第3章 桥接模式

提出问题

> 问题描述：设计一个程序能够画出各种几何形状，例如圆形、矩形、三角形等。

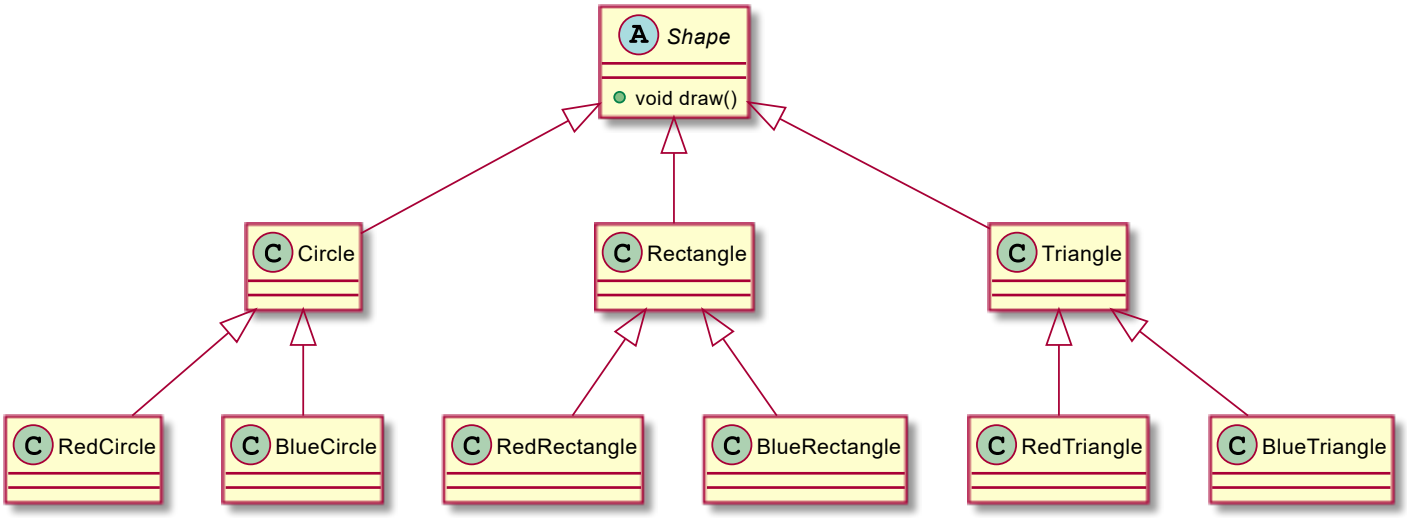
设计一个几何形状类以及具体的实现类：

```
public abstract class Shape {
    public abstract void draw();
}
public class Rectangle extends Shape{
    @Override
    public void draw() {
        System.out.println("画一个矩形");
    }
}
public class Circle extends Shape{
    @Override
    public void draw() {
        System.out.println("画一个圆形");
    }
}
public class Triangle extends Shape{
    @Override
    public void draw() {
        System.out.println("画一个三角形");
    }
}
```

> 假设现在需要绘制红色和蓝色的形状。如果不希望修改原系统代码，这里只有采用继承方式扩展，以圆形为例：

```
public class RedCircle extends Circle {
    @Override
    public void draw() {
        super.draw();
        System.out.println("填充红色");
    }
}
public class BlueCircle extends Circle {
    @Override
    public void draw() {
        super.draw();
        System.out.println("填充蓝色");
    }
}
```

