una vez iniciada sesión, se deben desplegar las siguientes opciones:
  - Lista de mercados disponibles.
  - Lista de mercados en los que el usuario está registrado.
  - Registrarse en un mercado específico.
  - Lista de *orders*: Se deberá dar la opción de desplegar en consola, de forma **ordenada**[1], una lista de todos los *orders* del día, una fecha específica, entre dos fechas o de un mercado en particular.
  - Lista de *orders* activas: Se deberá dar la opción de desplegar en consola de forma una lista de todos los *orders* que están activos (no han hecho *match* todavía).
  - Ingresar *ask*: Una vez registrado en un mercado, se deberá dar la opción de ingresar al sistema una nueva oferta de venta con todos los datos correspondientes.
  - Ingresar *bid*: Una vez registrado en un mercado, se deberá dar la opción de ingresar al sistema una nueva oferta de compra con todos los datos correspondientes.
  - Desplegar información de un mercado: Se deberá ofrecer la opción de mostrar la información relevante de todos los mercados o de alguno en específico. Estos son: el número de órdenes, el *spread* actual —que es la diferencia entre el menor precio de las órdenes activas de venta y el mayor precio de las órdenes activas de compra—, el volumen acumulado de *asks*, el volumen

[1] Apoveche el uso de `{ }.format`.

acumulado de *bids*[2], el ask_{best} y el bid_{best}. El ask_{best} y el bid_{best} corresponden respectivamente al mejor precio de venta y compra del día actual. El ask_{best} se calcula como el menor precio de venta y el bid_{best} como el mayor precio de compra.

- Banco: En esta opción, el usuario puede enviar dinero desde un usuario a otro usuario. Al elegir esta última opción, `DCCapital Exchange` obtiene el **5 %** del monto a cambiar en forma de comisión. Esta comisión es guardada dentro del mercado respectivo.
- Salir del sistema: Cuando un usuario desee dejar de utilizar el sistema, el programa debe cerrarse y todos los cambios realizados deben actualizarse en los archivos de texto correspondientes. Esto quiere decir guardar las orders que haya realizado el usuario.
- Consultas:
  - Información de usuarios: Despliega la información básica de todos los usuarios registrados en el sistema, es decir, nombre, apellido, agrupados por tipo de usuario.
  - Historial de *matches*: Despliega el historial de todos los *matches* realizados hasta la fecha.
  - Consultar información de una moneda: despliega el nombre y código de la moneda. Además, debe indicar: cantidad de mercados con órdenes activas, precio actual en cada uno de los mercados abiertos y fecha de la última orden en cada mercado.

6.1. Ejemplo: funcionamiento del mercado `NECCLP`

A modo de ejemplo, imaginemos que el *exchange* abre el flamante mercado `NECCLP`. A partir del *ticker*, es posible deducir que, en este mercado, los usuarios venderán y comprarán *nebcoins* (de símbolo `NEC`), mediante el intercambio de pesos chilenos. Veamos (en la figura 1) una representación gráfica de las órdenes hasta ahora.

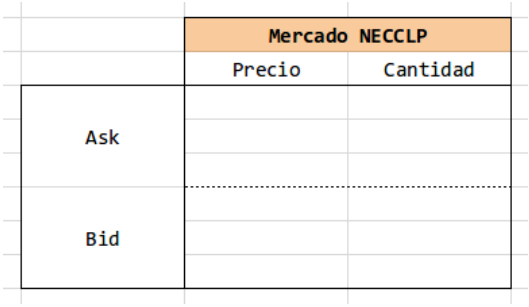

Figura 1: El mercado `NECCLP` abre sus puertas

Como es de esperar, no hay órdenes disponibles, puesto que el mercado fue recién creado. Imaginemos ahora que un *trader* que tiene muchos *nebcoins* decide colocar una oferta de venta (i.e. *ask*) de 2 `NEC`, buscando venderlos a 100 mil pesos **cada uno**. Veamos cómo se actualiza nuestra tabla, en la figura 2.

La orden de venta aparece ahora como orden activa del mercado `NECCLP`. Ahora, supongamos que dos usuarios más deciden realizar una nueva orden. El primero también quiere vender *nebcoins*, pero pidiendo un mejor precio: 3 `NEC` a 95 mil pesos cada una. El segundo usuario quiere obtener *nebcoins*; por lo tanto realiza un *bid* de 1.5 `NEC`, ofreciendo 90 mil pesos. En la figura 3, veamos nuevamente cómo sigue la tabla.

