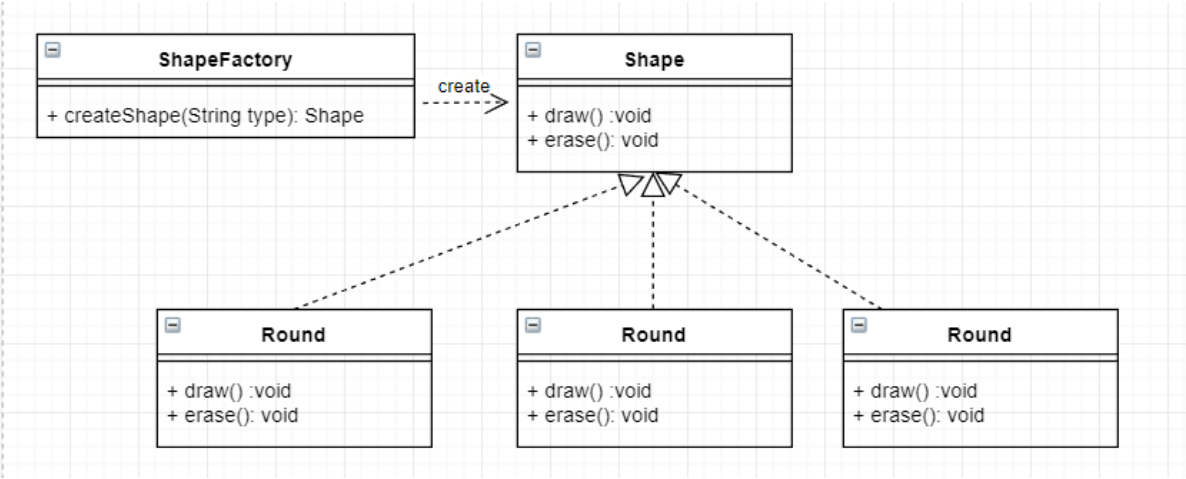
设计模式作业3

1.


ShapeFactory
+ createShape(String type): Shape
create
Shape
+ draw() :void
+ erase(): void
Round
+ draw() :void
+ erase(): void
Round
+ draw() :void
+ erase(): void
Round
+ draw() :void
+ erase(): void

```java
public abstract class AbstractFactory {
    public static Shape createShape(String type) throws Exception {
        Shape shape;
        if ("圆形".equals(type)) {
            shape = new Round();
        }
        else if ("方形".equals(type)) {
            shape = new Square();
        }
        else if ("三角形".equals(type)) {
            shape = new Triangle();
        }
        else {
            throw new Exception("UnSupportedShapeException");
        }
        return shape;
    }

}
```

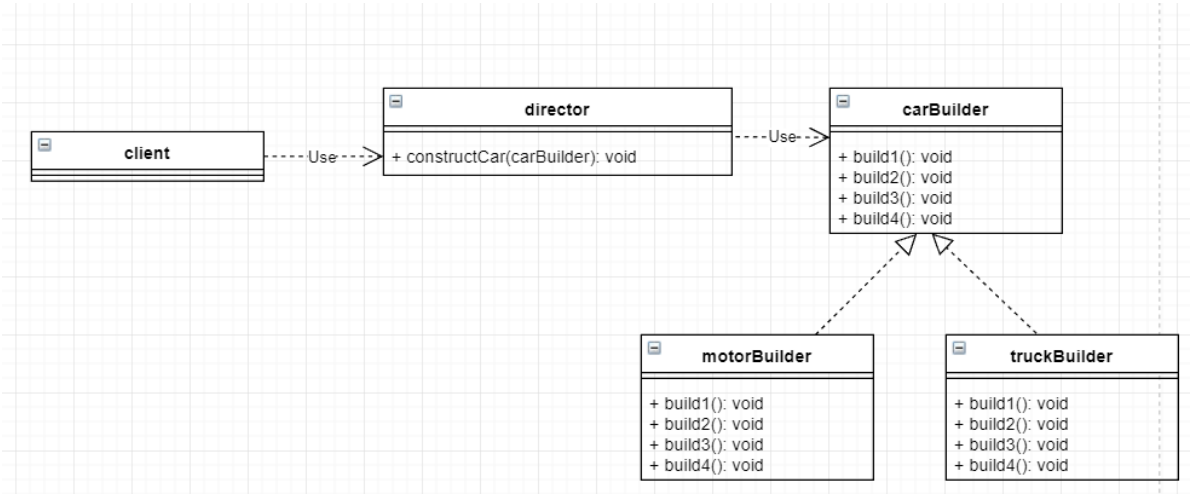
```java
public class Round extends Shape {
    @Override public void draw() { System.out.println("绘制圆形"); }
    @Override public void erase() { System.out.println("擦除圆形"); }
}
```

```java
public abstract class Shape {
    3 implementations
    public abstract void draw();
    3 implementations
    public abstract void erase();
}
```

```java
public class Square extends Shape {
    @Override public void draw() { System.out.println("绘制方形"); }
    @Override public void erase() { System.out.println("擦除方形"); }
}
```

```java
public class Triangle extends Shape {
    @Override
    public void draw() { System.out.println("绘制三角形"); }

    @Override
    public void erase() { System.out.println("擦除三角形"); }
}
```

