DOCUMENTACIÓN DIAGRAMA DE CLASES ARG-BROKER

Este diagrama de clases UML representa la estructura de un sistema de gestión de inversiones, modelando las entidades clave y sus relaciones. El sistema permite a los inversores gestionar sus portafolios, realizar transacciones, y monitorear el estado de sus inversiones.

Convenciones de nomenclatura UML

- Clases: PascalCase, ej. Usuario, Portafolio.
- Atributos: snake_case, comenzando con un verbo en infinitivo, ej. agregar_accion.
- Métodos: snake_case, comenzando con un verbo en infinitivo, ej. agregar_accion.

Descripción de las Clases

Entidades del Dominio

CLASE INVERSOR: Representa a un usuario del sistema con información personal y financiera.

Atributos:

id_inversor: Identificador único del inversor (int).

nombre: Nombre del inversor (str).

apellido: Apellido del inversor (str).

email: Correo electrónico del inversor (str).

dni: Documento Nacional de Identidad del inversor (str).

usuario: Nombre de usuario del inversor (str).

contrasena: Contraseña del inversor (str).

estado_cuenta: Estado de la cuenta (int).

estado_activo: Estado activo del inversor (bool).

is_staff: Indica si el inversor tiene permisos de administrador (bool).

Métodos:

Esta clase tiene implementado por cada atributo su respectivo getter y setter. Su uso se debe a que se pretende mantener un acceso controlado a la información. No se los enumera uno por uno porque todos tienen una estructura muy similar, por ej:

el decorador property permite ver el atributo:

```
    @property
    def id_inversor(self):
        return self._id_inversor
```

y el setter permite acceder al atributo para modificarlo:

```
 @id_inversor.setter
    def id_inversor(self, id_inversor):
        if not isinstance(id_inversor, int):
            raise ValueError('El ID del inversor debe ser un número entero.')
        self._id_inversor = id_inversor
```

__str__: Devuelve una representación en cadena de la instancia de la clase Inversor.

CLASE Accion: Representa una acción en el mercado de valores.

Atributos:

id_accion: Identificador único de la acción (int).

nombre_accion: Nombre de la acción (str).

simbolo_accion: Símbolo de la acción (str).

Métodos:

Esta clase tiene implementado por cada atributo su respectivo getter y setter. Su uso se debe a que se pretende mantener un acceso controlado a la información. No se los enumera uno por uno porque todos tienen una estructura muy similar, por ej:

el decorador property permite ver el atributo:

```
@property
    def nombre_accion (self):
        return self._nombre_accion
```

y el setter permite acceder al atributo para modificarlo:

@nombre_accion.setter

    def nombre_accion (self, nuevo_nombre):

__str__: Devuelve una representación en cadena de la instancia de la clase accion.

CLASE Transaccion: Representa una compra o venta de acciones.

Atributos:

id_transaccion: Identificador único de la transacción (int).

id_accion: Identificador de la acción asociada (int).

id_portafolio: Identificador del portafolio asociado (int).

tipo: Tipo de transacción (str).

fecha: Fecha de la transacción (date).

precio: Precio de la transacción (int).

comision: Comisión de la transacción (float).

Métodos:

Esta clase tiene implementado por cada atributo su respectivo getter y setter. Su uso se debe a que se pretende mantener un acceso controlado a la información. No se los enumera uno por uno porque todos tienen una estructura muy similar, por ej:

el decorador property permite ver el atributo:

@property

    def id_transaccion(self):

        return self._id_transaccion

y el setter permite acceder al atributo para modificarlo:

@id_transaccion.setter

    def id_transaccion(self, id_transaccion):

        self._id_transaccion = id_transaccion

__str__: Devuelve una representación en cadena de la instancia de la clase Transaccion.

obtener_monto_total: Calcula y devuelve el monto total de la transacción (cantidad * precio).

CLASE Portafolio: Representa una colección de acciones de un inversor.

Atributos:

id_portafolio: Identificador único del portafolio (int).

id_inversor: Identificador del inversor propietario del portafolio (int).

