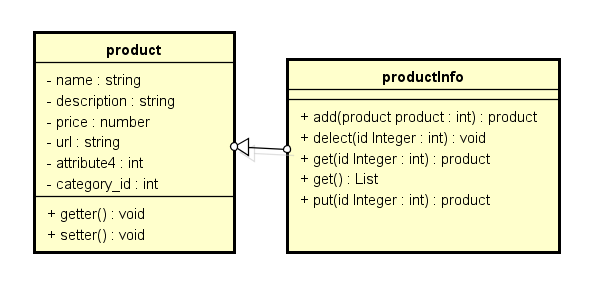
**面向对象分析**

## 一、商品面向对象分析

1. **类图设计**

****

 Product产品

属性:

* + id (integer): 商品唯一标识符
  + name (string): 商品名称
  + description (string): 商品描述
  + price (decimal): 商品价格
  + url (string): 商品链接
  + image (string): 商品图片链接
  + category\_id (integer): 分类ID，关联到Category类

方法:

* + addProduct(): 添加新商品
  + deleteProductById(id): 根据ID删除商品
  + getProductById(id): 根据ID获取商品详细信息
  + getAllProducts(): 获取所有商品列表
  + updateProductById(id, updatedDetails): 更新商品信息

 Category类别

属性:

* + id (integer): 分类唯一标识符
  + name (string): 分类名称

方法:

* + addCategory(): 添加新分类
  + deleteCategoryById(id): 根据ID删除分类
  + getCategoryById(id): 根据ID获取分类信息
  + getAllCategories(): 获取所有分类列表

 Order订单

属性:

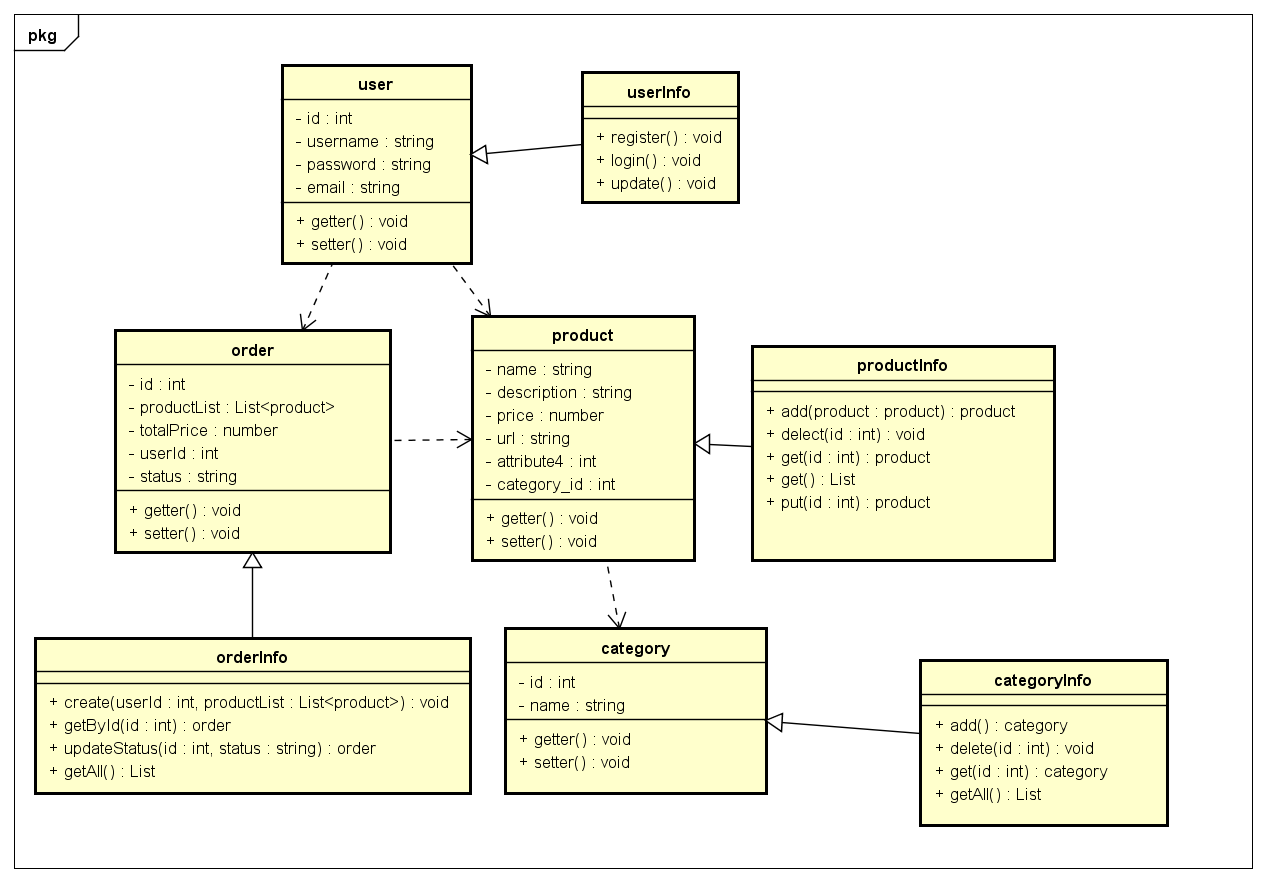
* + id (integer): 订单唯一标识符
  + productList (List<Product>): 订单中包含的商品列表
  + totalPrice (decimal): 订单总金额
  + userId (integer): 用户ID，关联到User类
  + orderStatus (string): 订单状态（如：待支付、已发货、已完成等）