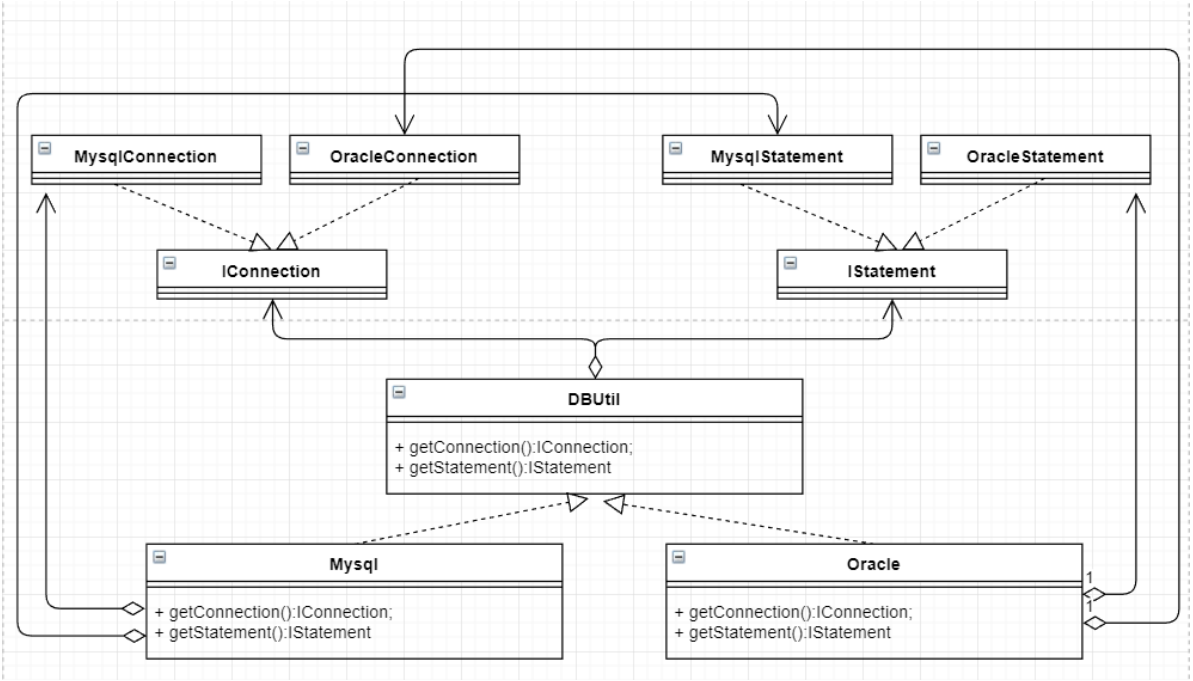
2.


client
Use
director
+ constructCar(carBuilder): void
Use
carBuilder
+ build1(): void
+ build2(): void
+ build3(): void
+ build4(): void
motorBuilder
+ build1(): void
+ build2(): void
+ build3(): void
+ build4(): void
truckBuilder
+ build1(): void
+ build2(): void
+ build3(): void
+ build4(): void

```java
public  interface carBuilder {
    1 usage  2 implementations
    public void build1();
    1 usage  2 implementations
    public void build2();
    1 usage  2 implementations
    public void build3();
    1 usage  2 implementations
    public void build4();
}
public class client {
    public static void main(String[] args) {
        carBuilder carBuilder = new motorCarBuilder();
        carBuilder carBuilder1 = new truckBuilder();

        director director = new director();

        director.ConstructCar(carBuilder);
        director.ConstructCar(carBuilder1);
    }
}
public class director {

    2 usages
    public void ConstructCar(carBuilder carbuilder)
    {
        carbuilder.build1();
        carbuilder.build2();
        carbuilder.build3();
        carbuilder.build4();
    }
}

public class motorCarBuilder implements carBuilder{
    1 usage
    @Override
    public void build1() {
        System.out.println("mo 1");
    }

    1 usage
    @Override
    public void build2() {
        System.out.println("mo 2");
    }

    1 usage
    @Override
```

```
    @Override
    public void build3() {
        System.out.println("mo 3");
    }

    1 usage
    @Override
    public void build4() {
        System.out.println("mo 4");
    }
}

public class truckBuilder implements carBuilder{
    1 usage
    @Override
    public void build1() {
        System.out.println("tr 1");
    }

    1 usage
    @Override
    public void build2() {
        System.out.println("tr 2");
    }

    1 usage
    @Override
    public void build3() {
        System.out.println("tr 3");
    }

    1 usage
    @Override
    public void build4() {
        System.out.println("tr 4");
    }
}
```

