

Tarea 1

(EN GRUPO DE 2-3 PERSONAS)

El trabajo consiste en modificar el código del expendedor de bebidas que ha hecho en PA3P. En particular, además de bebidas, se desea vender dulces. El proyecto sigue con una entrada para monedas, un selector para elegir el tipo de producto y un retorno de producto. Cada vez que se le ingresa una moneda y un número, retorne siempre un producto del tipo solicitado, si queda alguno en el depósito interno correspondiente. Debe haber un comprador que compre un producto, recupere todo el vuelto, lo consume, y responda su tipo y cuanta plata le devolvió expendedor

- Si el producto es más barato que el valor de la Moneda debe devolver la diferencia en monedas de \$100 como vuelto.
- Si no hay productos, devuelve la misma Moneda como vuelto. Debe utilizar una excepción personalizada (NoHayProductoException)
- Si se quiere comprar un producto por un valor inferior al precio, solo devuelve la misma Moneda como vuelto. Debe utilizar una excepción personalizada (PagoInsuficienteException)
- Si se quiere comprar Bebidas sin dinero (moneda null) no retorna vuelto, ni producto. Debe utilizar una excepción personalizada (PagoIncorrectoException)

Fecha de entrega en Canvas: 02 de mayo hasta las 23:59.

Entrega de un enlace de repositorio (nuevo repositorio publico) en GitHub:

- El readme debe contener los nombres completos de los/las estudiantes
- Diagrama UML (foto o imagen) de la aplicación en el origen del proyecto con clases, métodos, propiedades y relaciones (no es necesario especificar las cardinalidades, poner la clase usado de Java tipo ArrayList, el Main o los getter/setter). Se puede usar software (como <https://online.visual-paradigm.com/diagrams/features/uml-tool/>) o a mano
- Código del programa y archivos del proyecto

Este documento contiene un recordatorio de cómo funciona el software (actualizado), una lista de cambios a realizar, un tutorial de git y la rúbrica de evaluación.

Resumen del funcionamiento del software

Imagina que main es “alguien” que le da a Comprador una sola moneda de algún valor y lo manda a comprar un producto de algún tipo y que luego le consume. Después ese “alguien” le pregunta a Comprador lo que consumió y cuanta plata le dieron de vuelto. Por supuesto puede ir a comprar productos que el expendedor puede tener o no, puede tratar de comprar con monedas que no alcanzan, o bien con monedas que sobrepasan el precio. Después de la compra:

- si la moneda no alcanza o no hay producto, no le dan producto y el vuelto es la misma moneda
- si trata de comprar sin moneda (null), no le devuelven nada
- o le alcanza y le dan vuelto, pero en este caso el expendedor devuelve sólo en monedas de 100.

Debes ponerte en el lugar del comprador y del expendedor para resolver el problema.

El Comprador al momento de su creación, en el constructor, debe:

- Recibir la moneda con la que comprará, un número que identifique el tipo de producto y la referencia al Expendedor en el que comprará (no necesita que sea almacenada permanente como propiedad)
- Comprar en el expendedor entregando una moneda y el entero con el número de depósito (imaginar que lo puede ver)
- Si consiguió un producto deberá consumirlo y registrar su sabor en una propiedad
- Recuperar las monedas del vuelto una a una, hasta que se acaben y sumar la cantidad en la propiedad que almacena la cantidad total
- NO almacenar las monedas como propiedad ni los productos como propiedad
- Las propiedades son un String y un entero, nada más. Los valores se devuelven cuando se lo pidan posteriormente

El Expendedor debe:

