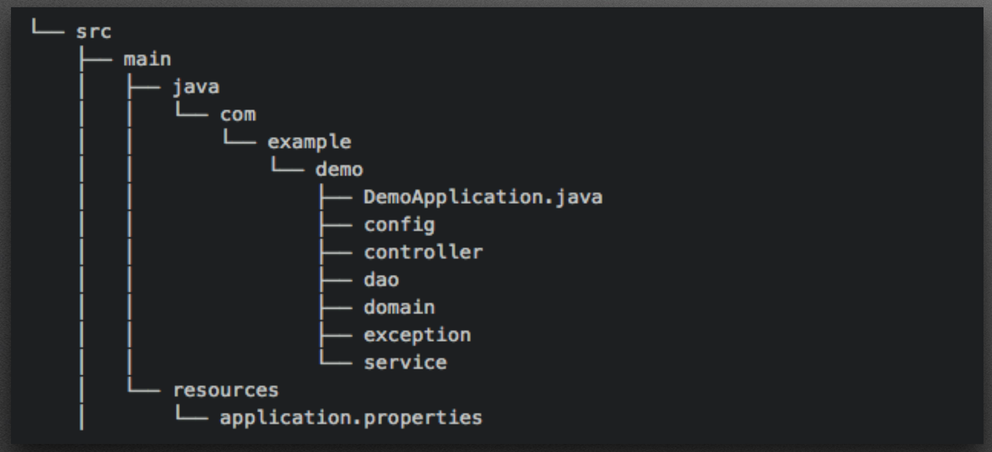
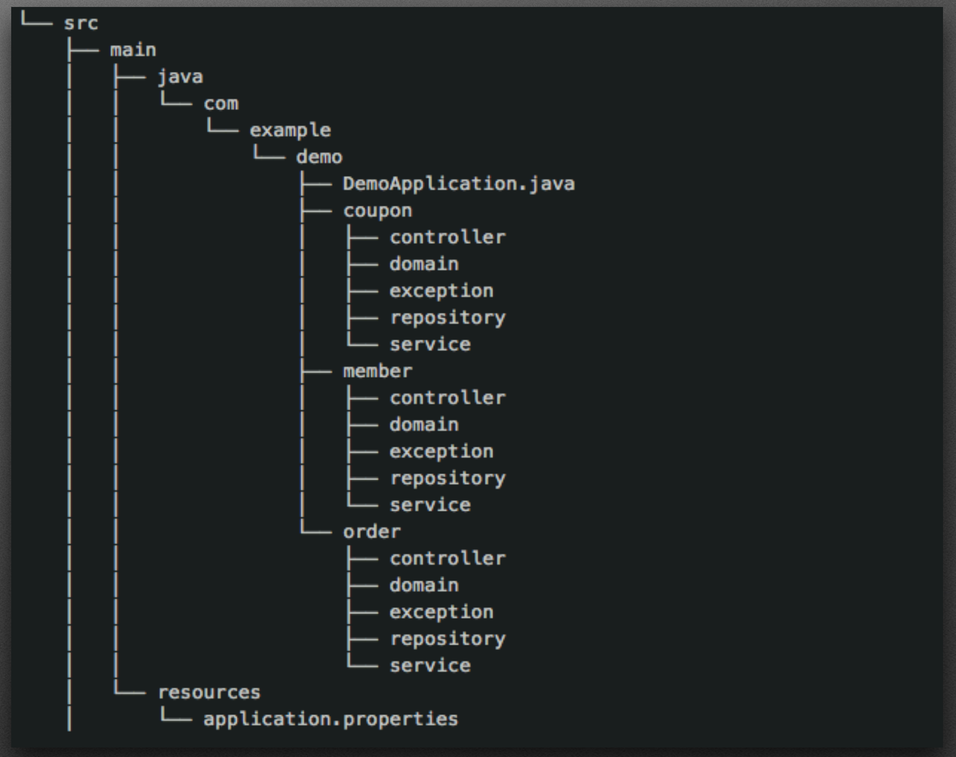
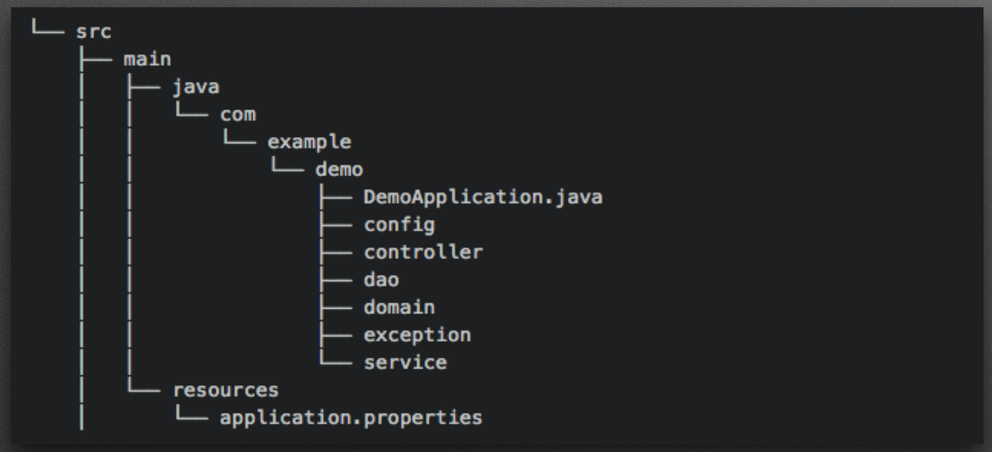
디렉토리 구조