Métodos

Esta clase tiene implementado por cada atributo su respectivo getter y setter. Su uso se debe a que se pretende mantener un acceso controlado a la información. No se los enumera uno por uno porque todos tienen una estructura muy similar, por ej:

el decorador property permite ver el atributo:

```
@property
    def id_portafolio(self):
        return self._id_portafolio
```

y el setter permite acceder al atributo para modificarlo:

```
   @id_portafolio.setter
    def id_portafolio(self, id_portafolio):
        self._id_portafolio = id_portafolio
```

__str__: Devuelve una representación en cadena de la instancia de la clase Portafolio.

CLASE HistorialSaldo: Registra los cambios en el saldo de un inversor a lo largo del tiempo.

Atributos:

id_historial_saldo: Identificador único del historial de saldo (int).

id_inversor: Identificador del inversor asociado (int).

fecha: Fecha del registro de saldo (date).

valor_saldo: Valor del saldo (float).

moneda: Tipo de moneda (str).

Métodos:

Esta clase tiene implementado por cada atributo su respectivo getter y setter. Su uso se debe a que se pretende mantener un acceso controlado a la información. No se los enumera uno por uno porque todos tienen una estructura muy similar, por ej:

el decorador property permite ver el atributo:

```
@property
    def id_historial_saldo(self):
        return self._id_historial_saldo
```

y el setter permite acceder al atributo para modificarlo:

```
@id_historial_saldo.setter
    def id_historial_saldo(self, value):
        self._id_historial_saldo = value
```

__str__: Devuelve una representación en cadena de la instancia de la clase HistorialSaldo.

CLASE EstadoPortafolio: Representa el estado actual de un portafolio en un momento dado

Atributos:

id_estado: Identificador único del estado (int).

fecha: Fecha de registro del estado (date).

cantidad_compra_accion: Cantidad de acciones compradas (int).

cantidad_venta_accion: Cantidad de acciones vendidas (int).

precio_compra_actual: Precio actual de compra (float).

precio_venta_actual: Precio actual de venta (float).

valor_apertura: Valor de apertura (float).

nombre_clase: Nombre de la clase de activos (str).

maximo_dia: Valor máximo del día (int).

valor_cierre: Valor de cierre (float).

Métodos:

Esta clase tiene implementado por cada atributo su respectivo getter y setter. Su uso se debe a que se pretende mantener un acceso controlado a la información. No se los enumera uno por uno porque todos tienen una estructura muy similar, por ej:

el decorador property permite ver el atributo:

```
@property
    def id_estado_portafolio(self):
        return self._id_estado_portafolio
```

y el setter permite acceder al atributo para modificarlo:

```
 @id_estado_portafolio.setter
    def id_estado_portafolio(self, value):
        self._id_estado_portafolio = value
```

__str__: Devuelve una representación en cadena de la instancia de la clase EstadoPortafolio.

CLASE CotizacionDiaria: Representa la cotización de una acción en un día determinado.

Atributos:

id_cotizacion: Identificador único de la cotización (int).

id_accion: Identificador de la acción asociada (int).

fecha: Fecha de la cotización (date).

valor_apertura: Valor de apertura (float).

valor_minimo_dia: Valor mínimo del día (float).

valor_maximo_dia: Valor máximo del día (float).

valor_cierre: Valor de cierre (float).

Métodos:

Esta clase tiene implementado por cada atributo su respectivo getter y setter. Su uso se debe a que se pretende mantener un acceso controlado a la información. No se los enumera uno por uno porque todos tienen una estructura muy similar, por ej:

el decorador property permite ver el atributo:

```
@property
    def id_cotizacion(self):
        return self._id_cotizacion
```

y el setter permite acceder al atributo para modificarlo:

```
@id_cotizacion.setter
    def id_cotizacion(self, value):
        self._id_cotizacion = value
```

__str__: Devuelve una representación en cadena de la instancia de la clase CotizacionDiaria.

DAOs (Data Access Objects)

CLASE AccionDAO es una implementación de la interfaz DAOInterface que maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la entidad Accion en una base de datos MySQL. Utiliza un conector a la base de datos, que se espera sea una instancia de la clase MySQLConnector.

Atributos:

__base_de_datos: Es el conector a la base de datos que se utiliza para realizar las operaciones SQL.

Métodos:

Parámetros:

conector: Instancia de MySQLConnector.

Crear: Inserta una nueva acción en la base de datos.

Parámetros:

nueva_accion: Instancia de Accion que contiene los datos de la nueva acción.

Retorna: True si la operación es exitosa.

obtener_uno: Obtiene una acción de la base de datos por su ID.

