# 策略模式

消除程序中大量的if...else...和switch语句

定义一系列的算法,把每一个算法封装起来, 并且使它们可相互替换。

应用场景：

1. 许多相关的类仅仅是行为有异。 “策略”提供了一种用多个行为中的一个行为来配置一个类的方法。即一个系统需要动态地在几种算法中选择一种。
2. 一个类定义了多种行为 , 并且这些行为在这个类的操作中以多个条件语句的形式出现。将相关的条件分支移入它们各自的Strategy类中以代替这些条件语句

例子：上班的出行方式，可以走路、自行车、电动车、公交、地铁...

去公司接口：

```
package com.dml.pattern.strategy;

public interface GoCompany {
    void toCompany();
}
```

公交方式去公司：

```
package com.dml.pattern.strategy;

public class Bus implements GoCompany {
    @Override
    public void toCompany() {
        System.out.println("乘坐公交到公司");
    }
}
```

地铁：

```
package com.dml.pattern.strategy;

public class Metro implements GoCompany {
    @Override
    public void toCompany() {
        System.out.println("乘坐地铁到公司");
    }
}
```

步行：

```
package com.dml.pattern.strategy;

public class Walk implements GoCompany {
    @Override
    public void toCompany() {
        System.out.println("步行到公司");
    }
}
```

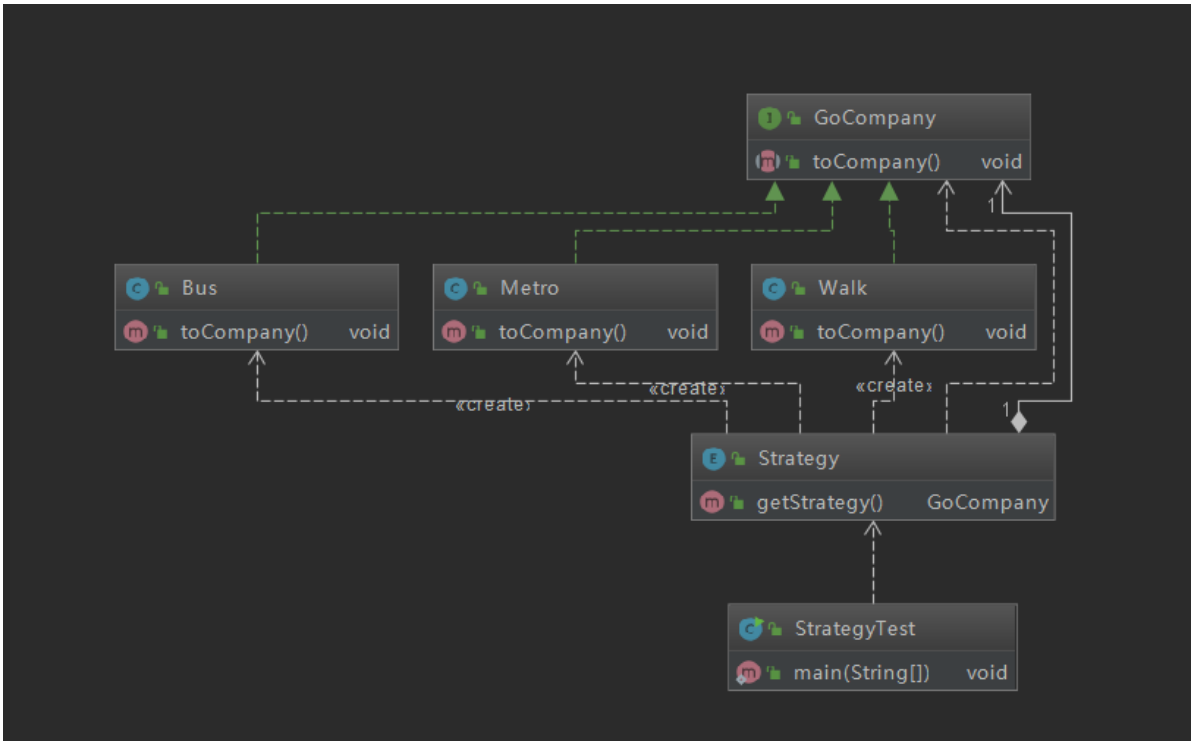
策略枚举：

```
package com.dml.pattern.strategy;

public enum Strategy {
    BUS(new Bus()),
    WALK(new Walk()),
    METRO(new Metro())
    ;
    private GoCompany strategy;
    Strategy(GoCompany strategy) {
        this.strategy = strategy;
    }

    public GoCompany getStrategy() {
        return strategy;
    }
}
```

测试：

```
package com.dml.pattern.strategy;

public class StrategyTest {
    public static void main(String[] args) {
        Strategy.BUS.getStrategy().toCompany();
    }
}
```

类图


GoCompany
toCompany() void
Bus
toCompany() void
Metro
toCompany() void
Walk
toCompany() void
«create»
«create»
«create»
1
1
Strategy
getStrategy() GoCompany
StrategyTest
main(String[]) void