实验 2 Java 高级特性

一、实验目的

1、了解泛型的基本概念以及为什么要使用泛型；
2、了解注解的基本概念及使用方法以；
3、了解 Java 典型的高级特性。

二、实验内容

1、反射

(1) 利用 Object 类中提供的 getClass()方法获取实例化对象

```
class Member {}
public class JavaReflectDemo {
    public static void main(String[] args) throws Exception {
        // 【操作特点】需要获取一个类的实例化对象之后才可以获取Class类实例
        Member member = new Member() ;                    // 实例化Member类对象
        Class<?> clazz = member.getClass() ;              // 获取Class类实例化对象
        System.out.println(clazz);
    }
}
```

(2) 使用“类.clazz”形式获取指定类或接口的 Class 实例化对象

```
class Member {}
public class JavaReflectDemo {
    public static void main(String[] args) throws Exception {
    //直接通过一个类的完整名称可以获取Class类实例，需要编写import或编写完整类名称
        Class<?> clazz = Member.class ;                // 获取Class类实例化对象
        System.out.println(clazz);
    }
}
```

(3) 使用 Class 类内部提供的 forName()方法根据类的完整名称获取实例化对象

```
class Member {}
public class JavaReflectDemo {
    public static void main(String[] args) throws Exception {
        // 【操作特点】通过名称字符串（包.类）可以获取Class类实例，可以不使用import导入
        Class<?> clazz = Class.forName("cn.mldn.demo.Member") ;//获取Class类实例化对
象
        System.out.println(clazz);
    }
```

2、泛型

(1) 使用“?”接收数据

```
class Message<T> {                                          // 定义泛型类对象
    private T content;                                      // 泛型属性
    public void setContent(T content) {
        this.content = content;
    }
    public T getContent() {
        return this.content;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Message<String> msg = new Message<String>() ;       // 实例化Message类对象
        msg.setContent("www.mldn.cn") ;
        fun(msg) ;                                          // 引用传递
    }
    public static void fun(Message<?> temp){        // 输出信息， 只允许取出不允许修改
    // 如果现在需要接收则会使用Object作为泛型类型，即：String str = (String)
temp.getContent() ;
        System.out.println(temp.getContent()) ;             // 获取数据
    }
}
```

(2) 定义泛型接口子类，在子类中继续声明泛型

```
interface IMessage<T> {                                 // 泛型接口
    public String echo(T msg) ;                         // 抽象方法
}
class MessageImpl<S> implements IMessage<S> {           // 子类继续声明泛型类型
    public String echo(S t) {                           // 方法覆写
        return "【ECHO】" + t ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        // 实例化泛型接口对象，同时设置泛型类型
        IMessage<String> msg = new MessageImpl<String>();
        System.out.println(msg.echo("www.mldn.cn"));    // 调用方法
    }
}
```

(3) 定义泛型方法

```
public class JavaDemo {
```

```
    public static void main(String args[]) {
        Integer num[] = fun(1, 2, 3);              // 传入了整数，泛型类型就是Integer
        for (int temp : num) {                     // foreach输出
            System.out.print(temp + "、");         // 输出数据
        }
    }
    // 定义泛型方法，由于类中没有设置泛型，所以需要定义一个泛型标记，泛型的类型就是传递的参数类型
    public static <T> T[] fun(T... args) {         // 可变参数
        return args;                               // 返回数组
    }
}
```

3、枚举

```
interface IMessage {
    public String getMessage();                    // 获取信息
}
enum Color implements IMessage {                   // 枚举类实现接口
    RED("红色"), GREEN("绿色"), BLUE("蓝色");       // 枚举对象要写在首行
    private String title;                          // 成员属性
    private Color(String title) {                  // 构造方法初始化属性
        this.title = title;
    }
    @Override
    public String toString() {                     // 输出对象信息
        return this.title;
    }
    @Override
    public String getMessage() {                   // 方法覆写
        return this.title ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IMessage msg = Color.RED ;                 // 对象向上转型
        System.out.println(msg.getMessage());
    }
}
```

4、接口新定义

```java
interface IMessage {                                    // 定义接口
    public String message() ;                           // 【抽象方法】获取信息
    public default boolean connect() {                  // 公共方法被所有子类继承
        System.out.println("建立MLDN订阅消息连接通道。") ;
        return true ;
    }
    public static IMessage getInstance() {              // 定义static方法，可以通过接
口名称调用
        return new MessageImpl() ;                      // 获得子类对象
    }
}
class MessageImpl implements IMessage {                 // 定义接口子类
    public String message() {                           // 覆写抽象方法
        if (this.connect()) {
            return "www.mldn.cn" ;
        }
        return "没有消息发送。" ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IMessage msg = IMessage.getInstance() ;         // 实例化接口子类对象
        System.out.println(msg.message()) ;             // 调用方法
    }
}
```

5、方法引用

```java
@FunctionalInterface                                    // 函数式接口
interface IFunction<P, R> {                             // P描述的是参数、R描述的是返回值
    public R change(P p);                               // 随意定义一个方法名称，进行方法引用
}
public class JavaDemo {
    public static void main(String args[]) {
        // 引用String类中所提供的一个静态方法
        IFunction<Integer, String> fun = String::valueOf;
        String str = fun.change(100);                   // 利用change()表示valueOf()
        System.out.println(str.length());               // 调用String类方法
    }
}
```