- Al momento de su creación, en su constructor recibir la cantidad única con la que debe llenar “mágicamente” todos depósitos de productos con la misma cantidad
- Manejar monedas de \$1000, \$500 y \$100 (para recibir como pago)
- Devolver un producto si se le entrega una moneda y el número del depósito en el que está el tipo de bebida deseada, si la moneda es de un valor mayor o igual al precio
- Si se le trata de comprar con una moneda null, se lanza una excepción PagoIncorrectoException
- Si el número de depósito es erróneo, no hay producto o no alcanza. Se lanza una excepción NoHayProductoException y no entrega bebida y deja la misma moneda que recibió en el depósito de monedas de vuelto.
- Si la compra es exitosa, devolver el vuelto (sólo monedas de \$100), en el depósito de monedas de vuelto. El vuelto se crea “mágicamente” dejándolo en el depósito de vuelto: una a más monedas de 100
- Las monedas que recibe como pago no se almacenan en ningún deposito (se extinguen mágicamente)
- Las Monedas son polimórficas: Moneda1000, Moneda500, Moneda100, ..., múltiplos de 100.
- La(s) Moneda(s) de vuelto se rescatan una a una por cada llamada al método al método correspondiente
- Las monedas se diferencian por la dirección de memoria (el hashCode que tiene this y equivale a su número de serie por lo tanto no hay que manejarlo) en la que se encuentran y su valor. Deben entregar su valor y número de serie, al ser requerido (usando **toString**)
- Las bebidas deben ser de al menos tres tipos: CocaCola, Sprite, Fanta
- Los dulces deben ser de al menos dos tipos: Snickers, Super8
- Se debe entregar **con un método main de pruebas** (en Main.java por ejemplo) con las pruebas que demuestren el funcionamiento: crear un Expendedor, Monedas, y Comprador e interrogar al comprador con cada producto, como pruebas las excepciones deben ser levantada y capturada también (en este caso se muestra un mensaje). También tienen que ordenar una lista de monedas utilizando su *comparable*.

- Se debe entregar con **un segundo método main interactivo** (entre otra clase tipo MainInteractivo.java por ejemplo) donde se preguntará al usuario qué bebida desea comprar y con qué moneda en un bucle hasta que el usuario decida salir del programa. Para lograrlo, necesitará utilizar la clase [Scanner](#) de java. Hay un ejemplo de su uso en la semana 3 en las diapositivas y el código proporcionado como ejemplo.

Sobre la recuperación del vuelto:

- el comprador debe llamar varias veces a getVuelto obteniendo una moneda cada vez, hasta que le retorne null. Imagina que estás sacando las monedas desde el depósito donde caen, en uno real.

Diferencias notables con PA3P

Se espera que reutilices lo que hiciste antes en PA3P, pero con algunas diferencias que son el trabajo para esta tarea:

Estructura del proyecto:

- Cada clase debe tener su propio archivo (Comprador en Comprador.java, Bebida en Bebida.java...). ¡no cree un archivo principal con todo el código ¡Le recomiendo que empiece por esto!
- Debe utilizar GIT tal y como se ha visto durante el curso
- Haz commits poco a poco para las modificaciones, y distribuye el trabajo. Por ejemplo, una persona puede trabajar en los productos, mientras otra se encarga de hacer excepciones. Una persona pueden hacer el main de pruebas y la otra el main interactivo...

Producto, Dulce, Bebida:

- Necesitas cambiar tu jerarquía de clases usando herencia. Para no tener más sólo bebidas, pero también una clase **Producto** que puede utilizarse polimórficamente para **Dulce** y Bebida (y las subclases de eso).
- También tendrá que actualizar sus otras clases para poder utilizar los productos
- Todos los productos deben poder tener un precio diferente mediante una enumeración (ver más abajo)

Excepciones:

- Deben crear y usar **PagoIncorrectoException**, **NoHayProductoException** y **PagoInsuficienteException**.
- Los throws se hacen desde el método comprarProducto().
- Los try y catch se hacen en el Main (entonces el Comprador puede hacer Throws también).
- No tienen que aparecer en el UML.

Depósitos genéricos:

- A diferencia de en PA3P, los depósitos deben usar un tipo genérico <T> para determinar si van a almacenar un Producto o Moneda.

Monedas:

- Sin cambios en los tipos de monedas, pero deben implementar la interfaz [Comparable](#) y entonces el método compareTo (ver el curso sobre interfaz).
- Para ordenar en el main, deben [usar el sort disponible en ArrayList](#).