Parámetros:

id_accion: ID de la acción a obtener.

Retorna: Una instancia de Accion con los datos obtenidos.

obtener_todos: Obtiene todas las acciones de la base de datos.

Retorna: Una lista de instancias de Accion con los datos obtenidos.

actualizar: Actualiza una acción existente en la base de datos.

Parámetros:

nuevos_datos: Instancia de Accion con los nuevos datos.

id_accion_a_modificar: ID de la acción a actualizar.

Retorna: True si la operación es exitosa.

eliminar: Elimina una acción de la base de datos por su ID.

Parámetros:

id: ID de la acción a eliminar.

Retorna True si la operación es exitosa.

CLASE CotizacionDAO: es una implementación de la interfaz DAOInterface que maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la entidad CotizacionDiaria en una base de datos MySQL. Utiliza un conector a la base de datos, que se espera sea una instancia de la clase MySQLConnector.

Métodos:

crear: Inserta una nueva cotización diaria en la base de datos.

Parámetros:

cotizacion_diaria: Instancia de CotizacionDiaria que contiene los datos de la nueva cotización.

Retorna True si la operación es exitosa.

obtener_uno: Obtiene una cotización diaria de la base de datos por su ID.

Parámetros:

id_cotizacion: ID de la cotización a obtener.

Retorna: Una instancia de CotizacionDiaria con los datos obtenidos.

obtener_todos: Obtiene todas las cotizaciones diarias de la base de datos para una acción específica.

Parámetros:

id_accion: ID de la acción cuyas cotizaciones se desean obtener.

Retorna una lista de instancias de CotizacionDiaria con los datos obtenidos.

actualizar: Actualiza una cotización diaria existente en la base de datos.

Parámetros:

cotizacion_modificada: Instancia de CotizacionDiaria con los nuevos datos.

id_de_cotizacion_a_modificar: ID de la cotización a actualizar.

Retorna True si la operación es exitosa.

eliminar: Elimina una cotización diaria de la base de datos por su ID.

Parámetros:

id_cotizacion: ID de la cotización a eliminar.

Retorna True si la operación es exitosa.

CLASE Dao_interface: Esta clase abstracta define la interfaz para las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) que deben implementar las clases DAO. Esta interfaz asegura que cualquier clase que la implemente tendrá los métodos necesarios para interactuar con la base de datos de manera consistente.

Métodos:

crear: Método abstracto para insertar un nuevo objeto en la base de datos.

Parámetros:

objeto: El objeto a insertar en la base de datos.

obtener_uno: Método abstracto para obtener un objeto de la base de datos por su ID.

Parámetros:

id: El ID del objeto a obtener.

obtener_todos: Método abstracto para obtener todos los objetos de la base de datos relacionados con un ID específico.

Parámetros:

id: El ID relacionado con los objetos a obtener.

actualizar: Método abstracto para actualizar un objeto existente en la base de datos.

Parámetros:

objeto: El objeto con los datos actualizados.

eliminar:Método abstracto para eliminar un objeto de la base de datos por su ID.

Parámetros:

id: El ID del objeto a eliminar.

CLASE EstadoPortafolioDAO es una implementación de la interfaz DAOInterface que maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la entidad EstadoPortafolio en la base de datos MySQL. Utiliza un conector a la base de datos, que se espera sea una instancia de la clase MySQLConnector.

Atributos:

__base_de_datos:

Tipo: MySQLConnector

Descripción: Es el conector a la base de datos que se utiliza para realizar las operaciones SQL.

Métodos:

crear: Inserta un nuevo estado de portafolio en la base de datos.

Parámetros:

estado_portafolio: Instancia de EstadoPortafolio que contiene los datos del nuevo estado de portafolio.

Retorna True si la operación es exitosa.

actualizar: Actualiza un estado de portafolio existente en la base de datos.

Parámetros:

estado_portafolio: Instancia de EstadoPortafolio con los datos actualizados.

id_estado_a_modificar: ID del estado de portafolio a actualizar.

Retorna True si la operación es exitosa.

eliminar: Elimina un estado de portafolio de la base de datos por su ID.

Parámetros:

id_estado_portafolio: ID del estado de portafolio a eliminar.

Retorna True si la operación es exitosa.

obtener_uno: Obtiene un estado de portafolio de la base de datos por su ID.