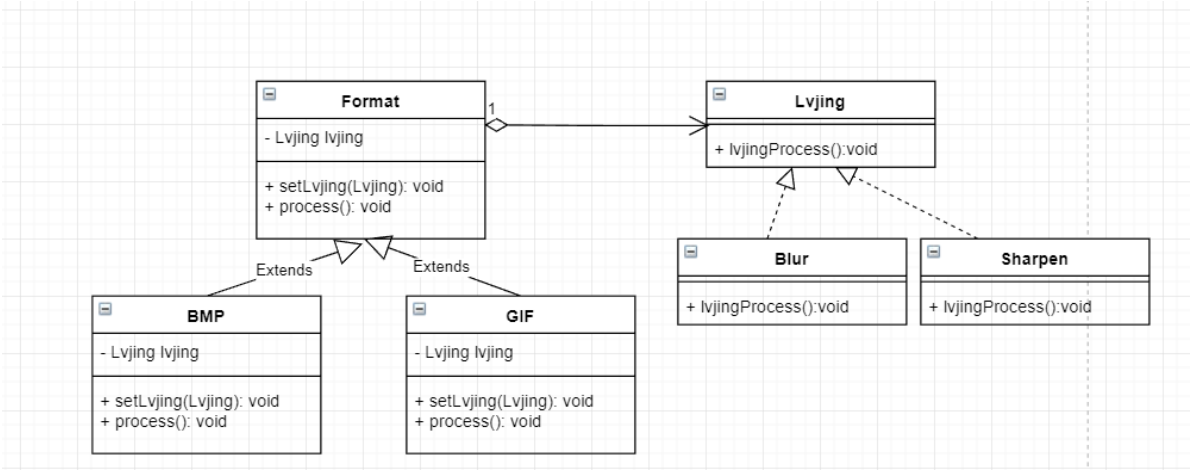
3.




```java
public interface DBUti {
    2 implementations
    public IConnection getConnection();
    2 usages  2 implementations
    public IStatement useStatement();
}
public interface IConnection {
    2 usages  2 implementations
    public String connectionName();
}
public interface IStatement {
    2 usages  2 implementations
    public String statementName();
}

public class MysqlConnection implements IConnection{
    2 usages
    @Override
    public String connectionName() {
        return "connection to mysql";
    }
}
public class MysqlStatement implements IStatement{
    2 usages
    @Override
    public String statementName() {
        return "state to mysql";
    }
}
```

```
public class MysqlUtil implements DBUti{
    @Override
    public IConnection getConnection() {
        return new MysqlConnection();
    }

    2 usages
    @Override
    public IStatement useStatement() { return new MysqlStatement(); }
}
 public class OracleConnection implements IConnection{
     2 usages
     @Override
     public String connectionName() {
         return "connection to oracle";
     }
 }

public class OracleStatement implements IStatement{
    2 usages
    @Override
    public String statementName() {
        return "state to oracle";
    }
}
public class OracleUtil implements DBUti{
    @Override
    public IConnection getConnection() {
        return new OracleConnection();
    }

    2 usages
    @Override
    public IStatement useStatement() { return new OracleStatement(); }
}
```

4.