Notemos que el mercado recibe ambas órdenes. Ahora tenemos, entonces, tres órdenes activas: dos *ask* y un *bid*. Notemos que ellas se ordenan en la tabla, para conocer rápidamente el mejor precio de venta (95 mil) y el mejor precio de compra (90 mil). Veamos, además, ninguna de ellas ha hecho *match*, puesto que los precios no calzan... todavía. Imaginemos ahora que un usuario decide hacer un *bid* para comprar 3 `NEC` en 95 mil pesos cada uno. Observemos qué ocurre, con la figura 4.

[2] "Volumen acumulado" se refiere a la cantidad total de *currency* que está actualmente en *asks* y *bids*, respectivamente.

	Mercado NECCLP	
	Precio	Cantidad
Ask		
	100.000 CLP	2 NEC
Bid		

Figura 2: La primera oferta de NECCLP ha arribado

	Mercado NECCLP	
	Precio	Cantidad
Ask	100.000 CLP	2 NEC
	95.000 CLP	3 NEC
	90.000 CLP	1.5 NEC
Bid		

Figura 3: Dos ofertas más llegan al mercado

	Mercado NECCLP	
	Precio	Cantidad
Ask		
	100.000 CLP	2 NEC
	90.000 CLP	1.5 NEC
Bid		

Figura 4: El momento histórico del primer *match*

¡Tenemos el primer *match*! El usuario que había puesto el *ask* de 3 NEC a 95 mil pesos recaudará 285 mil pesos (menos la comisión del mercado), mientras que el usuario que (recién) realizó el *bid* recibirá los 3 NEC (así es, menos la comisión). Ahora, supongamos, que un mismo usuario coloca dos *bid*: uno de 1.5 NEC a 98 mil pesos y otro de 1.2 NEC a 100 mil pesos. Observemos, en la figura 5 qué sucede con el mercado.

Nuevamente, tenemos otro *match*. Sin embargo, en esta ocasión, obtuvimos un ***match* parcial**, puesto que al usuario que había realizado el *ask* de 2 NEC, sólo le igualaron su oferta de forma **parcial**. Por lo tanto, recibirá 120 mil pesos (menos la comisión) y todavía tendrá un *ask* parcial activo en el mercado. Por otra parte, el usuario del *bid* recibió los que 1.2 NEC (menos la comisión) que había solicitado. Veamos, en la figura 6, qué ocurre si un usuario hace una oferta al mismo precio, pero de mayor cantidad a una orden activa. Siguiendo con el ejemplo, digamos, entonces, que un usuario decide poner un *ask* de 2 NEC a 98 mil pesos cada uno. Este monto de 2 NEC es superior a la cantidad que ofrece el *bid* (actualmente de 1.5 NEC) de 98 mil pesos.

Vemos que el *match* se realiza en 98 mil pesos. Pero como el usuario que vendió, ofreció 0.5 NEC más (la

	Mercado NECCLP	
	Precio	Cantidad
Ask		
	100.000 CLP	0.8 NEC
Bid	98.000 CLP	1.5 NEC
	90.000 CLP	1.5 NEC

Figura 5: Los efectos de un *match* parcial

	Mercado NECCLP	
	Precio	Cantidad
Ask	100.000 CLP	0.8 NEC
	98.000 CLP	0.5 NEC
	90.000 CLP	1.5 NEC
Bid		

Figura 6: Los efectos de ofrecer una cantidad mayor a lo solicitado

diferencia entre 2 y 1.5) que la oferta de *bid* existente, este remanente queda de forma automática ingresado como una nueva orden de *ask* a nombre del mismo usuario que hizo la venta.

Finalmente, supongamos que otro usuario (que piensa que el valor de los *nebcoins* pronto irán al alza) hace un *bid* para comprar 4 `NEC` a 105 mil pesos cada uno. Veamos cómo se actualiza el estado de las órdenes, señalado en la figura 7.