Parámetros:

id_estado_portafolio: ID del estado de portafolio a obtener.

Retorna: Una instancia de EstadoPortafolio con los datos obtenidos.

Actualizar: Actualiza un estado de portafolio existente en la base de datos.

Parámetros:

estado_portafolio_modificado: Instancia de EstadoPortafolio con los nuevos datos.

id_estado_portafolio_a_modificar: ID del estado de portafolio a actualizar.

Retorna: True si la operación es exitosa.

Eliminar: Elimina un estado de portafolio de la base de datos por su ID.

Parámetros:

id_estado_portafolio: ID del estado de portafolio a eliminar.

Retorna: True si la operación es exitosa, False en caso de error.

CLASE historial_saldo_dao: representa el Data Access Object (DAO) para la entidad HistorialSaldo.

Atributos:

__base_de_datos: Conector a la base de datos utilizado para realizar las operaciones CRUD.

Métodos:

crear: Inserta un nuevo registro de historial de saldo en la base de datos.

obtener_uno: Obtiene un registro de historial de saldo de la base de datos por su identificador único.

obtener_todos: Obtiene todos los registros de historial de saldo de la base de datos.

actualizar: Actualiza un registro de historial de saldo en la base de datos.

eliminar: Elimina un registro de historial de saldo de la base de datos por su identificador único.

CLASE InversorDAO maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la entidad Inversor en una base de datos MySQL. Implementa la interfaz DAOInterface.

Atributos:

__base_de_datos:

Tipo: MySQLConnector

Es el conector a la base de datos que se utiliza para realizar las operaciones SQL.

Métodos:

crear: Inserta un nuevo inversor en la base de datos.

Parámetros:

inversor: Instancia de Inversor que contiene los datos del nuevo inversor.

Retorna: True si la operación es exitosa.

obtener_uno: Obtiene un inversor de la base de datos por su ID.

Parámetros:

id_inversor: ID del inversor a obtener.

Retorna: Una instancia de Inversor con los datos obtenidos.

obtener_todos(self): Obtiene todos los inversores de la base de datos.

Retorna: Una lista de instancias de Inversor con los datos obtenidos.

Actualizar: Actualiza un inversor existente en la base de datos.

Parámetros:

inversor: Instancia de Inversor con los nuevos datos.

id: ID del inversor a actualizar.

Retorna: True si la operación es exitosa.

eliminar: Elimina un inversor de la base de datos por su ID.

Parámetros:

id_inversor: ID del inversor a eliminar.

Retorna: True si la operación es exitosa, False en caso de error.

CLASE PortafolioDAO maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la entidad Portafolio en la base de datos MySQL. Implementa la interfaz DAOInterface.

Atributos:

__base_de_datos:

Tipo: MySQLConnector

Es el conector a la base de datos que se utiliza para realizar las operaciones SQL.

Métodos:

crear: Inserta un nuevo portafolio en la base de datos.

Parámetros:

portafolio: Instancia de Portafolio que contiene los datos del nuevo portafolio.

Retorna: True si la operación es exitosa.

actualizar: Actualiza un portafolio existente en la base de datos.

Parámetros:

portafolio: Instancia de Portafolio con los nuevos datos.

Retorna: True si la operación es exitosa, False en caso de error.

eliminar: Elimina un portafolio de la base de datos por su ID.

Parámetros:

id_portafolio: ID del portafolio a eliminar.

Retorna: True si la operación es exitosa, False en caso de error.

obtener_todos: Obtiene todos los portafolios de la base de datos.

Parámetros: Ninguno.

Retorna: Una lista de instancias de Portafolio con los datos obtenidos.

obtener_uno: Obtiene un portafolio de la base de datos por el ID del inversor.

Parámetros:

id_inversor: ID del inversor cuyo portafolio se desea obtener.

Retorna: Una lista de instancias de Portafolio con los datos obtenidos.

CLASE TransaccionDAO maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la entidad Transaccion en una base de datos MySQL. Implementa la interfaz DAOInterface.

Atributos:

_base_de_datos:

Tipo: MySQLConnector

Es el conector a la base de datos que se utiliza para realizar las operaciones SQL.

Métodos:

crear: Inserta una nueva transacción en la base de datos.

Parámetros:

transaccion: Instancia de Transaccion que contiene los datos de la nueva transacción.