Documentación:

- Tienen que documentar el código usando el formato Javadoc que vimos durante el curso.
- No olvide que su editor de código (IntelliJ) puede ayudarle a generar la documentación

Enumeraciones:

- Debe sustituir las constantes que utiliza para seleccionar e indicar el precio del producto por una enumeración.
- Esta enumeración tendrá diferentes valores constantes, pero también variables y métodos para recuperar las propiedades asociadas a estos valores.
- Con eso debería poder obtener fácilmente un precio que puede ser diferente para cada producto.
- Aparte de fuera de la interfaz de la consola en su main interactivo que puede mostrar valores numéricos

al usuario (tipo 1. Coca Cola, 2. Fanta...), debe utilizar la enumeración en todas partes en lugar de la constante entera dentro de su código.

Mains:

- Su primero main debe demostrar el correcto funcionamiento de **todas las funciones de su código**. Crear y utilizar todas sus clases con los diferentes tipos de producto posibles, y también probar todas sus diferentes excepciones. Esta main también debería mostrar la posibilidad de ordenar las monedas utilizando la interfaz comparable (ver [sort](#) en la documentación y el ejemplo de uso en el curso sobre la interfaz en la semana 4).
- Su segundo main debe ser interactivo y permitir la compra de varios tipos de producto con varios tipos de moneda seguidos hasta que el usuario utilice su opción de salida. Inspírese en lo que hemos visto en las clases de la semana 3. Proponer un menú en la interfaz de la consola.
- Las dos main deben estar en un archivo diferente, no hay necesidad de incluirlos en su diagrama uml.

Resultados de aprendizaje evaluados:

1. Aplicar el paradigma de la orientación a objetos al desarrollo de aplicaciones software de mediana envergadura, en un entorno de trabajo en equipos.
3. Aplicar principios de diseño para el desarrollo de sistemas informáticos de calidad.

Rúbrica de evaluación Tarea 1



Crterios	Calificaciones			Pts
Ejecución del código KPI: 1.3, 2.1	12 para >8 pts Excelente El código se ejecuta sin problemas y el main de pruebas es suficiente para dar una idea de cómo funciona el software. El main interactivo es totalmente funcional.	8 para >4 pts Bien El código se ejecuta sin problemas pero falta algunas partes en los mains o hay errores menores.	4 para >0 pts Por mejorar El código tiene problemas durante la ejecución o la implementación de los mains está incompleta.	12 pts
Compleitud de la aplicación KPI: 1.1	9 para >6 pts Excelente La aplicación contiene todo el código necesario y funciona correctamente siguiendo las instrucciones dadas.	6 para >3 pts Bien Faltan algunas partes menores del código o hay problemas menores de implementación.	3 para >0 pts Por mejorar Faltan algunas partes importantes del código o hay problemas importantes con la implementación.	9 pts
Calidad del código KPI: 1.3, 2.1	9 para >6 pts Excelente El código proporcionado utiliza correctamente los conceptos de programación OO y JAVA vistos en clase, tales como: encapsulación, polimorfismo, herencia, abstracción, uso de interfaces, excepciones y enumeraciones.	6 para >3 pts Bien La mayor parte del código hace un buen uso de los conceptos OO y de las capacidades del lenguaje JAVA, pero algunas partes menores del código deben mejorarse en estos aspectos.	3 para >0 pts Por mejorar Gran parte del código proporcionado no utiliza correctamente los conceptos OO, o muestra una falta de dominio de las posibilidades que ofrece el lenguaje JAVA.	9 pts
Documentación KPI: 1.2	6 para >4 pts Excelente Existe documentación en formato javadoc para todas las clases y métodos públicos.	4 para >2 pts Bien alta informaciones menores en el modelo UML y/o hay errores menores.	2 para >0 pts Por mejorar Documentación incompleta o formato incorrecto.	6 pts
Trabajo en grupo KPI: 5.1	6 para >4 pts Excelente El trabajo está bien distribuido entre los miembros del equipo y esto se refleja tanto en Git.	4 para >2 pts Bien El trabajo está bastante bien distribuido en general pero hay un ligero desequilibrio entre los miembros del grupo.	2 para >0 pts Por mejorar El trabajo está desequilibrado o existe una falta de cooperación por parte de uno o más miembros del grupo. Si no hay una contribución significativa de una o varias personas, se puede aplicar una penalización de puntos en el total de la nota o poner NCR.	6 pts
Uso de GIT y GitHub KPI: 5.2	6 para >4 pts Excelente GIT se ha utilizado correctamente (los tamaños, las frecuencia, ramas y los mensajes de commits son relevantes).	4 para >2 pts Bien El uso de git es correcto en general, pero algunos mensajes de commits no son claros o las commits deberían ser un poco más frecuentes.	2 para >0 pts Por mejorar Se hace un uso moderado de GIT, pero hay problemas recurrentes con los commits en mensajes o frecuencia. Si no se usa git lo suficiente, se puede aplicar una penalización de puntos en el total de la nota o poner NCR (especialmente si se utiliza la interfaz de github en lugar de git directamente)	6 pts
Modelo de clases UML KPI: 1.1	6 para >4 pts Excelente El modelo UML es correcto (contiene todas las clases, métodos, propiedades o relaciones para resolver el problema) y se respeta el formato UML.	4 para >2 pts Bien Falta informaciones menores en el modelo UML y/o hay errores menores.	2 para >0 pts Por mejorar El modelo UML es bastante incompleto o no respeta el estándar UML.	6 pts
Correspondencia entre diagrama UML y código KPI: 1.2	6 para >4 pts Excelente El código proporcionado y el diagrama UML proporcionado corresponden.	4 para >2 pts Bien Hay pequeñas incoherencias entre el modelo UML proporcionado y el código.	2 para >0 pts Por mejorar Hay muchas incoherencias entre el modelo UML y el código proporcionado.	6 pts
Puntos totales: 60				