	Mercado NECCLP	
	Precio	Cantidad
Ask		
	105.000 CLP	2.7 NEC
Bid	90.000 CLP	1.5 NEC

Figura 7: Los *nebcoins* están al alza

La orden de compra de 4 `NEC` hace *match* de forma inmediata con los *ask* existentes a precios menores (98 y 100 mil pesos) puesto que están a un precio menor del ofertado, que era de 105 mil pesos. En este caso, cuando el precio del *ask* y el *bid* no son iguales, es fundamental entender que se **respetan los precios que fueron puestos en primer lugar**. En otras palabras, el primer *match* se hace a 98 mil pesos, mientras que el segundo se hace a 100 mil pesos, **y no ambos a 105 mil pesos**. Además, sobran 2.7 `NEC` que no hicieron *match*, por lo que, de forma análoga a lo que sucedió anteriormente, este monto sobrante queda ingresado como oferta de compra.

Lo que ocurrió hubiese sido equivalente a que el usuario —que hizo el *bid* de 4 `NEC`— hubiese colocado:

1. una oferta de compra de 0.5 `NEC` a 98 mil pesos,
2. una oferta de compra de 0.8 `NEC` a 100 mil pesos,
3. finalmente, una oferta de compra de 2.7 `NEC` a 105 mil pesos.

7. Precisión [7 %]

Con el fin de evitar otro gran desastre dado por la pérdida de precisión en los números de tipo *float* (ver desastre **acá**), `DCCapital Exchange` ha decidido incluir en su política el uso del estándar internacional **IEEE754** en todas sus transacciones. De esta forma, el cálculo de dinero debe seguir este estándar utilizando la librería `decimal` ofrecida por Python.

8. Seguridad y trazabilidad [23 %]

Para la tranquilidad de los usuarios, es necesario que el sistema asegure que los datos dentro del programa son correctos y válidos. De esta forma, se debe garantizar que no se perderá información de sus cuentas. Asimismo, las siguientes restricciones tienen que ser verificadas dentro del programa:

- No es posible abonar dinero negativo
- No es posible tener comisiones negativas en el mercado
- No es posible poner fechas de nacimiento menores al año 1900 y mayores a 2017
- No es posible tener un balance negativo
- No es posible ofrecer una cantidad menor o igual a 0 en un *ask* o en un *bid*
- No es posible ofrecer un precio (por unidad) menor o igual a 0 en un *ask* o en un *bid*

Además, los creadores del sistema quieren tener una lista de las acciones que han realizado los usuarios durante sus sesiones en la aplicación. La idea es poder analizar el comportamiento de los usuarios y adaptarse a sus necesidades. Para esto es necesario que se guarde la hora y los detalles de las acciones (los valores de los conceptos en itálicas) en los siguientes casos:

- *Usuario* consulta saldo
- *Usuario* consulta sus orders
- *Usuario* consulta los *orders vigentes* o *históricas* de un *mercado*
- *Usuario* consulta realiza un *order* en un *mercado*
- Sistema realiza un match entre dos *orders*

9. Bases de datos [3 %]

`DCCapital Exchange` cuenta con bases de datos previamente pobladas con la información actual de usuarios, *orders* y *currencies*. Para el trabajo pedido, `DCCapital Exchange` le facilitará estos datos a los alumnos de Programación Avanzada, los cuales se entregan en formato `CSV` (del inglés, *comma-separated values*). **Para leer y manipular estos archivos su programa debe leer las bases de datos sin usar el módulo** `csv`. Cabe destacar que en el formato `csv` el orden de las columnas puede variar. Este orden es dado por la primera fila del archivo, el cual indica el nombre de cada columna y el tipo de dato que corresponde. Junto

al enunciado se entregará un archivo csvgenerator.py que genera las bases de datos en columnas distintas y puedan comprobar que lectura de archivos CSV acepte distinto orden de las columnas.

IMPORTANTE: Cualquier modificación que se haga en alguno de estos archivos debe verse reflejado en todo momento.

Las bases de datos se describen en las siguientes subsecciones.

9.1. Usuarios

Esta base de datos se encuentra en el archivo `users.csv` y cada fila representa un usuario del sistema. Cada columna del archivo representa lo indicado en la siguiente tabla.

