Taller de Sistemas de Información
Java

Iteración 1
Lógica de negocio de Pasarela Pagos

Introducción.

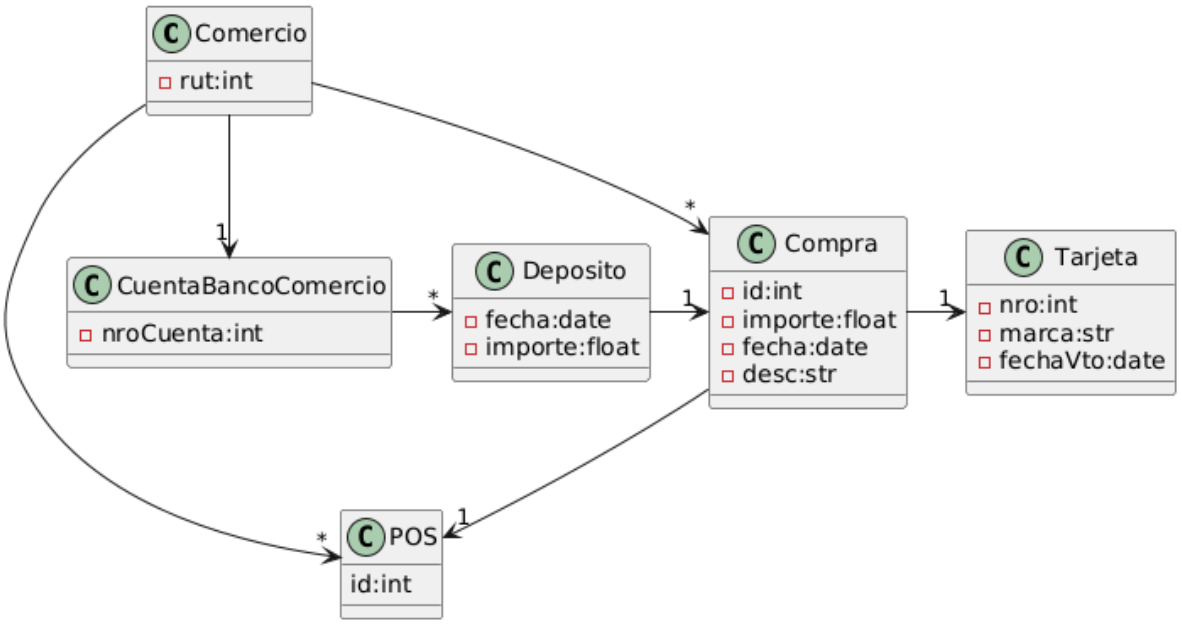
El siguiente documento incluye los detalles de la Tarea 1, a realizar en la materia Taller Java. Además contiene las interfaces que surgen de la etapa de diseño. Las mismas son una **lista tentativa de referencia**, la cual puede ser completada/modificada a medida que avance la implementación de las mismas.

Desarrollo de Pasarela de Pagos.

La pasarela de pagos es el sistema principal de nuestra solución. El mismo contará con una serie de módulos los cuales tendrán una responsabilidad bien definida.