Layer VS Domain





Layer



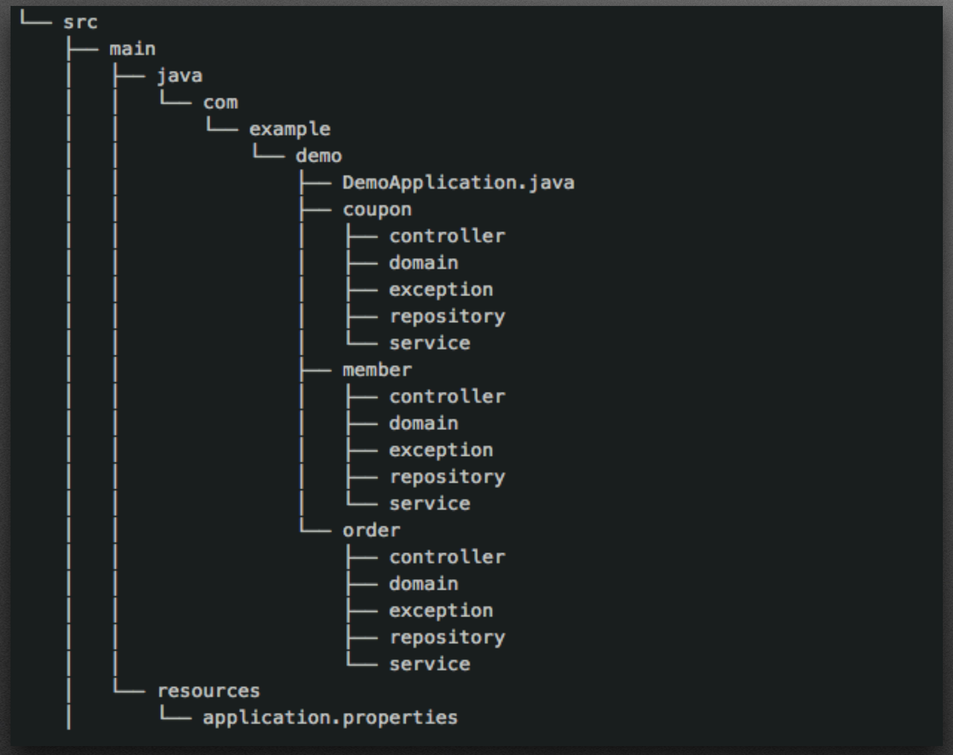
장점

* 프로젝트에 이해가 낮아도 전체적인 구조를 빠르게 파악 가능
* 작성 하고자하는 계층이 명확할 경우 빠르게 개발 가능

단점

* 각 레이어별로 수십개의 클래스들이 존재 하여 코드 파악이 어려움
* Layer를 기준으로 분리했기 때문에 코드의 응집력이 떨어짐

Domain

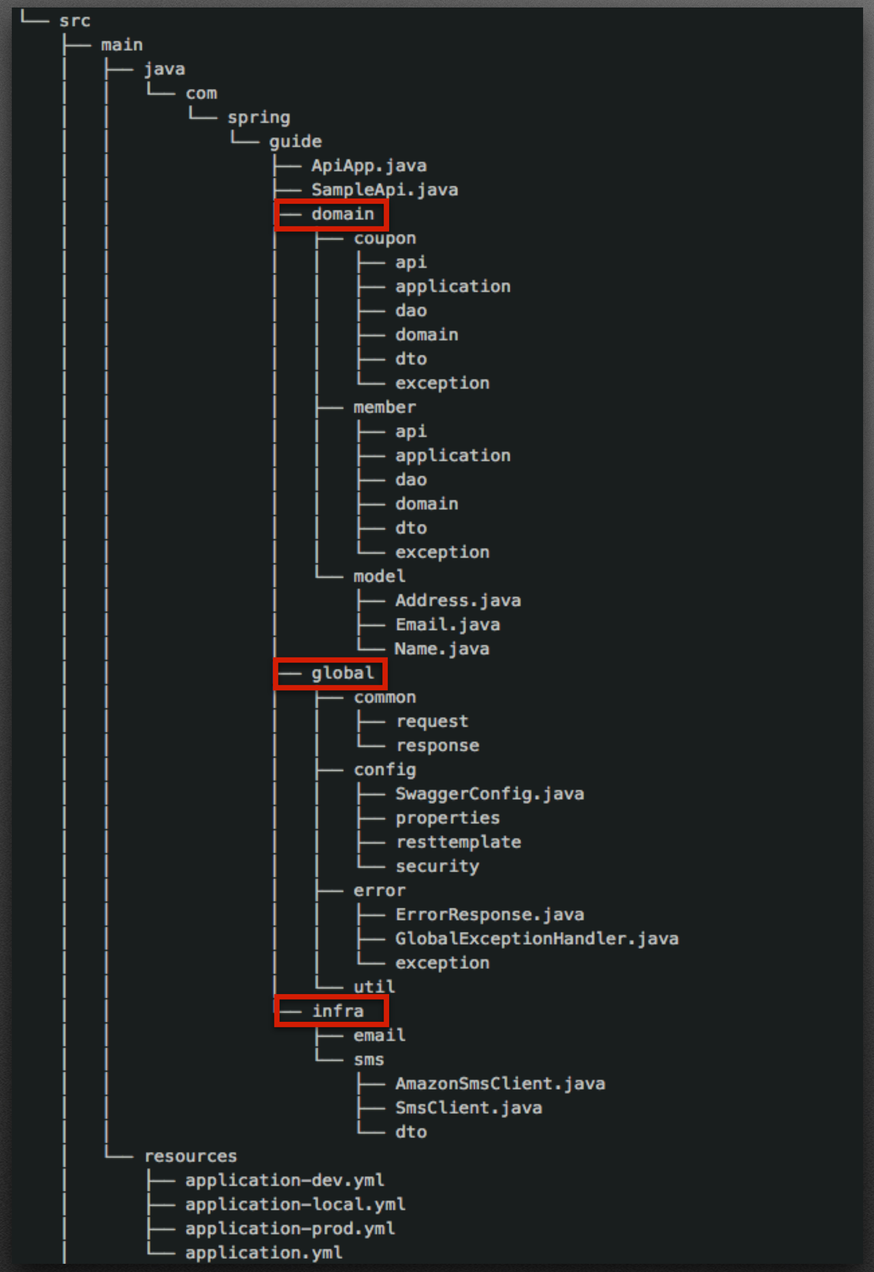