Breve tutorial sobre Git y GitHub (ver curso o enlaces del curso para más detalles)

Git está instalado en las máquinas Linux de las salas de laboratorio,

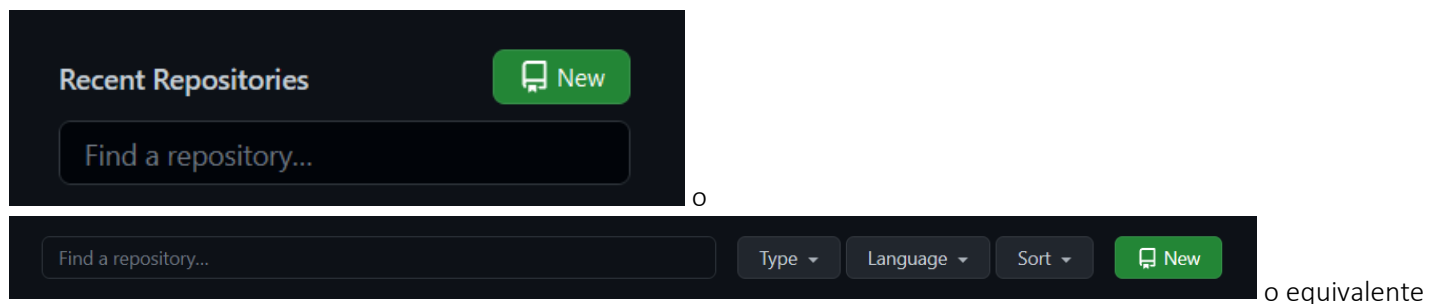
Si quieren instalarlo en su máquina pueden hacerlo desde aquí: <https://git-scm.com/downloads>

(debería ser posible utilizar la versión portátil en las ventanas del laboratorio también - pero no hay garantía)

También deben crear una cuenta en github : <https://github.com/>

Usando **Sign Up**. Respondan a la pregunta y asegúrense de seleccionar la oferta gratuita (normalmente deberían tener una cuenta de estudiante con el correo electrónico de la universidad, pero no es necesario)

Una vez conectado se debe crear un nuevo repositorio (uno por grupo)



Haga clic en new (nuevo)

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * GeoffreyHecht / Repository name * tarea1 ✓

Great repository names are short and memorable. Need inspiration? How about [automatic-eureka?](#)

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)
.gitignore template: Java



Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)
License: None


This will set main as the default branch. Change the default name in your [settings](#).