6、Lambda 表达式

```
interface IMessage {                                    // 定义接口
    public void send(String str);                       // 抽象方法
}
public class JavaDemo {
    public static void main(String args[]) {
        IMessage msg = (str) -> {                       // Lambda等价于匿名内部类
            System.out.println("发送消息：" + str);      // 方法体
        };
        msg.send("www.yootk.com");                      // 调用接口方法
    }
}
```

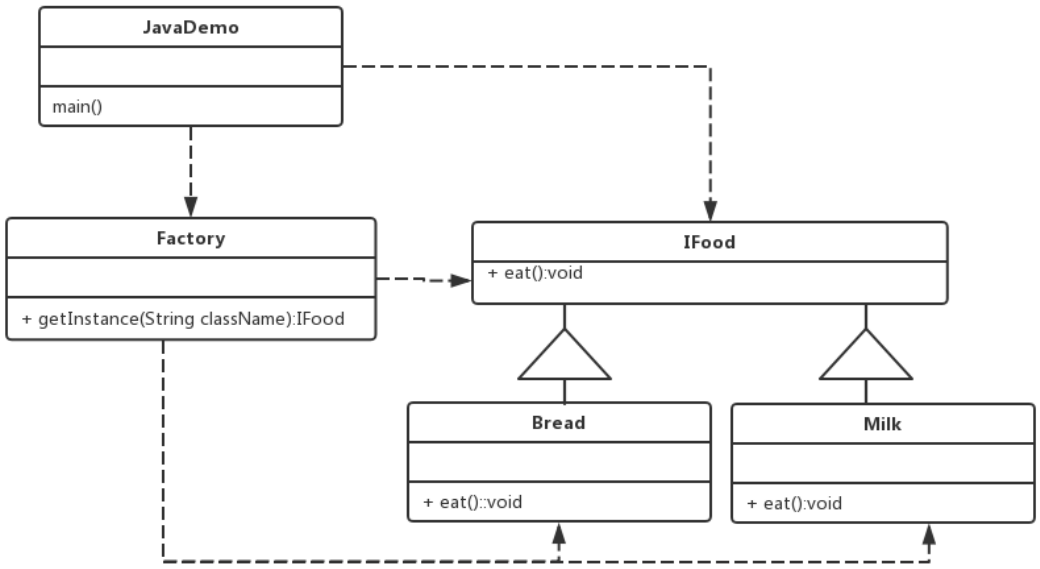
三、根据下面的要求编写程序（三选一）

1、工厂模式：项目开发中需要考虑实例化对象的解耦和问题，工厂模式隐藏接口对象实例化操作细节。简单案例如下：

接口定义：

```
interface IFood {                                       // 定义食物标准
    public void eat();                                  // 食物的核心功能：吃
}
```

类图如下：

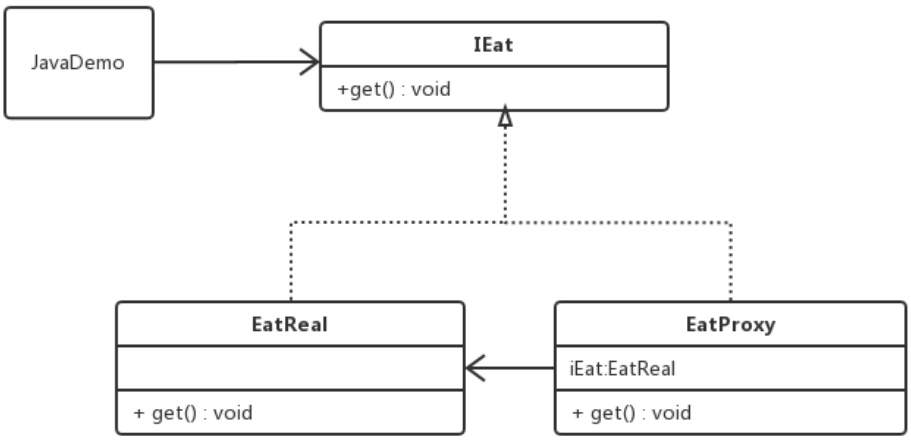


请实现Bread、Milk、Factory及测试主类。

2、代理模式：指用一个代理主题来操作真实主题，真实主题执行具体的业务操作，而代理主题负责其他相关业务的处理。例如：一个人肚子饿了要吃饭，但不会做饭，只能去饭店吃饭，饭店完成做饭的操作。接口定义如下：

```
interface IEat {                                    // 定义核心业务标准
  public void get();                                // 业务方法
}
```

具体类图如下：



请实现各个类。

3、ORMaping 是面向对象程序设计中典型的思想，实际就是把 Java 的实体类与数据库的实体进行映射。请根据 Java 的实体类采用注解的方法，动态生成 Create SQL 语句，请参考课本 4.5。

四、实验结果

写实验报告。内容包括：

1. 习题的运行结果，源程序。
2. 程序调试中出现的错误提示。（英文、中文对照）
3. 若有没通过的程序，分析原因。