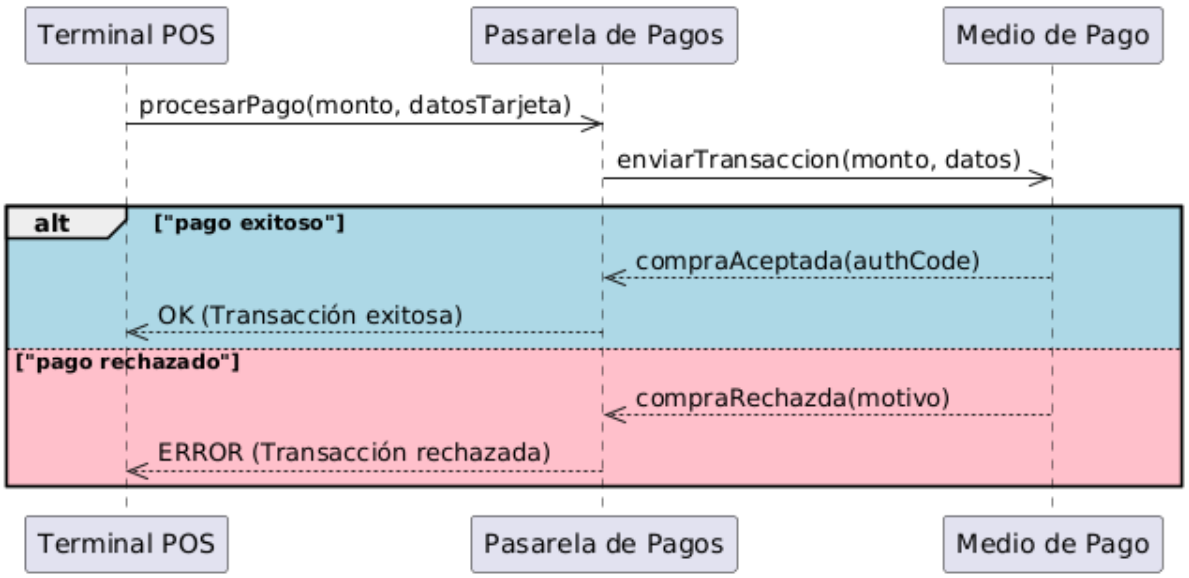
Análisis: modelo del dominio.

Un análisis preliminar identificó los siguientes conceptos. El modelo no pretende estar completo ni ser el definitivo.

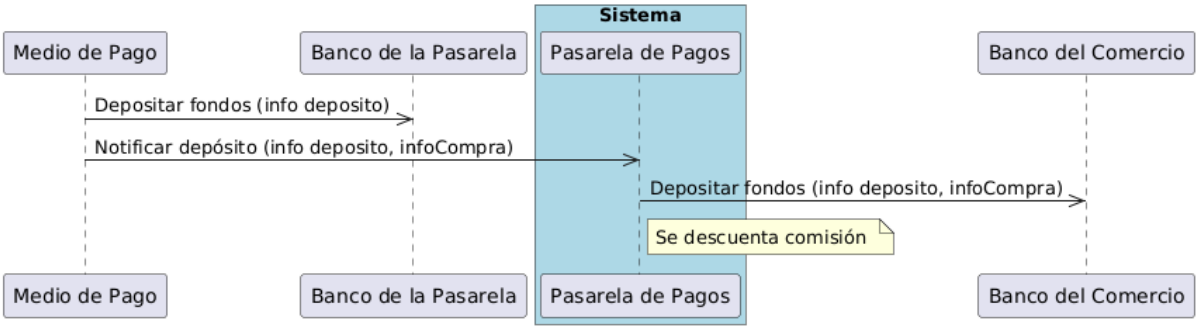


Diseño.

Casos de uso principal: procesarPago()



Caso de uso: recibirNotificacionTransferenciaDesdeTarjeta()



Lineamientos de diseño.

Los siguientes lineamientos de diseño tienen como objetivo crear una aplicación modular.

Cada módulo eventualmente puede evolucionar en una aplicación separada que ejecute independientemente (en otro espacio de memoria) del resto de los módulos.

Una evaluación natural de un diseño monolítico modular puede dar paso a una arquitectura de microservicios. Una arquitectura de microservicios lleva la capacidad de escalamiento de una aplicación a un nivel superior respecto a una arquitectura monolítica. Sin embargo, como requisito imprescindible para que esta transición sea exitosa, se deben de construir módulos desacoplados desde el inicio.

Uno de los objetivos de este taller es incursionar en el desarrollo de módulos desacoplados.

Estructura de paquetes.