Retorna: True si la operación es exitosa.

obtener_todos: Obtiene todas las transacciones de la base de datos.

Parámetros: Ninguno.

Retorna: Una lista de instancias de Transaccion con los datos obtenidos.

Actualizar: Actualiza una transacción existente en la base de datos.

Parámetros:

transaccion: Instancia de Transaccion con los nuevos datos.

id_transaccion: ID de la transacción a actualizar.

Retorna: True si la operación es exitosa.

eliminar: Elimina una transacción de la base de datos por su ID.

Parámetros:

id_a_eliminar: ID de la transacción a eliminar.

Retorna: True si la operación es exitosa.

obtener_uno: Obtiene una transacción de la base de datos por su ID.

Parámetros:

id_transaccion: ID de la transacción a obtener.

Retorna: Una instancia de Transaccion con los datos obtenidos.

obtener_por_portafolio_y_accion: Obtiene todas las transacciones de la base de datos para un portafolio y acción específicos.

Parámetros:

id_portafolio: ID del portafolio.

id_accion: ID de la acción.

Retorna: Una lista de instancias de Transaccion con los datos obtenidos.

Servicios

CLASE ServicioDeAutenticacion maneja la autenticación de inversores en el sistema. Utiliza un objeto de tipo InversorDAO para interactuar con la base de datos.

Atributos:

__inversor_dao:

Tipo: InversorDAO DAO utilizado para interactuar con la base de datos de inversores.

Parámetros:

inversor_dao: Instancia de InversorDAO.

iniciar_sesion: Maneja el proceso de inicio de sesión de un inversor.

Parámetros:

email: Email del inversor.

contrasena: Contraseña del inversor.

Retorna: Una instancia de Inversor si la autenticación es exitosa.

CLASE ServiciodeCalculodeRendimientos se encarga de calcular el rendimiento de las acciones en un portafolio. Utiliza objetos DAO para interactuar con la base de datos y obtener la información necesaria.

Atributos:

transaccion_dao:

Tipo: TransaccionDAO

Descripción: DAO utilizado para obtener las transacciones de la base de datos.

cotizacion_dao:

Tipo: CotizacionDAO

Descripción: DAO utilizado para obtener las cotizaciones diarias de la base de datos.

Parámetros:

transaccion_dao: Instancia de TransaccionDAO.

cotizacion_dao: Instancia de CotizacionDAO.

calcular_rendimiento_por_accion: Calcula el rendimiento de una acción específica en un portafolio.

Parámetros:

id_portafolio: ID del portafolio.

id_accion: ID de la acción.

Retorna: El rendimiento calculado de la acción.

Proceso:

Obtiene las transacciones de compra para la acción en el portafolio.

Obtiene la última cotización de la acción.

Calcula el precio promedio de compra.

Calcula el rendimiento basado en el precio actual y el precio promedio de compra.

CLASE ServiciodeCompra maneja el proceso de compra de acciones para un inversor. Utiliza varios DAOs para interactuar con la base de datos y realizar las operaciones necesarias.

Atributos:

__dao_historial_saldo:

Tipo: HistorialSaldoDAO: DAO utilizado para registrar el historial de saldo del inversor.

__dao_cotizacion_diaria:

Tipo: CotizacionDAO: DAO utilizado para obtener la última cotización de la acción.

__dao_estado_portafolio:

Tipo: EstadoPortafolioDAO: DAO utilizado para actualizar el estado del portafolio del inversor.

__dao_transaccion:

Tipo: TransaccionDAO: DAO utilizado para registrar la transacción de compra.

__comision_broker:

Tipo: Decimal

Comisión del broker aplicada a la compra.

Parámetros:

dao_historial_saldo: Instancia de HistorialSaldoDAO.

dao_cotizacion_diaria: Instancia de CotizacionDAO.

dao_estado_portafolio: Instancia de EstadoPortafolioDAO.

dao_transaccion: Instancia de TransaccionDAO.

comision_broker: Comisión del broker.

__calcular_monto_compra: Calcula el monto total de la compra, incluyendo la comisión del broker.

Parámetros:

accion: La acción que se desea comprar.

cantidad: La cantidad de acciones a comprar.

Proceso:

Obtiene la última cotización de la acción.

Calcula el precio de compra actual.