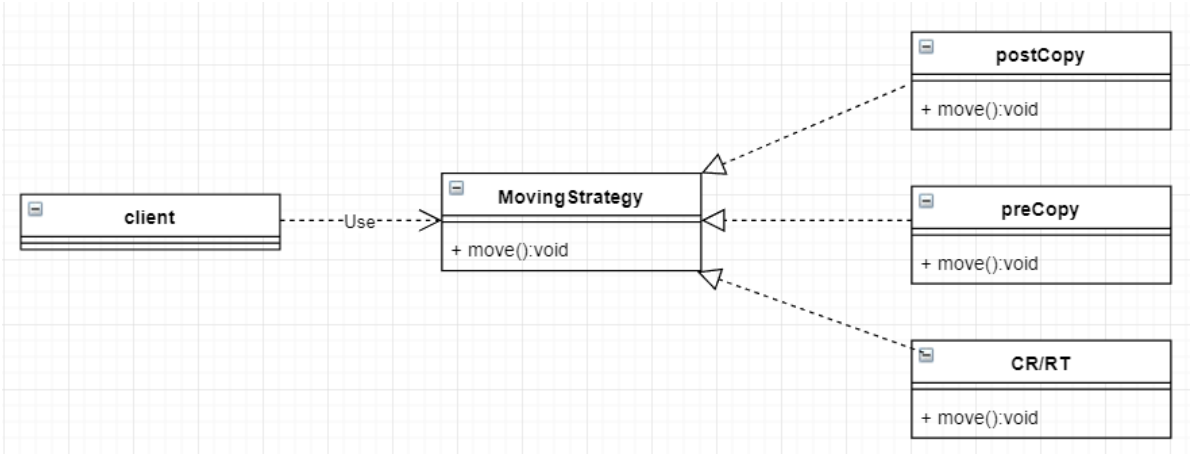



```java
public abstract class Format {
    4 usages
    lvjing lvjing;

    4 usages
    public void setLvjing(lvjing lvjing) { this.lvjing = lvjing; }

    4 usages  2 overrides
    public void process() { this.lvjing.process(); }
}
public interface lvjing {
    3 usages  2 implementations
    public void process();
}
```

```java
public class blur implements lvjing{
    3 usages
    @Override
    public void process() {
        System.out.println("blur");

    }
}

public class BMP extends Format{

    4 usages
    @Override
    public void process()
    {
        this.lvjing.process();
        System.out.println("bmp");
    }
}
public class client {
    public static void main(String[] args) {
        BMP bmp = new BMP();
        bmp.setLvjing(new blur());
        bmp.process();

        bmp.setLvjing(new Sharpen());
        bmp.process();

        GIF gif = new GIF();
        gif.setLvjing(new blur());
        gif.process();

        gif.setLvjing(new Sharpen());
        gif.process();

    }}

public class GIF extends Format{
    4 usages
    @Override
    public void process()
    {
        this.lvjing.process();
        System.out.println("gif");
    }
}
```

```java
public class Sharpen implements lvjing{
    3 usages
    @Override
    public void process() {
        System.out.println("sharpen");
    }
}
```

5.




```java
public interface movingStrategy {

    3 implementations
    public String getName();

}
public class postcopy implements movingStrategy{
    @Override
    public String getName() {
        return "postcopy";
    }
}
public class preCopy implements movingStrategy{
    @Override
    public String getName() {
        return "precopy";
    }
}
public class crrtmotion implements movingStrategy{
    @Override
    public String getName() {
        return "crrt";
    }
}
```

实验小结

使用五种设计模式分别对于不同的问题进行建模，对于能够使用的场景进行一定小结：

简单工厂模式：

1. 客户只知道创建产品的工厂名，而不知道具体的产品名
2. 创建对象得到任务有多个具体子工厂中的某一个完成，而抽象工厂只提供创建产品的接口
3. 客户不关心创建产品的细节，只关心品牌

建造者模式：

1. 创建的对象较复杂，由多个部件构成，各部件面临复杂的变化，但构件间的建造方式是稳定的
2. 产品的构建过程和最终的表示是独立的

抽象工厂模式：

1. 创建的对象是一系列相互关联或相互依赖的产品族（电器工厂中电视机，洗衣机）
2. 系统中有多个产品族，但每次只使用某一族产品
3. 系统中提供了产品的类库，且所有产品的接口相同，客户端不依赖产品实例的创建细节和内部结构

桥接模式：

将抽象和实现分离，使它们可以独立变化，聚合关系建立在抽象层

策略模式：

将定义的每个算法封装起来，使他们可以相互替换，且算法的变换不影响用户的使用

设计模式最终的目的就是对于一段代码所具备功能的角色划分，通过将功能划分为不同的角色达到解耦的目的，同时尽量满足面向对象的设计原则，对系统做修改和扩展的时候，能够将对系统其他模块的影响降到最低。