ⓘ You are creating a public repository in your personal account.

[Create repository](#)

Poner un nombre, un readme, un **gitignore Java** y hacer clic en create repository

Para poder trabajar juntos más fácilmente, hacer click en  **Settings** y  **Collaborators** y [Add people](#) y

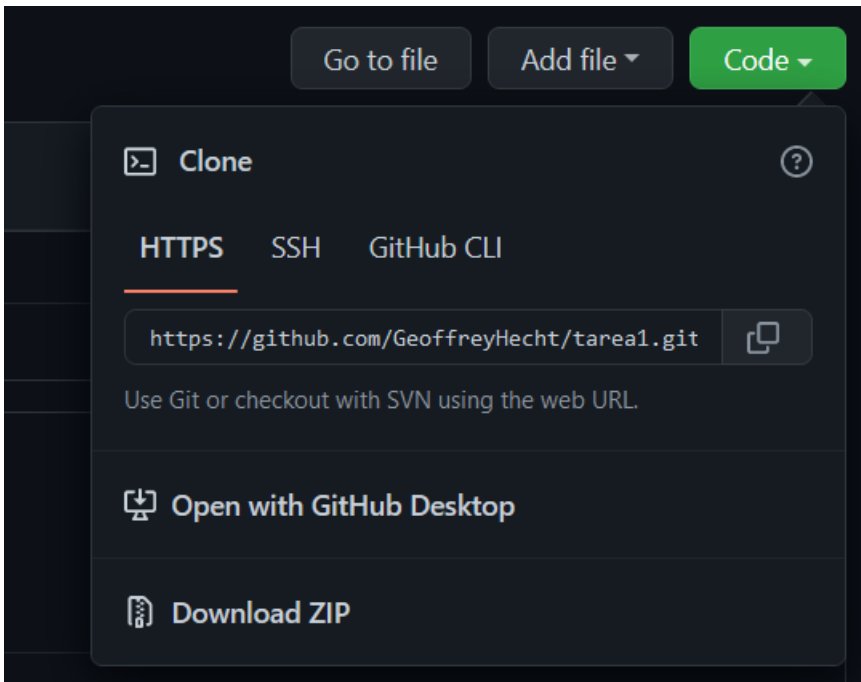


Add a collaborator to tarea1

[Select a collaborator above](#)

Busque a su colega con su nombre de usuario o su nombre

Luego ambas personas pueden clonar el depósito (desde la pagina principal del proyecto)



Hacer clic en code y copiar la dirección dada que termina en .git

Ahora en el shell bash (o git bash bajo windows) y desde la carpeta donde quieren trabajar

