桥接模式

2020年2月13日 13:28

1. 概念

将抽象部分与它的具体实现部分分离，使它们都可以独立地变化。通过组合的方式建立两个类之间的联系，而不是继承。

主要解决的问题是在有多种可能会变化的情况下，用继承会造成类爆炸问题，扩展起来不灵活。

其中抽象类常为抽象类，实现类为接口。

2. UML结构图

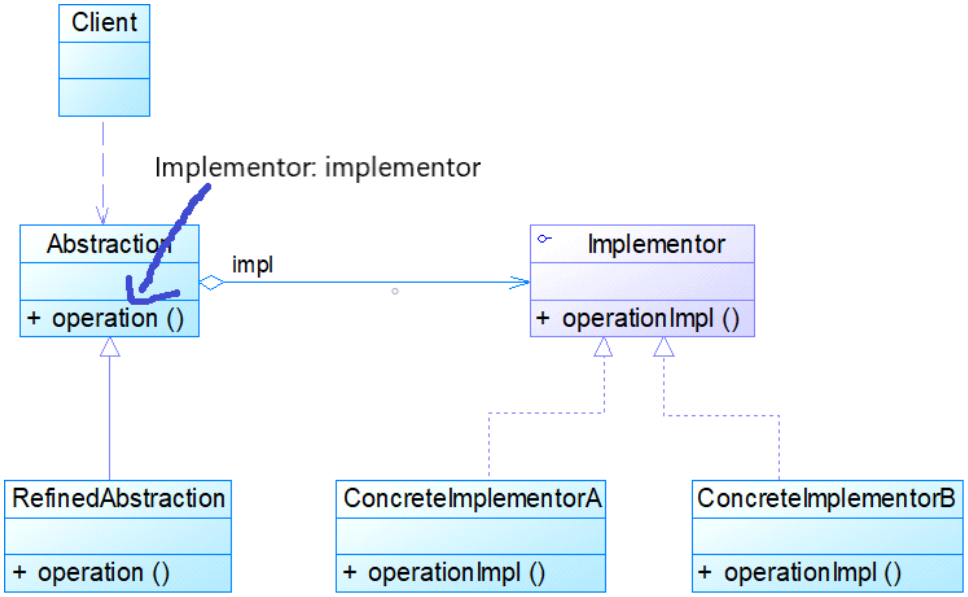

图2.1 桥接模式UML图

3. 桥接模式实例

我国银行有中国农业银行，中国工商银行，中国建设银行等等，对于银行账户也分为定期账户和活期账户，若采用继承的方式（例如中国农业银行继承银行抽象类，中国农业银行活期账户继承中国农业银行.....）进行构造，将会造成类爆炸。遂采用桥接模式将两者进行分离构造。

1. 抽象类

```
public abstract class Bank {
    public Account account;
    public Bank(Account account) {
        this.account = account;
    }
    public abstract Account openAccount();
}
```

2. 扩充抽象类

```
public class ABCBank  extends Bank{
    public ABCBank(Account account) {
        super(account);
    }

    @Override
    public Account openAccount() {
        System.out.println("打开中国农业银行账号");
        account.openAccount();
        return account;
    }
}
```

```
public class ICBCBank  extends Bank{
```

```
    public ICBCBank(Account account) {
        super(account);
    }

    @Override
    public Account openAccount() {
        System.out.println("打开中国工商银行账号");
        account.openAccount();
        return account;
    }
}
```

3. 实现接口

```
public interface Account {
    Account openAccount();
    void showAccountType();
}
```

4. 具体实现类

```
public class DepositAccount implements Account {
    @Override
    public Account openAccount() {
        System.out.println("打开定期账号");
        return new DepositAccount();
    }

    @Override
    public void showAccountType() {
        System.out.println("这是一个定期账号");
    }
}
```

```
public class SavingAccount  implements Account{

    @Override
    public Account openAccount() {
        System.out.println("打开定期账号");
        return new SavingAccount();
    }

    @Override
    public void showAccountType() {
        System.out.println("这是一个活期账号");
    }
}
```

5. 客户端

```
public class bridge {
    @Test
    public void bridgeTest(){
        Bank icbcBank = new ICBCBank(new DepositAccount());
        Account icbcAccount = icbcBank.openAccount();
        icbcAccount.showAccountType();

        Bank abcBank = new ABCBank(new SavingAccount());
        Account abcAccount = abcBank.openAccount();
        abcAccount.showAccountType();
    }
}
```

6. 结果

```
打开中国工商银行账号
打开定期账号
这是一个定期账号
打开中国农业银行账号
打开定期账号
这是一个活期账号
```

4. 优/缺点

1. 优点

- 分离抽象部分及其具体实现部分。
- 提高了系统的可扩展性。
- 符合开闭原则。

- 符合合成复用原则。

2. **优点**

- 增加了系统的理解和设计难度。
- 需要正确地识别出系统两个独立变化的维度。

5. 使用场景

1. 抽象和具体实现之间增加更多的灵活性。
2. 一个类存在存在（或多个）独立变化的维度，且这两个（或多个）维度都需要独立进行扩展。
3. 不希望使用继承，或因为多层继承导致系统类的个数剧增。

6. 参考资料

[1]. https://www.runoob.com/design-pattern/bridge-pattern.html