La estructura de paquetes de cada módulo deberá ser la siguiente:

```
src
        modulo1
                dominio
                        repositorio
                aplicación
                interface
                infraestructura
                        configuración
                        persistencia

        modulo2
                …
```

dominio: clases que contienen la lógica del negocio
repositorio: interfaces para la persistencia de datos
aplicación: interfaces con los servicios que ofrece el módulo y su correspondiente implementación
interface:
- interface remotas para acceder a los servicios (no se implementan en esta etapa)
- interface para interactuar con otro módulos de forma local
- observadores de eventos

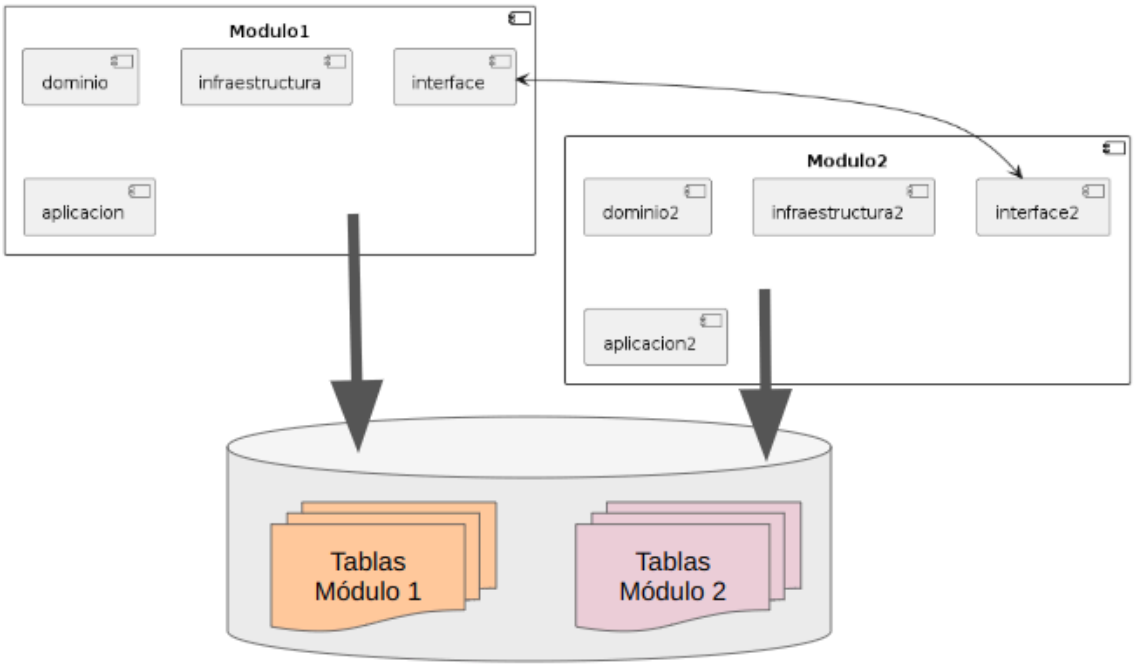
infraestructura: código transversal
persistencia: implementación de las interfaces del repositorio

Tenga en cuenta que existe en el repositorio del curso, un proyecto de Referencia donde existe una implementación base que muestra el concepto.

Acoplamiento entre módulos.

Cada módulo dependerá de la funcionalidad implementada en la interfaz definida en el paquete **interfaces**.
En otras palabras lo único que conocerá un módulo de otro será dicha interfaz.
Además cada módulo contará con su repositorio de datos exclusivo (misma base de datos pero en distinto esquemas)

Nota: es posible que los mismos objetos de dominio existan en diferentes módulos



Uso de eventos

Cuando un módulo necesite invocar una funcionalidad de otro módulo, considere siempre como primera opción el uso de eventos, especialmente si la operación no requiere el retorno de valores. El uso de eventos es la forma más desacoplada que podemos utilizar para la comunicación.

Uso de inyección de dependencias.

