**INFORME DE LABORATORIO**

**(Formato Estudiante)**

| **INFORMACIÓN BÁSICA** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **ASIGNATURA:** | Pruebas de Software | | | | |
| **TÍTULO DE LA PRÁCTICA:** | *Pruebas Unitarias y Refactorización de un Cajero Automático (ATM)* | | | | |
| **NÚMERO DE PRÁCTICA:** | *02* | **AÑO LECTIVO:** | *2023* | **NRO. SEMESTRE:** | *VII* |
| **FECHA DE PRESENTACIÓN** | *19/05/2023* | **HORA DE PRESENTACIÓN** | *07:20* | | |
| **INTEGRANTE (s):**  Hincho Jove, Angel Eduardo  Neira Carrasco, Darwin Jesus | | | | **NOTA:** |  |
| **DOCENTE(s):**  *Arisaca Mamani, Robert Edison* | | | | | |

| **SOLUCIÓN Y RESULTADOS** |
| --- |
| 1. **SOLUCIÓN DE EJERCICIOS/PROBLEMAS**   ***PROBLEMAS PROPUESTOS***  *La solución o programa donde se refactoriza el Cajero Automático (ATM) se encuentra disponible en la plataforma de GitHub bajo el siguiente enlace:* [*https://github.com/ahincho/PrS-TeoA-ATM.git*](https://github.com/ahincho/PrS-TeoA-ATM.git)   1. ***Casos de Prueba:***     *Siguiendo una metodología Test Driven Development, se especificaron los casos de prueba y contratos antes de realizar la implementación. Para este problema se identificaron dos entidades principales: Usuario y ATM. Estamos viendo los casos de prueba que debe cumplir la implementación de un Usuario.*    *Los dos primeros casos de prueba para la Clase Usuario nos especifican que tanto la contraseña como el nombre del Usuario deben ser valores de tipo String. Así mismo, deben cumplir con alguna condición, en este caso nuestra condición será que tenga más de 4 caracteres.*    *Los tres siguientes casos de pruebas o pruebas unitarias se refieren a la parte económica. Tanto el salario, total de depósitos y retiros realizados en el día deben ser valores enteros o flotantes mayores a 0.*    *Las pruebas unitarias restantes nos ayudarán a comprobar el correcto funcionamiento de la asignación de atributos y propiedades de la clase Usuario. Revisaremos una correcta inicialización del nombre, contraseña, cantidad de dinero, retiro y abono.*    *Ahora crearemos las pruebas unitarias para el Cajero Automático o ATM. En este caso necesitaremos verificar que recibe un Usuario ha atender, los movimientos bancarios deben ser cantidades positivas y no debe superarse el límite impuesto de 3000 unidades durante un solo día tanto en abonos como retiros.*    *En los primeros 5 casos de prueba verificaremos algunas excepciones y errores de asignación que podrían darse al crear el Usuario del ATM así como al realizar un movimiento. Los movimientos bancarios o transaccionales deben ser enteros o flotantes positivos. Pruebas unitarias para el retiro.*    *Siguiendo con las pruebas unitarias para los retiros. No deben exceder el límite corporativo impuesto de 3000 unidades monetarias. Tampoco deben exceder el salario disponible por el Usuario y tampoco deben tomar valores negativos.*    *De igual manera se crean pruebas unitarias para la operación de depósito. Donde se verifica el tipo de dato ingresado así como no superar el límite establecido en 3000 unidades. Tampoco se aceptarán negativos.*   1. ***Implementación:***     *En la implementación de la clase Usuario, delegamos la asignación de los parámetros o atributos a otros métodos setters que contemplarán algunos errores y excepciones que podrían presentarse.*    *Los métodos setters para el nombre y contraseña verificarán que el tipo de dato ingresado sea un string o conjunto de caracteres sino devolverá un error de tipado. Así mismo si no tienen una longitud mayor a 4 caracteres entonces devolverá un error de valor. El salario debe ser un entero o flotante positivo.*    *También verificamos los valores que ingresan como valores iniciales para el monto de retiro y depósito diario. Tienen que tratarse de valores enteros o flotantes positivos.*    *Se hace uso de una clase Enumerador ATM\_Options para especificar de mejor manera las opciones disponibles dentro del Cajero Automático y que sea más entendible en código.*    *También se crea una clase Enumeradora ATM\_Config para guardar algunos valores útiles en la configuración inicial del Cajero y tener un mantenimiento a futuro más cómodo y flexible.