Cuadro 1: `users.csv`

Nombre	Tipo de dato	Comentarios
`username`	`string`	Nombre de usuario unico
`name`	`string`	Nombre del usuario.
`lastname`	`string`	Apellido del usuario.
`birthdate`	`string`	Año de nacimiento.
`orders`	`list`	Lista con id's de los orders del usuario

9.2. *Orders*

Esta base de datos se encuentra en el archivo `orders.csv` y cada fila representa un *order* junto a la información de éste. Cada columna del archivo representa lo indicado en la siguiente tabla.

Cuadro 2: `orders.csv`

Nombre	Tipo de dato	Comentarios
`order_id`	`int`	Identificador único de cada order.
`date_created`	`string`	Fecha en la cual fue creado el order.
`date_match`	`string`	Fecha en la cual ocurrió el match.
`amount`	`float`	Cantidad de monedas a transar.
`price`	`float`	Precio por cada moneda a transar.
`type`	`string`	Representa si es *ask* o *bid*.
`ticker`	`string`	Ticker del mercado en donde se encuentra la orden.

9.3. *Currencies*

Esta base de datos se encuentra en el archivo `currencies.csv` y cada fila representa una *currency* junto a la información de ésta. Cada columna del archivo representa lo indicado en la siguiente tabla.

Cuadro 3: `currencies.csv`

Nombre	Tipo de dato	Comentarios
`name`	`string`	Nombre de la criptomoneda.
`symbol`	`string`	Símbolo de la criptomoneda.

10. Últimas consideraciones

- El menú debe ser distinto según el tipo de usuario.
- Si hay un empate cuando ocurre un *match*, se considera el primero que llegó (salvo que haya un `Investor`). En otras palabras, cuando dos o más usuarios realizan una oferta al mismo precio y llega un usuario que desea comprar, se le da prioridad al usuario que primero realizó la orden de **venta**.
- Cuando ocurre un *match*, el sistema debe informarle al usuario y hacer inmediatamente el traspaso de *currencies* entre los dos usuarios.
- Si se necesitan guardar más datos de los que hay en los archivos CSV creados, su programa tiene que poder crearlos y luego usarlos en futuras ejecuciones del programa.
- No se puede editar el script entregado para generar los CSV, para evaluar se usara el original.

11. Diagrama de clases [8 %]

Junto con el programa pedido, se debe entregar, mediante un cuestionario en el Siding, un diagrama de clases con todo el modelamiento del problema. Esto incluye las clases junto con sus métodos y atributos, y todas las relaciones existentes entre estas (asociación, composición y herencia).

12. Restricciones y alcances

- Tu programa debe ser desarrollado en Python v3.6.
- Esta tarea es estrictamente individual, y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Tu código debe seguir la guía de estilos descrita en el `PEP8`.
- Si no se encuentra especificado en el enunciado, asume que el uso de cualquier librería Python está prohibida. Pregunta en el foro si es que es posible utilizar alguna librería en particular.
- Si no realizas la entrega preliminar del diagrama de clases, tendrás 5 décimas menos en la tarea.
- El ayudante puede castigar el puntaje[3] de tu tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debes adjuntar un archivo `README.md` donde comentes sus alcances y el funcionamiento del sistema (*i.e.* manual de usuario) de forma *concisa* y *clara*. **Tendrás hasta 24 horas después de la fecha de entrega** de la tarea para subir el `README.md` a tu repositorio.
- Crea un módulo para cada conjunto de clases. Divídelas por las relaciones y los tipos que poseen en común. **Se descontará hasta un punto si se entrega la tarea en un solo módulo[4].**
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

[3] Hasta −5 décimas.

[4] No agarres tu código de un solo módulo para dividirlo en dos; separa su código de forma lógica

13. Entrega

- `Diagrama de Clases`
  - **Preliminar**: 17 de agosto del 2017, 23:59 horas.
  - **Final**: 27 de agosto del 2017, 23:59 horas.
  - **Lugar:** Cuestionario en el Siding
- `Tarea`
  - **Fecha/hora:** 27 de agosto del 2017, 23:59 horas.
  - **Lugar:** GitHub – Carpeta: Tareas/T01
- `README.md`
  - **Fecha/hora:** 28 de agosto del 2017, 23:59 horas.
  - **Lugar:** GitHub – Carpeta: Tareas/T01

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).