简述

概念：抽象工厂模式（Abstract Factory Pattern）是围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂

比如说生产鼠标和键盘

- 对于工厂模式来说，就有生产鼠标的工厂和生产键盘的工厂。生产鼠标的工厂既可以生产罗技的鼠标也可以生产微软的鼠标，生产键
- 对于抽象工厂来说，生产鼠标和生产键盘就一定是同一个品牌的

总的来说：需要鼠标和键盘，如果用工厂模式，那么可以选用不同品牌的鼠标键盘组合。而选用工厂模式就一个是同一品牌的鼠标键盘套装

代码实现

```
public abstract class AbsFactory {

    public abstract void getMouseAndBoard();

    public static AbsFactory getInstance(String type){
        AbsFactory factory = null;
        if ("hp".equals(type)){
            factory = new HpFactory();
        }else if ("luoji".equals(type)){
             factory = new LuojiFactory();
         }
         return factory;
    }
}

public class HpFactory extends AbsFactory {

    private final String type = "hp";

    @Override
    public void getMouseAndBoard() {
        Mouse mouse = new Mouse(this.type);
        KeyBoard keyBoard = new KeyBoard(this.type);
        System.out.println("鼠标: " + mouse.getType() + " : 键盘: " + keyBoard.getType());
    }
}


public class LuojiFactory extends AbsFactory {

    private final String type = "luoji";

    @Override
```

```
    public void getMouseAndBoard() {
        Mouse mouse = new Mouse(this.type);
        KeyBoard keyBoard = new KeyBoard(this.type);
        System.out.println("鼠标: " + mouse.getType() + " : 键盘: " + keyBoard.getType());
    }
}



@Getter
@Setter
public class Mouse {
    private String type;

    public Mouse(String type) {
        this.type = type;
    }
}

@Getter
@Setter
public class KeyBoard {
    private String type;

    public KeyBoard(String type) {
        this.type = type;
    }
}

public class Main {
    public static void main(String[] args) {
        AbsFactory factory = AbsFactory.getInstance("hp");
        factory.getMouseAndBoard();
    }
}
```

总结

工厂模式：是某一个具体的产品生产者
抽象工厂模式：是工厂模式的组织者，生产出来的同一系列的产品，品牌机
建造者模式：也可以将工厂组织起来，但是使用的工厂不一定是同一个品牌，组装机

如果生产的产品比较多：
用工厂模式会需要非常多的工厂，调用者就会比较繁琐。
用抽象工厂，调用者一步到位，同一个品牌，但是产品线扩充，那么扩展比较麻烦
建造者模式，调用者需要进行筛选，自己组装，相对也比较繁琐。但是灵活一些