方法:

* + createOrder(productList, userId): 创建新订单
  + getOrderById(id): 根据ID获取订单信息
  + updateOrderStatus(id, status): 更新订单状态
  + getAllOrders(): 获取所有订单列表

****

1. **关系分析**



* **Product**与**Category**之间存在关联关系。每个商品属于一个特定的分类（通过category\_id）。
* **Order**类与**Product**类之间存在聚合关系。一个订单可以包含多个商品。
* **User**类与**Order**类之间存在一对多的关系，一个用户可以拥有多个订单。

**3. 设计考虑**

* **可扩展性**: 设计时考虑将来可能需要添加更多功能，如支付处理、库存管理等。可以通过引入新的类（如Payment和Inventory）来实现。
* **安全性**: 用户的密码应该进行加密存储，敏感信息的处理需要遵循最佳安全实践。
* **用户体验**: 商品查询、分类管理等功能应优化以提高用户体验。例如，提供搜索和筛选功能。
* **异常处理**: 应在每个方法中考虑异常情况，例如商品不存在、库存不足等，并提供友好的错误信息。

**4. 业务逻辑示例**

* **添加商品**: 使用addProduct()方法，将商品信息存入数据库，并返回结果。
* **查询商品**: 使用getAllProducts()获取所有商品，支持分页和过滤。

## 二、用户面向对象分析

1. **类图设计**

User

id: int  
 username: String  
 password: String

UserService

register(user: User): void  
login(username: String, password: String): Boolean  
getUserById(id: String): User   
getAllUsers(): List<User>

属性:

`id`: 用户唯一标识符。

`username`: 用户名。

`password`: 用户密码。

方法:

`login(username: String, password: String)`: 用于用户登录，返回布尔值表示是否成功。

`register(user: User)`: 用于用户注册。

`getUserById`(id: String): User 用于根据id查询用户

`getAllUsers()`: List<User> 用于查询所有用户

Admin  
id: int  
username: String  
password: String

AdminService

registerAdmin(admin: Admin): void  
loginAdmin(username: String, password: String): Boolean

属性:

`id`: 管理员唯一标识符。

`username`: 管理员用户名。

`password`: 管理员密码。

方法:

`login(admin: Admin)`: 管理员登录。

`register(username: String, password: String)`: 管理员注册。

1. **关系分析**

（1）User 与 Admin:

两者都是用户，但角色不同。可以使用继承关系，Admin 类可以继承自 User 类，这 样 Admin 将拥有 User 的所有属性和方法，同时可以扩展特定管理员功能。

（2）User 与 UserService:

User通过 UserService进行用户相关操作：注册、登录等。可以说 UserService 有一个 User 的 组合关系。

（3）Admin 与 AdminService:

Admin通过 AdminService进行管理员相关操作：注册、登录、查询用户等。同样地， AdminService具有对 Admin 的组合关系。

1. **设计考虑**

（1）安全性:

密码存储时应加密，避免明文存储。例如，可以使用 bcrypt 等加密算法。

登录时应实施防止暴力攻击的策略，例如限制尝试次数。

（2）可扩展性:  
 -通过定义接口或抽象类来允许未来可能对服务的扩展，例如添加新的支付方式、用户类型、 优惠策略等。

（3）数据一致性:

在进行注册或更新用户信息时，应该先检查数据是否有效（例如，用户名唯一性、电子邮 件格式等）。

（4）服务分层:  
 -通过将业务逻辑分为不同的服务层，确保业务逻辑与表现逻辑的分离，这也便于维护和测 试。

（5）异常处理:

在用户注册和登录流程中，应该处理各种可能的异常情况，例如网络错误、数据库错误等， 给予用户清晰的错误反馈。

1. **业务逻辑实例**

（1）用户注册:

用户提交注册表单 -> UserService.registerUser(user) 被调用 -> 检查用户名是否唯一 -> 如果唯一，密码进行加密 -> 用户信息存入数据库。

（2）用户登录:

用户输入用户名和密码 -> UserService.loginUser(username, password) 被调用 -> 查询用 户 -> 验证输入的密码和数据库中的哈希值 -> 如果匹配，返回登录成功；否则返回失败 信息。

（3）管理员查询所有用户:

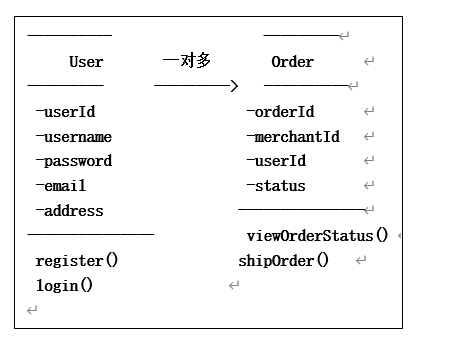
管理员发起查询请求 -> AdminService.getAllUsers() 被调用 -> 从数据库中获取所有用户 记录并返回。

（4）根据 ID 查询用户:

管理员输入用户 ID -> AdminService.getUserById(id) 被调用 -> 根据 ID 查询用户信息 -> 将用户信息返回给管理员。

## 三、订单面向对象分析

**1.类图设计**



 Product产品

属性:

* + id (integer): 商品唯一标识符
  + name (string): 商品名称
  + description (string): 商品描述
  + price (decimal): 商品价格
  + url (string): 商品链接
  + image (string): 商品图片链接
  + category\_id (integer): 分类ID，关联到Category类

方法:

* + addProduct(): 添加新商品
  + deleteProductById(id): 根据ID删除商品
  + getProductById(id): 根据ID获取商品详细信息
  + getAllProducts(): 获取所有商品列表
  + updateProductById(id, updatedDetails): 更新商品信息

 Category类别

属性:

* + id (integer): 分类唯一标识符
  + name (string): 分类名称

方法:

* + addCategory(): 添加新分类
  + deleteCategoryById(id): 根据ID删除分类
  + getCategoryById(id): 根据ID获取分类信息
  + getAllCategories(): 获取所有分类列表

 Order订单

属性:

* + id (integer): 订单唯一标识符
  + productList (List<Product>): 订单中包含的商品列表
  + totalPrice (decimal): 订单总金额
  + userId (integer): 用户ID，关联到User类
  + orderStatus (string): 订单状态（如：待支付、已发货、已完成等）

方法:

* + createOrder(productList, userId): 创建新订单
  + getOrderById(id): 根据ID获取订单信息
  + updateOrderStatus(id, status): 更新订单状态
  + getAllOrders(): 获取所有订单列表

****

**2. 关系分析**

* User ↔ Order：1对多（一个用户可以有多个订单）
* ShoppingCart ↔ Product：多对多（一个购物车可以有多个商品，一个商品可以被多个购物车引用）。
* User类与Order类之间存在一对多的关系，一个用户可以拥有多个订单。

**3. 设计考虑**

* **可扩展性**: Payment: 用于处理支付相关的功能，包括支付方式的选择、支付确认、退款等。
* **安全性**: 在User类中，用户的密码应使用安全的算法进行加密存储，避免明文存储。
* **用户体验**: 商品查询、分类管理等功能应优化以提高用户体验。例如，提供搜索和筛选功能。
* **异常处理**: 应在每个方法中考虑异常情况，例如商品不存在、库存不足等，并提供友好的错误信息。

**4. 业务逻辑示例**

* **创建订单**: 使用createOrder()方法，生成新订单，并自动更新商品库存。