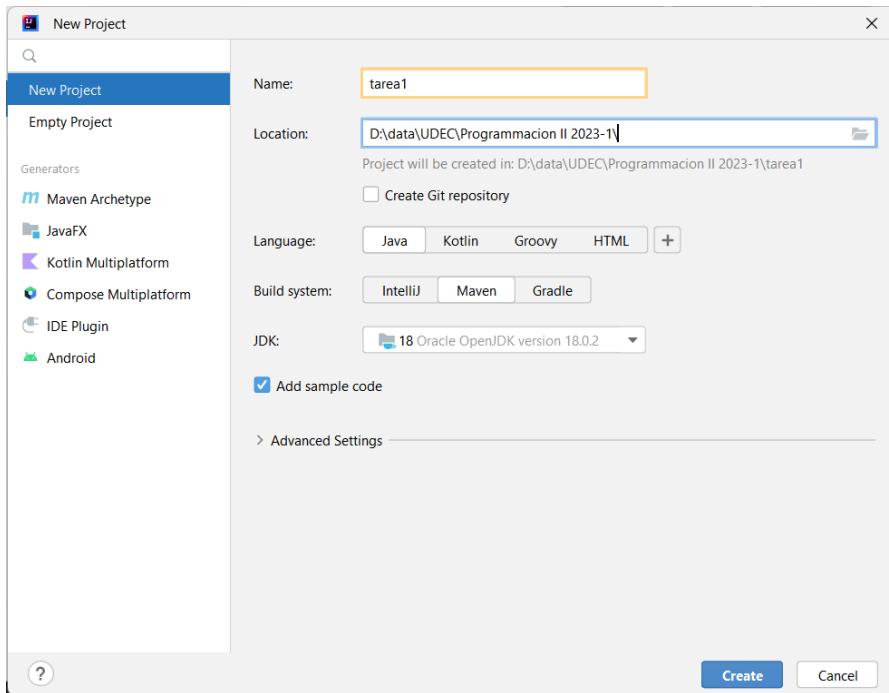
Usar

`git clone {{ dirección.git }}`

```
MINGW64:/d/data/UDEC/Programmacion II 2023-1
coldf@LAPTOP-0UF8SOG9 MINGW64 /d/data/UDEC/Programmacion II 2023-1
$ git clone https://github.com/GeoffreyHecht/tarea1.git
Cloning into 'tarea1'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 18 (delta 1), reused 15 (delta 1), pack-reused 0
Receiving objects: 100% (18/18), 16.51 KiB | 2.06 MiB/s, done.
Resolving deltas: 100% (1/1), done.

coldf@LAPTOP-0UF8SOG9 MINGW64 /d/data/UDEC/Programmacion II 2023-1
$ |
```


Ahora una de las dos personas puede crear un proyecto IntelliJ apuntando a la misma carpeta (aquí tarea1)



Desde el shell ahora pueden ver y añadir sus primeros archivos con git status, git add y git commit

```
$ git add .

coldf@LAPTOP-0UF8SOG9 MINGW64 ~/OneDrive/Documents/NetBeansProjects/tarea1 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   .gitignore
    new file:   TareaProg/build.xml
    new file:   TareaProg/manifest.mf
    new file:   TareaProg/nbproject/build-impl.xml
    new file:   TareaProg/nbproject/genfiles.properties
    new file:   TareaProg/nbproject/project.properties
    new file:   TareaProg/nbproject/project.xml
    new file:   TareaProg/src/tareaprog/TareaProg.java

coldf@LAPTOP-0UF8SOG9 MINGW64 ~/OneDrive/Documents/NetBeansProjects/tarea1 (main)
$ git commit -m "Estructura del proyecto y primera clase main"
[main 49afac6] Estructura del proyecto y primera clase main
 8 files changed, 1987 insertions(+)
 create mode 100644 TareaProg/build.xml
 create mode 100644 TareaProg/manifest.mf
 create mode 100644 TareaProg/nbproject/build-impl.xml
 create mode 100644 TareaProg/nbproject/genfiles.properties
 create mode 100644 TareaProg/nbproject/project.properties
 create mode 100644 TareaProg/nbproject/project.xml
 create mode 100644 TareaProg/src/tareaprog/TareaProg.java
```

Luego puedes subirlos a github con git push

```
$ git push
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 12 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 15.73 KiB | 5.24 MiB/s, done.
Total 14 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/GeoffreyHecht/tarea1.git
58146c8..49afacb main -> main
```

Dependiendo de su instalación y de su sistema operativo, tendrá que iniciar la sesión a través del sitio web (Windows, Mac o algunas versiones de Linux) o con las credenciales en el Shell.

En este último caso, consulte esta documentación (o si tienen problemas para push a github) :

<https://docs.github.com/es/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>

Su colega puede entonces clonar el proyecto y trabajar desde su máquina (o hacer un pull si ya tiene un clon).

Ahora puedes trabajar con los comandos vistos en el curso (add, branch, checkout, merge, status...). **No olviden hacer commits/pushes/pulls regulares con mensajes claros y útiles.**

Deben poner. idea/* en el .gitignore (Así se evita compartir archivos innecesarios).

No dude en pedir ayuda a los ayudantes o a mí.

No olviden dar la dirección de su repositorio de GitHub (sin el .git, sólo la página) en Canvas

No olviden poner su nombres completos en el readme de GitHub