完整类图如下：



由此可见，通过继承的方式扩展有时并不灵活，抽象与实现之间是强耦合关系，扩展的灵活性和代码复用性较差，不同的需求会导致类数量剧增。

模式名称

桥接（Bridge）或则柄体分离（handle/body）

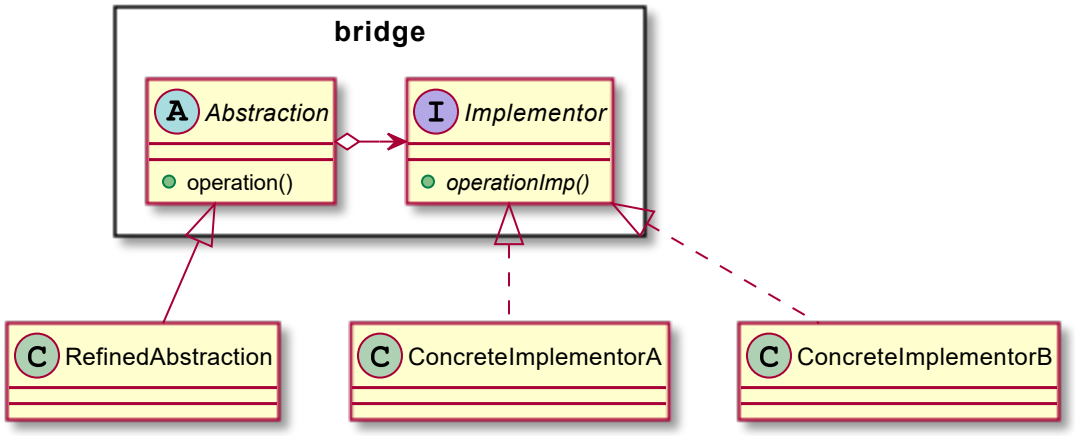
设计意图

桥接模式力图将系统的抽象部分与实现部分解耦，使得这两个部分中的一部分发生变化时都不会

影响对方。

> Decouple an abstraction from its implementation so that the two can vary independently.

设计结构

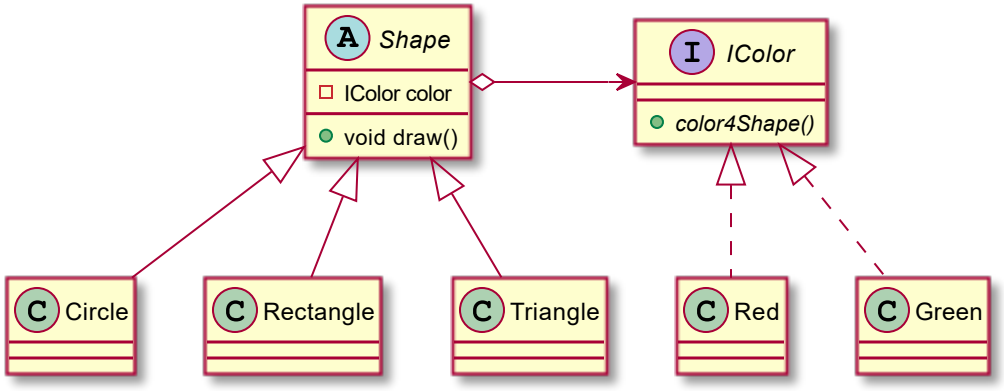


参与者主要包括抽象和实现两部分：

- 抽象（Abstraction）：
- 修正抽象（RefinedAbstraction）：抽象的具体实现，对抽象进行完善和扩展
- 实现接口（Implementor）：确定实现的基本操作，一般为接口或抽象类。
- 具体实现者（ConcreteImplementor）：具体实现。

解决问题

针对形状扩展问题，首先要划定抽象和实现两部分，对于形状对象来说，它的几何性质决定了它的基础绘制方式，因此可以将形状的几何性质（圆形、矩形、三角形等）归为抽象部分，其他的绘制形式归为实现部分。那么类图设计为：



效果与适用性

优点

- 抽象与实现部分解耦。支持运行时动态配置抽象的实现方式。实现部分的修改不会影响抽象部分，不需要对抽象部分进行重新编译。抽象与实现分离有助于分层，使得系统结构更好。
- 扩展性改善。抽象部分和实现部分独立扩展。
- 实现部分对客户端透明。客户端不用关心实现细节。