장점

* 관련된 코드들이 응집해 있음
* 디렉토리 구조를 통해 도메인을 이해할 수 있음

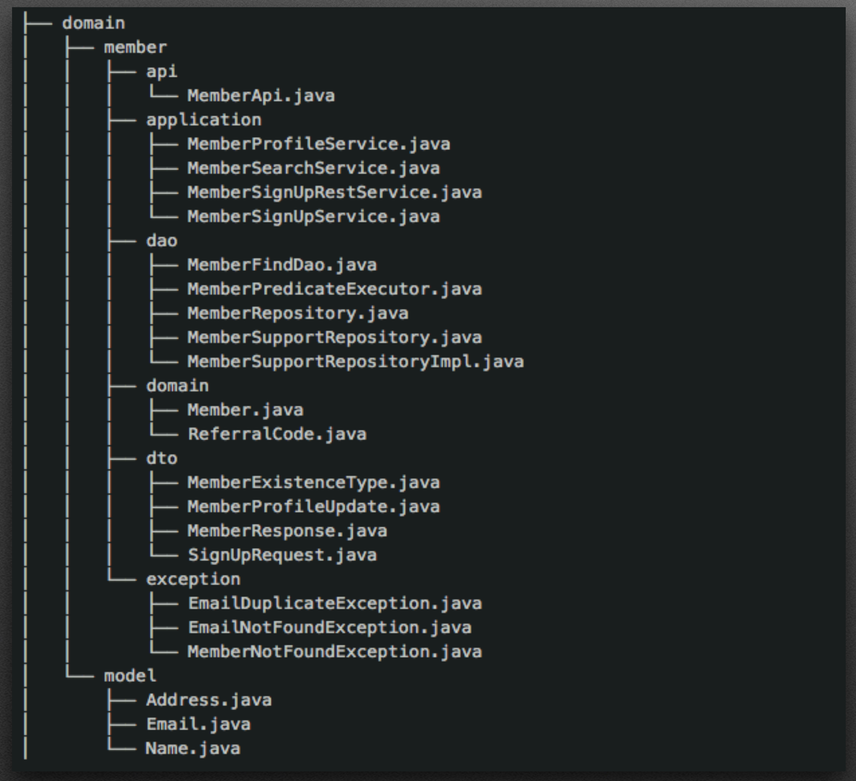
단점

* 도메인 지식 없이 이해하기 어려움
* 각 계층을 구분하기 위한 논의가 필요



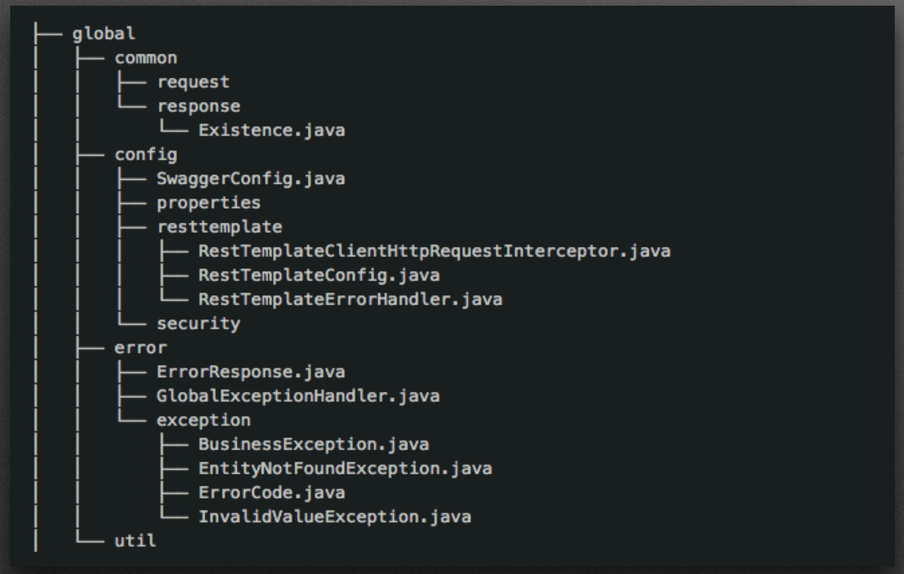
* domain : 도메인을 담당
* global : 프로젝트의 전체담당
* Infra: 외부 인프라스트럭처 담당

Domain



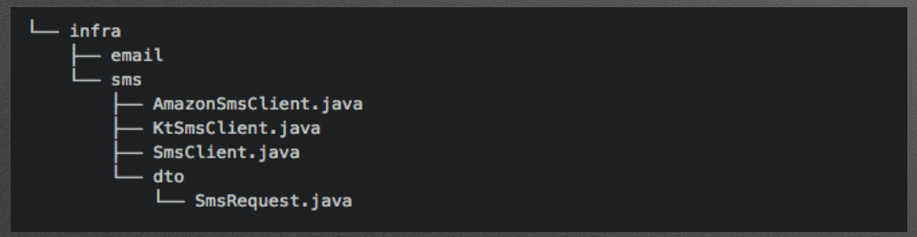
* api : 컨트롤러 클래스들이 존재합니다. 외부 rest api로 프로젝트를 구성하는 경우가 많으니 api라고 지칭했습니다.
* domain : 도메인 엔티티에 대한 클래스로 구성됩니다. 특정 도메인에만 속하는 Embeddable, Enum 같은 클래스도 구성됩니다.
* dto : 주로 Request, Response 객체들로 구성됩니다.
* exception : 해당 도메인이 발생시키는 Exception으로 구성됩니다.