Calcula el monto de la compra multiplicando el precio de compra actual por la cantidad.

Calcula la comisión del broker.

Suma el monto de la compra y la comisión para obtener el monto total.

__verificar_saldo_suficiente: Verifica si el inversor tiene suficiente saldo para realizar la compra.

Parámetros:

inversor: El inversor que desea realizar la compra.

monto_total: El monto total de la compra.

Proceso:

Compara el saldo de la cuenta del inversor con el monto total.

Si el saldo es insuficiente, lanza una excepción.

__obtener_fecha_actual: Obtiene la fecha actual en formato YYYY-MM-DD.

Retorno: Retorna la fecha actual como una cadena de texto.

realizar_compra: Realiza el proceso completo de compra de acciones.

Parámetros:

inversor: El inversor que desea realizar la compra.

accion: La acción que se desea comprar.

cantidad: La cantidad de acciones a comprar.

Proceso:

Calcula el monto total de la compra y la comisión.

Verifica si el inversor tiene suficiente saldo.

Actualiza el saldo del inversor.

Actualiza el estado del portafolio del inversor.

Registra la transacción en el historial de saldo.

Crea una nueva transacción de compra.

Retorno: Retorna True si la compra se realiza con éxito.

CLASE ServicioDeRegistro maneja el registro de nuevos inversores en el sistema. Utiliza un objeto de tipo InversorDAO para interactuar con la base de datos.

Atributos:

__inversor_dao:

Tipo: InversorDAO: DAO utilizado para interactuar con la base de datos de inversores.

Parámetros:

inversor_dao: Instancia de InversorDAO.

registrar_usuario: Registra un nuevo inversor en la base de datos.

Parámetros:

inversor: Instancia de Inversor que contiene los datos del nuevo inversor.

Proceso:

Verifica si el email del inversor ya está registrado.

Si el email está registrado, lanza una excepción.

Intenta crear el nuevo inversor en la base de datos.

Si la creación es exitosa, retorna True.

Si ocurre un error durante la creación, lanza una excepción con el mensaje de error.

CLASE ServiciodeVenta maneja el proceso de venta de acciones. Utiliza varios DAOs para interactuar con la base de datos y asegurar que todas las operaciones se registren correctamente.

Atributos:

__dao_historial_saldo:

Tipo: HistorialSaldoDAO: DAO utilizado para registrar el historial de saldo.

__dao_cotizacion_diaria:

Tipo: CotizacionDAO: DAO utilizado para obtener las cotizaciones diarias.

__dao_estado_portafolio:

Tipo: EstadoPortafolioDAO: DAO utilizado para obtener y actualizar el estado del portafolio.

__dao_transaccion:

Tipo: TransaccionDAO: DAO utilizado para registrar las transacciones de venta.

__comision_broker:

Tipo: Decimal

Comisión del broker aplicada a las ventas.

__monto_venta:

Tipo: Decimal

Monto total de la venta calculado.

Parámetros:

dao_historial_saldo: Instancia de HistorialSaldoDAO.

dao_cotizacion_diaria: Instancia de CotizacionDAO.

dao_estado_portafolio: Instancia de EstadoPortafolioDAO.

dao_transaccion: Instancia de TransaccionDAO.

comision_broker: Comisión del broker.

__calcular_monto_venta: Calcula el monto total de la venta, incluyendo la comisión del broker.

Parámetros:

accion: La acción que se desea vender.

__verificar_cantidad_acciones: Verifica si el inversor tiene suficientes acciones para vender.

Parámetros:

inversor: El inversor que desea realizar la venta.

cantidad_acciones: La cantidad de acciones a vender.

__obtener_fecha_actual: Obtiene la fecha actual en formato YYYY-MM-DD.

Retorno: La fecha actual como una cadena de texto.

realizar_venta: Realiza el proceso completo de venta de acciones.

Parámetros:

inversor: El inversor que desea realizar la venta.

accion: La acción que se desea vender.

cantidad_acciones: La cantidad de acciones a vender.

Proceso:

Verifica si el inversor tiene suficientes acciones.

Calcula el monto total de la venta y la comisión.

Actualiza el estado del portafolio del inversor.

Actualiza el saldo del inversor.

Registra la transacción en el historial de saldo.

Crea una nueva transacción de venta.

Retorno: True si la venta se realiza con éxito.