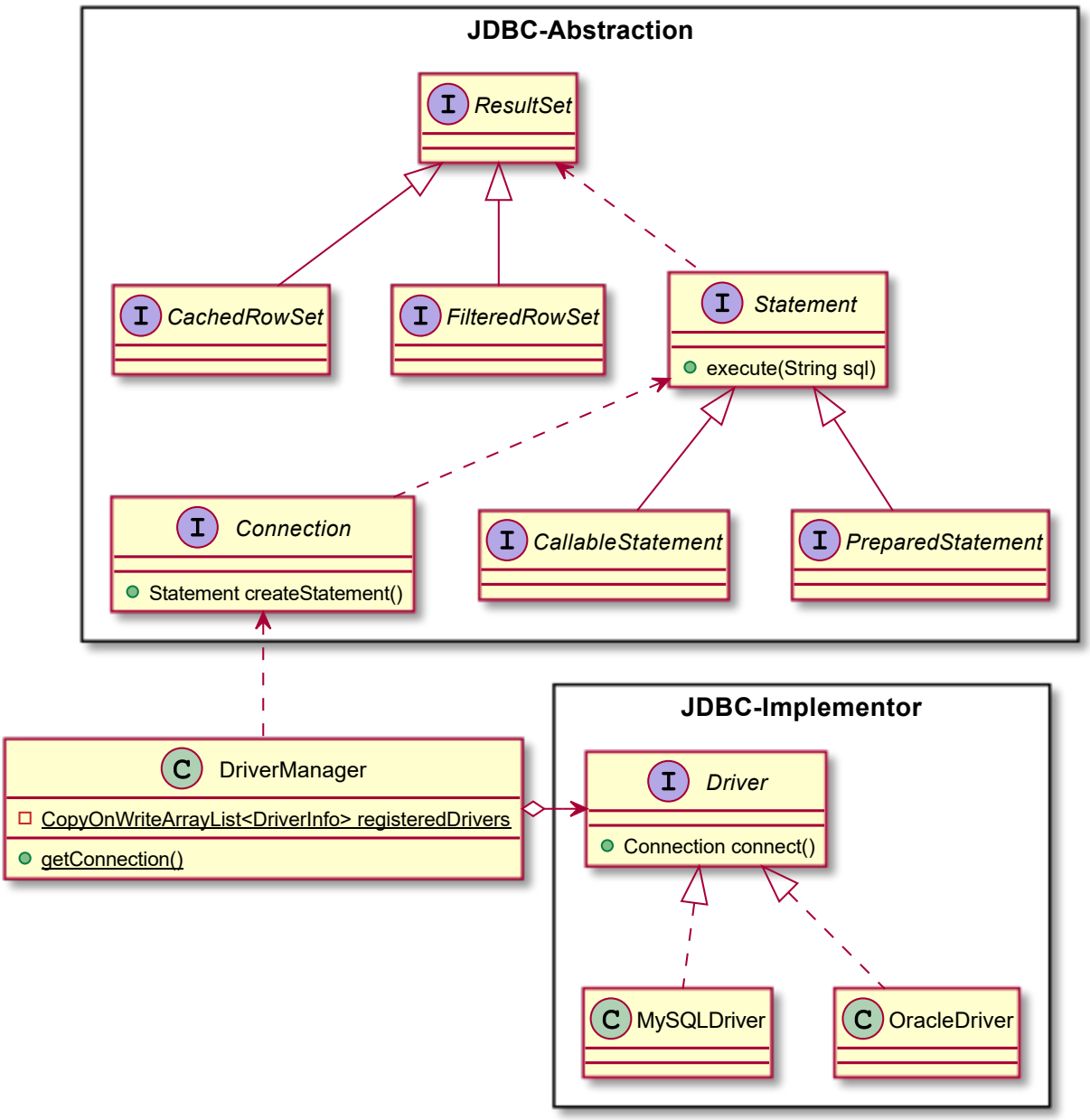
适用性

- 在抽象和实现之间需要增加更多灵活性的场景。
- 一个类存在多个独立变化维度，这两个维度都需要独立进行扩展。
- 不希望使用继承，或因多层继承导致类的数量剧增。

扩展案例

JDBC

JDBC中定义了数据库操作接口（`Connection`、`Statement`、`ResultSet`），将具体操作实现通过的Driver接口开放出来，由各数据库厂家自己实现。类结构如下：



分析类结构图得出，JDBC的抽象部分定义了一套操作数据库的接口，具体由数据厂商实现。`Driver`是数据库操作的入口，定义了数据库连接操作，`DriverManager`充当桥梁作用，既代理JDBC（或JDBC程序）持有有效的驱动，又能调用驱动实例创建一个连接对象（`Connection`），连接对象创建一个查询语句对象（`Statement`），执行SQL语句返回结果对象（`ResultSet`）。

创建数据库连接并执行查询的一般代码结构：

```
Class.forName("com.mysql.jdbc.Driver");
String url="jdbc:mysql://localhost:3306/db?user=root&password=pd";
Connection con = DriverManager.getConnection(url);
Statement stmt = con.createStatement();
String query = "select * from test";
ResultSet rs = stmt.executeQuery(query);
```

注意分析这里的驱动是如何注册的，`DriverManager`又是如何持有驱动的。关键在于驱动类加载时一般会执行一段静态代码：

```
public class Driver extends NonRegisteringDriver implements java.sql.Driver {
    static {
        try {
            java.sql.DriverManager.registerDriver(new Driver());
        } catch (SQLException E) {
            throw new RuntimeException("Can't register driver!");
        }
    }
}
```

日志管理

> 系统运行有数据库日志和用户操作日志两种，两种日志都可以用XML或TXT两种格式存储，讨论如何设计一个便于扩展的日志记录程序。

基于桥接模式的设计如下：