global



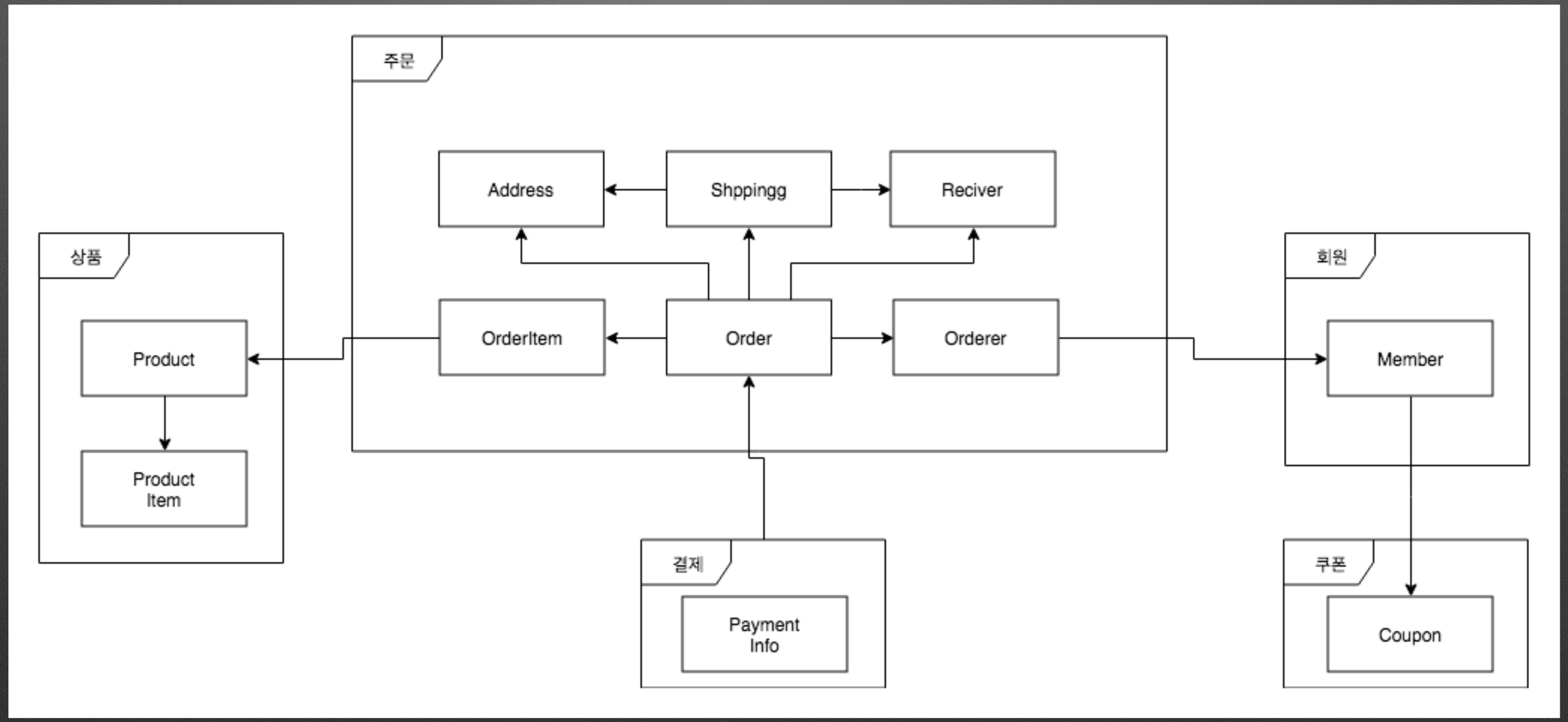
* global은 프로젝트 전방위적으로 사용되는 객체들로 구성됩니다.
* common : 공통으로 사용되는 Value 객체들로 구성됩니다. 페이징 처리를 위한 Request, 공통된 응답을 주는 Response 객체들이 있습니다.
* config : 스프링 각종 설정들로 구성됩니다.
* error : 예외 핸들링을 담당하는 클래스로 구성됩니다. Exception Guide에서 설명했던 코드들이 있습니다.
* util : 유틸성 클래스들이 위치합니다.