La resolución de dependencias en tiempo de ejecución deberá ser delegada el Contenedor de dependencias en todos los casos donde aplique.

Interfaces.

Las siguientes interfaces son producto del resultado del trabajo realizado en la etapa de diseño. Las misma tiene el carácter de tentativas y por lo que puede ser necesario modificar o agregar las que se consideren necesarias.

Módulo de Comercio.

altaComercio (datosComercio)	Permite registrar un comercio en el sistema.
modificarDatosComercio (datosComercio)	Permite modificar información relacionada al comercio.
altaPos (comercio, pos)	Registra un POS en el sistema.
cambiarEstadoPos (comercio, pos, estado)	Permite habilitar/deshabilitar un POS
cambioContraseña (nuevaPass)	Establece, cambia contraseña del comercio. La contraseña se utilizará para poder invocar a la API remota que ofrece información de ventas (a desarrollar en siguiente iteración)
realizarReclamo(textoReclamo)	Recibe un reclamo del comercio, el cual deberá ser atendido por el departamento de Soporte.

Módulo Compra

procesarPago (datosCompra)	procesa el pago de una compra realizada en el Comercio
resumenVentasDiarias (comercio);	retorna información de las ventas realizadas en el día
resumenVentasPorPeriodo (comercio);	retorna información de las ventas realizadas en un período de tiempo
montoActualVendido (comercio);	retorna el acumulado actual de las ventas del día del comercio.

Módulo de Transferencia.

recibirNotificacionTransferenciaDesdeMedioPago (datosTransferencia)	Se invoca desde el sistema de Medios de Pago, informando que se realizó un depósito relacionado a una compra realizada en un comercio. Cuando se recibe la notificación es necesario realizar un depósito en la cuenta del Banco del Cliente, descontado del importe la comisión del Sistema.
consultarDepositos (comercio, rangoFechas)	Devuelve los depósitos realizados para un comercio en un rango de fechas.

Módulo de Monitoreo

notificarPago()	Se invoca cada vez que se realiza un pago.
notificarPagoOk()	Se invoca cada vez que se realiza un pago exitoso.
notificarPagoError()	Se invoca cada vez que se rechaza un pago.
notificarTransferencia()	Se invoca cada vez que se realiza una transferencia desde el MedioDePago.
notificarReclamoComercio()	Se invoca cuando un comercio realizar un reclamo

Tareas a realizar.

Crear un proyecto web con la estructura de paquetes que refleje la realidad planteada
Para cada módulo:

- implementar las clases de dominio correspondientes
- implementa los casos de uso que se detallan en la sección Interfaces
- realizar los test unitarios de cada módulo
- realizar los test de integración de cada caso de uso

Se recomienda en una primera instancia que los test se realicen utilizando implementaciones de las clases Repositorios que obtengan datos desde memoria.
De esta manera se logra focalizar en entender el problema sin la complejidad agregada del acceso a datos en la base.

Tener en cuenta que las interfaces ofrecidas son producto de la etapa de diseño preliminar. En esta etapa deberá identificar si las mismas son suficientes y permiten la resolución del problema planteado. Es decir, en esta interacción se espera que se realice un Diseño detallado de la solución.

Los test de integración deberán de ofrecer la claridad necesaria para determinar si el dominio implementado, además de los casos de usos, representan una buena base para las iteraciones sucesivas.

Entrega.

1. La fecha de entrega de esta primera iteración está establecida en el calendario del curso.
2. Se debe entregar el link a un fork o branch del repositorio github utilizado.
3. Se debe entregar documentación de lo realizado; utilizar el archivo README.md del repositorio.
4. Aplicar los siguientes criterios para decidir que es importante y relevante documentar:
    a. priorice diagramas uml sobre texto (por ejemplo, diagrama de paquetes)
    b. contextualice la información ofrecida
    c. sea prolijo y mantenga uniformidad de formato
    d. jerarquice la información subtitulando contenido