*    *Para la clase NewATM utilizaremos un método auxiliar que imprimirá un salto de línea formado por asteriscos. El Cajero trabajará con un Usuario y se tendrá que asignar previa evaluación de tipo y valor.*    *Creamos el método withdraw() o retiro() que recibirá como argumento el valor o monto a retirar y verificará su tipo de dato y si se trata de un valor positivo. Luego verificará si no se excede el monto diario a retirar así como que se cuente con saldo suficiente para la operación. Finalmente hace el retiro.*    *De igual manera, para los abonos o depósitos verificaremos el tipo de dato así como su valor positivo. Luego revisaremos si se encuentra dentro del límite corporativo. Finalmente se hace el abono a la cuenta.*    *Finalmente tenemos el método para mostrar el estado de la cuenta. Devolverá el salario total que posee la cuenta así como la cantidad depositada y retirada el día de hoy.*    *Adicionalmente vamos a imprimir el menú con las opciones disponibles dentro del Cajero Automático.*    *Finalmente generamos un método para mostrar el menú general en el cual el usuario podrá seleccionar una opción entre las implementadas para revisar su cuenta, depositar o retirar.*    *Se crea un archivo JSON para guardar datos relevantes sobre nuestros usuarios y poder dar una configuración inicial a nuestro Cajero Automático ATM.*   1. ***Ejecución:***     *Ejecutamos los casos de prueba y podemos ver como todos los casos de pruebas han sido superados con éxito. Esto nos da una primera barrera de seguridad y confianza en nuestro sistema.*    *Ingresando al Cajero Automático bajo el Usuario Angel. Revisamos el estado de la cuenta.*    *Tratamos de depositar y abonar más de 3000 unidades monetarias que es el límite corporativo establecido. Podemos ver como el sistema actúa de manera adecuada indicando que no es posible.*    *Realizamos un depósito válido y revisamos el estado de la cuenta.*    *Realizamos un retiro válido y revisamos el estado de la cuenta.*    *Ingresando el Usuario Darwin con credenciales incorrectas hasta el último intento donde nos logueamos correctamente con las credenciales correspondientes.*    *En caso se ingrese un valor no entero en el menú entonces vamos a mostrar un mensaje que indique que solo aceptamos valores enteros mostrados en el menú.*    *Ahora intentamos abonar un valor no permitido, por ejemplo, escribimos un valor de tipo string. Vemos como el sistema se da cuenta de ello y muestra un mensaje concordante.*    *Intentando ingresar un monto a retirar negativo. Esto es imposible y se muestra el mensaje adecuado.*    *Intentamos ingresar una cadena de caracteres en el menú del Cajero Automático y vemos cómo responde correctamente manejando la excepción o error.*    *Finalmente, cuando fallemos 3 veces al intentar ingresar un Usuario y contraseña el sistema se cerrará y se notificará que se ha enviado a un personal del banco para dar el soporte necesario.* |
| 1. **SOLUCIÓN DEL CUESTIONARIO**   *No se compartió o publicó un Cuestionario para el presente Laboratorio.* |
| 1. **CONCLUSIONES**   ***Conclusión 1:*** *Durante el desarrollo del Laboratorio se utilizó TDD y se consideró varios casos de prueba de acuerdo al código base brindado para el Cajero Automático, estos casos sirvieron para hacer un mejor uso de la herramienta de unittest así como ver su utilidad al momento del desarrollo y especificación.*  ***Conclusión 2:*** *Utilizar unittest para automatizar pruebas unitarias es una excelente práctica de desarrollo. Permite verificar que cada componente de código funcione correctamente de forma aislada y que siga funcionando correctamente a medida que realizamos cambios en el código. Esto brinda confianza en la calidad y la estabilidad de tu código.*  ***Conclusión 3:*** *El manejo adecuado de errores y excepciones es crucial para garantizar la robustez y la confiabilidad del código. Es importante asegurarse que las pruebas unitarias cubran la mayor cantidad de casos posibles, incluyendo escenarios de éxito y situaciones de manejo de errores.* |

| **RETROALIMENTACIÓN GENERAL** |
| --- |
|  |

| **REFERENCIAS Y BIBLIOGRAFÍA** |
| --- |
| *[1] "Unittest Unit testing framework". Python.* [*https://docs.python.org/3/library/unittest.html*](https://docs.python.org/3/library/unittest.html) *.* |