infra

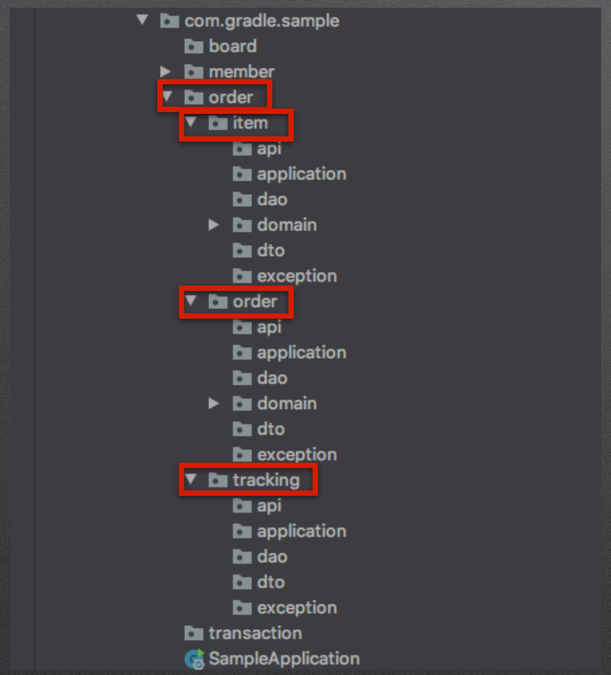


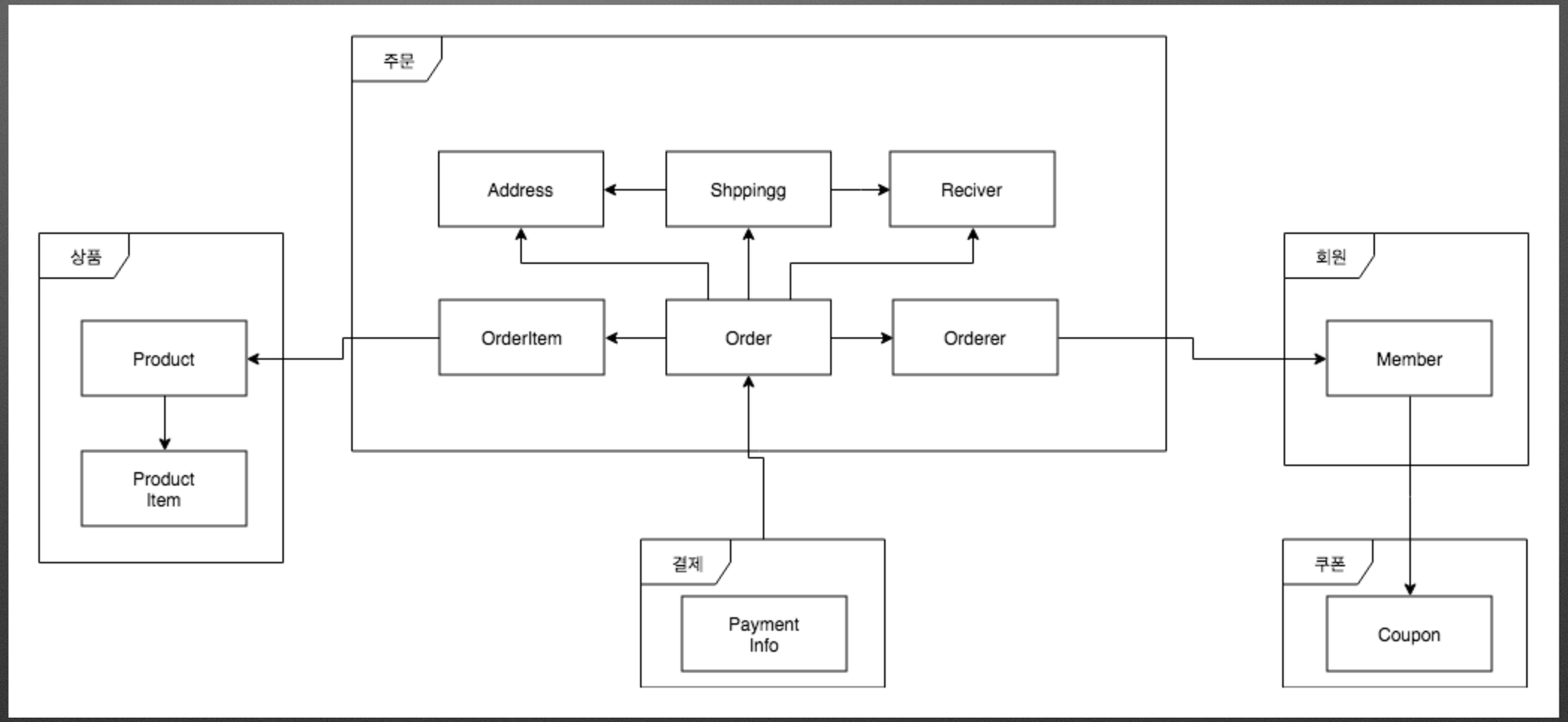
* infra 디렉터리는 인프라스트럭처 관련된 코드들로 구성됩니다.
* common : 공통으로 사용되는 Value 객체들로 구성됩니다. 페이징 처리를 위한 Request, 공통된 응답을 주는 Response 객체들이 있습니다.
* config : 스프링 각종 설정들로 구성됩니다.
* error : 예외 핸들링을 담당하는 클래스로 구성됩니다. Exception Guide에서 설명했던 코드들이 있습니다.
* util : 유틸성 클래스들이 위치합니다.

구조

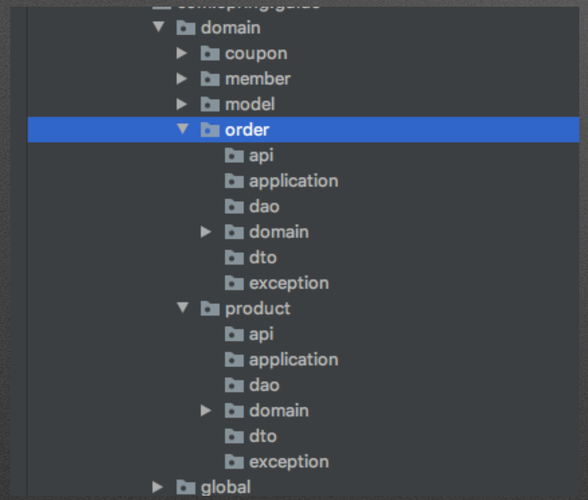

주문
Address
Shppingg
Reciver
OrderItem
Order
Orderer
상품
Product
Product
Item
회원
Member
결제
Payment
Info
쿠폰
Coupon

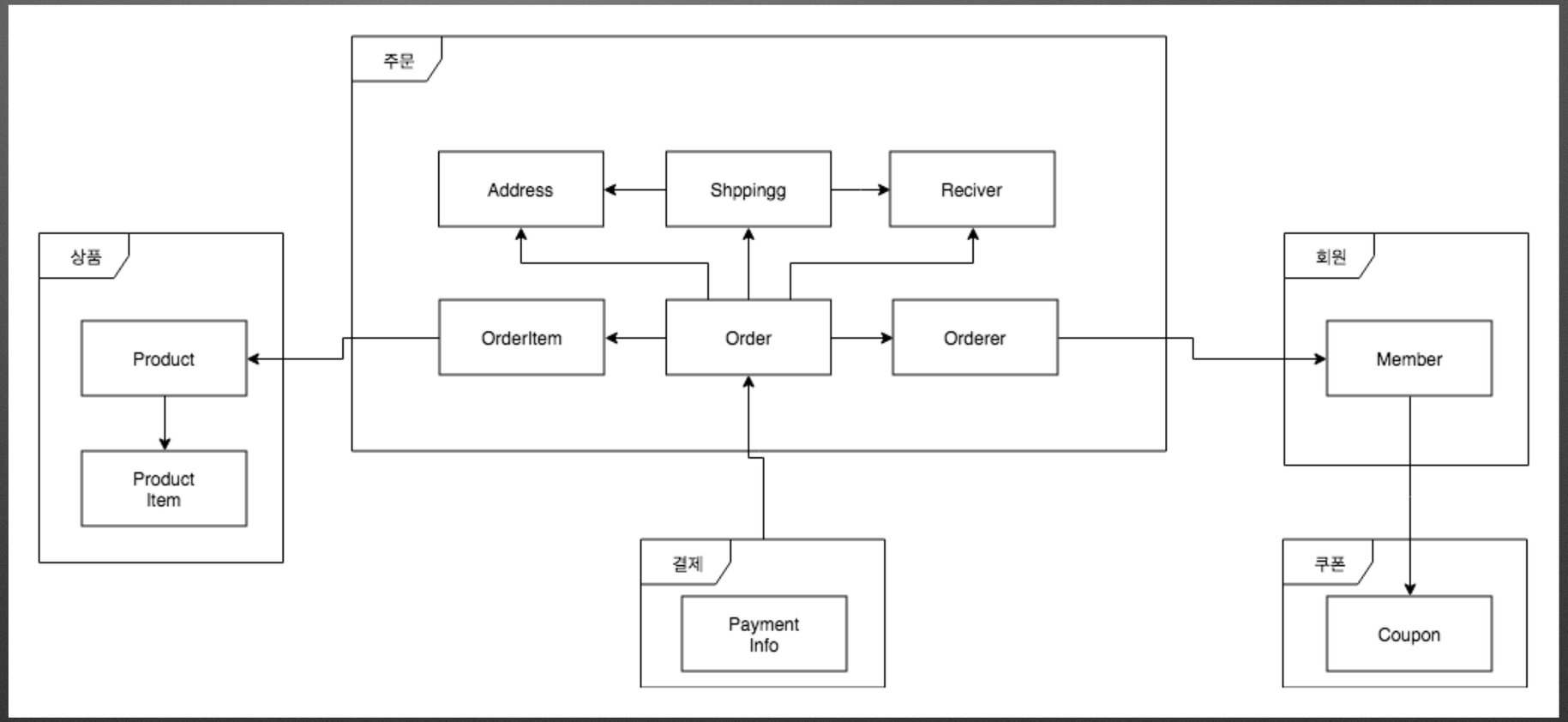
Repository


com.gradle.sample
board
member
order
item
api
application
dao
domain
dto
exception
order
api
application
dao
domain
dto
exception
tracking
api
application
dao
dto
exception
transaction
SampleApplication

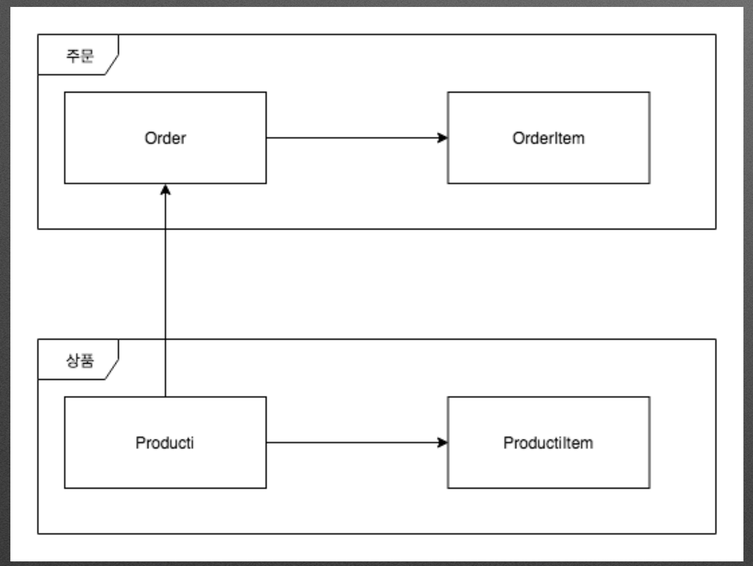

주문
Address
Shppingg
Reciver
OrderItem
Order
Orderer
상품
Product
Product Item
회원
Member
결제
Payment Info
쿠폰
Coupon

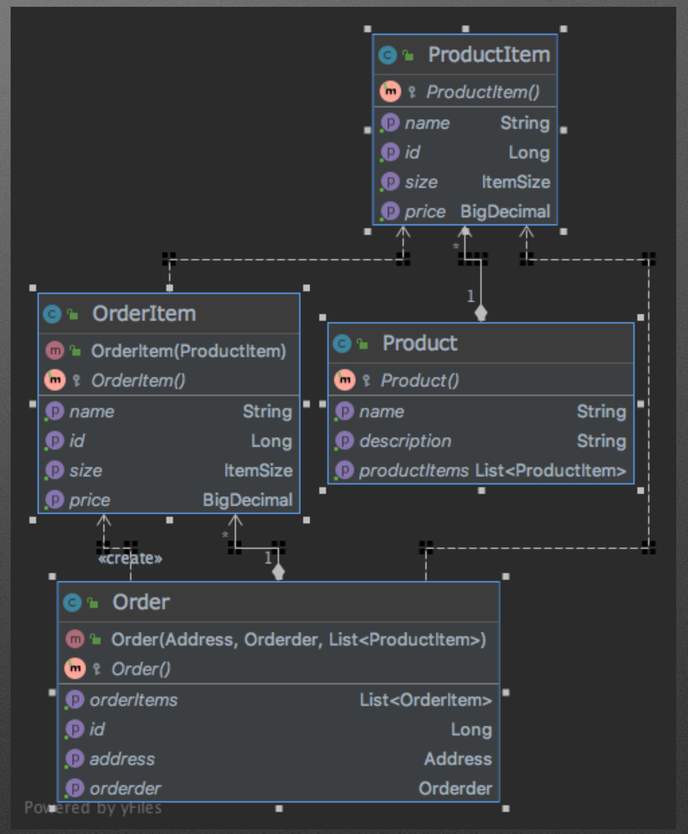
Aggregate


domain
coupon
member
model
order
api
application
dao
domain
dto
exception
product
api
application
dao
domain
dto
exception
global


주문
Address
Shppingg
Reciver
OrderItem
Order
Orderer
상품
Product
Product Item
회원
Member
결제
Payment Info
쿠폰
Coupon

강한 결합 관계


주문
Order
OrderItem
상품
Producti
ProductiItem


ProductItem
ProductItem()
name String
id Long
size ItemSize
price BigDecimal
OrderItem
OrderItem(ProductItem)
OrderItem()
name String
id Long
size ItemSize
price BigDecimal
Product
Product()
name String
description String
productItems List<ProductItem>
«create»
Order
Order(Address, Orderder, List<ProductItem>)
Order()
orderItems List<OrderItem>
id Long
address Address
orderder Orderder
Powered by yFiles

강한 결합 관계

```java
@Entity
@Table(name = "orders")
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Embedded
    private Address address;

    @Embedded
    private Orderder orderder;

    @ElementCollection
    @CollectionTable(name = "orders_item", joinColumns = @JoinColumn(name = "id"))
    private List<OrderItem> orderItems = new ArrayList<>();

    public Order(Address address, Orderder orderder, List<ProductItem> productItems) {
        Assert.notNull(address, message: "address must not be null");
        Assert.notNull(orderder, message: "address must not be null");
        Assert.notEmpty(productItems, message: "address must not be empty");

        this.address = address;
        this.orderder = orderder;
        for (ProductItem productItem : productItems) {
            orderItems.add(new OrderItem(productItem));
        }
    }
}
```

```java
@Embeddable
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class OrderItem {

    @Id
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;

    @Enumerated(EnumType.STRING)
    @Column(name = "size", nullable = false)
    private ItemSize size;

    @Column(name = "price", nullable = false)
    private BigDecimal price;

    public OrderItem(ProductItem productItem) {
        this.name = productItem.getName();
        this.size = productItem.getSize();
    }
}
```

```java
@Embeddable
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class ProductItem {

    @Id
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;

    @Enumerated(EnumType.STRING)
    @Column(name = "size", nullable = false)
    private ItemSize size;

    @Column(name = "price", nullable = false)
    private BigDecimal price;


}
```

* 주문은 상품이 있어야 주문이 가능하다 -> 상품 기반으로 주문을 만들어야 한다

객체 생성

```java
@Embeddable
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class Orderder {

    @Column(name = "member_id")
    private Long memberId;

    @Column(name = "orderer_name", nullable = false)
    private String name;

    private Orderder(Long memberId, String name) {
        Assert.notNull(name, message: "name must be not null");
        this.memberId = memberId;
        this.name = name;
    }

    public static Orderder memberOrderer(Long memberId, String name) {
        Assert.notNull(memberId, message: "memberId must be not null");
        return new Orderder(memberId, name);
    }

    public static Orderder nonMemberOrderer(String name) {
        return new Orderder( memberId: null, name);
    }
}
```

* 회원 주문이, 비회원 주문이 있다 -> 해당 요구사항을 최대한 코드로 해석 가능하게