# 设计模式实验（2）

## 一、实验目的

1. 结合实例，熟练绘制设计模式结构图。
2. 结合实例，熟练使用 Java 语言实现设计模式。
3. 通过本实验，理解每一种设计模式的模式动机，掌握模式结构，学习如何使用代码实现这些设计模式。

## 二、实验要求

1. 结合实例，绘制设计模式的结构图。
2. 使用 Java 语言实现设计模式实例，代码运行正确。

## 三、实验内容

### 1. 迭代器模式

设计一个逐页迭代器，每次可返回指定个数（一页）元素，并将该迭代器用于对数据进行分页处理。绘制对应的类图并编程模拟实现。

### 2. 适配器模式

某 OA 系统需要提供一个加密模块，将用户机密信息（例如口令、邮箱等）加密之后再存储在数据库中，系统已经定义好了数据库操作类。为了提高开发效率，现需要重用已有的加密算法，这些算法封装在一些由第三方提供的类中，有些甚至没有源代码。试使用适配器模式设计该加密模块，实现在不修改现有类的基础上重用第三方加密方法。要求绘制相应的类图并编程模拟实现，需要提供对象适配器和类适配器两套实现方案。

### 3. 模板方式模式和适配器模式

在某数据挖掘工具的数据分类模块中，数据处理流程包括 4 个步骤，分别是：①读取数据；②转换数据格式；③调用数据分类算法；④显示数据分类结果。对于不同的分类算法而言，第①步、第②步和第④步是相同的，主要区别在于第③③步。第③③步将调用算法库中已有的分类算法实现，例如朴素贝叶斯分类（Naive Bayes）算法、决策树（Decision Tree）算法、 K 最近邻（K - Nearest Neighbor , KNN）算法等。现采用模板方法模式和适配器模式设计该数据分类模块，绘制对应的类图并编程模拟实现。

### 4. 工厂方法模式

在某网络管理软件中，需要为不同的网络协议提供不同的连接类，例如针对 POP3 协议的连接类 POP3Connection、针对 IMAP 协议的连接类 IMAPConnection 、针对 HTTP 协议的连接类 HTTPConnection 等。由于网络连接对象的创建过程较为复杂，需要将其创建过程封装到专门的类中，该软件还将支持更多类型的网络协议。现采用工厂方法模式进行设计，绘制类图并编程模拟实现。

### 5. 单例模式

某 Web 性能测试软件中包含一个虚拟用户生成器（Virtual User Generator）。为了避免生成的虚拟用户数量不一致，该测试软件在工作时只允许启动唯一一个虚拟用户生成器。采用单例模式设计该虚拟用户生成器，绘制类图并分别使用饿汉式单例、双重检测锁和 IoDH三种方式编程模拟实现。

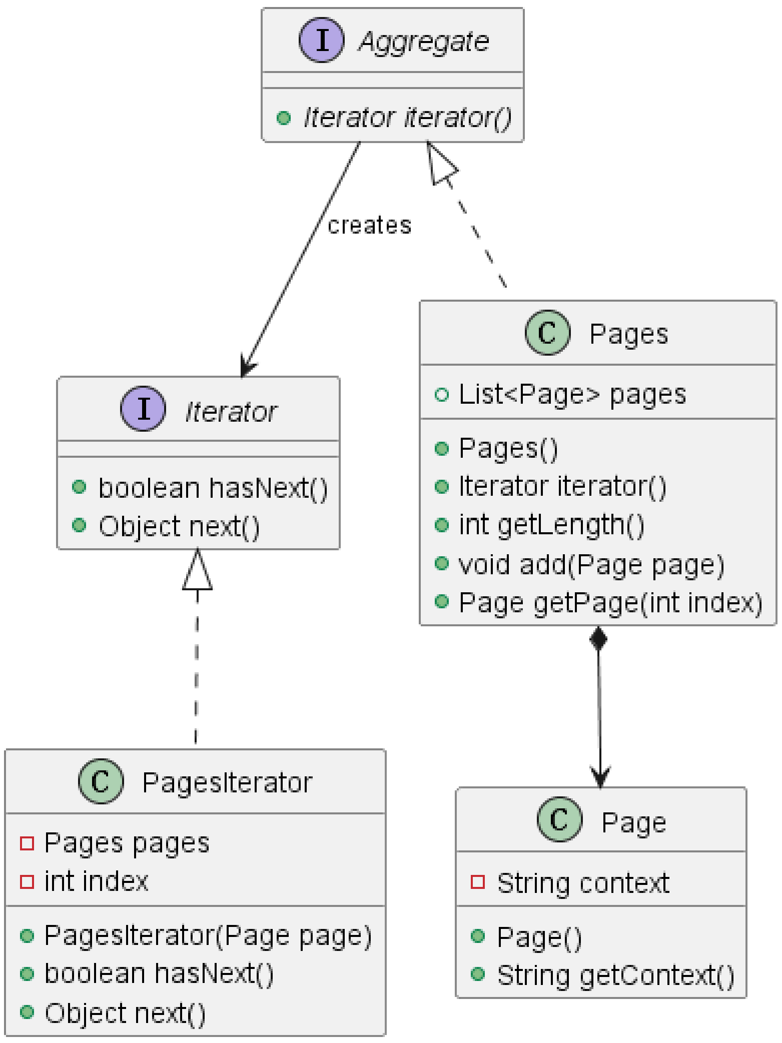
### 6. 原型模式

在某在线招聘网站中，用户可以创建一个简历模板。针对不同的工作岗位，可以复制该简历模板并进行适当修改后，生成一份新的简历。在复制简历时，用户可以选择是否复制简历中的照片：如果选择“是”，则照片将一同被复制，用户对新简历中的照片进行修改不会影响到简历模板中的照片，对模板进行修改也不会影响到新简历；如果选择“否”，则直接引用简历模板中的照片，修改简历模板中的照片将导致新简历中的照片一同修改，反之亦然。现采用原型模式设计该简历复制功能并提供浅克隆和深克隆两套实现方案，绘制对应的类图并编程模拟实现。

## 三、实验结果

### 1. 迭代器模式

#### 类图



## 实现代码

- Aggregate.java

```
public interface Aggregate {
    public abstract Iterator iterator();
}
```

- Iterator.java

```
public interface Iterator {
    public boolean hasNext();
    public Object next();
}
```

- Page.java

```
public class Page {
    private String content;
    public Page(String content){
        this.content=content;
    }

    public String getContent() {
        return content;
    }
}
```

- Pages.java

```
import java.util.ArrayList;
import java.util.List;

public class Pages implements Aggregate{
    public List<Page> pages;

    public Pages() {
        // TODO Auto-generated constructor stub
        pages = new ArrayList<Page>();
    }

    @Override
    public Iterator iterator() {
        // TODO Auto-generated method stub
        return new PagesIterator(this);
    }

    public int getLength() {
        return pages.size();
    }

    public void add(Page page) {
        pages.add(page);
    }

    public Page getPage(int index) {
        return pages.get(index);
    }
}
```

- PagesIterator.java

```

public class PagesIterator implements Iterator{
    private Pages pages;
    private int index;

    public PagesIterator(Pages pages) {
        // TODO Auto-generated constructor stub
        this.pages=pages;
        this.index=0;
    }

    @Override
    public boolean hasNext() {
        // TODO Auto-generated method stub
        return this.index<this.pages.getLength();
    }

    @Override
    public Object next() {
        // TODO Auto-generated method stub
        Page page = pages.getPage(index);
        index++;
        return page;
    }

}
```

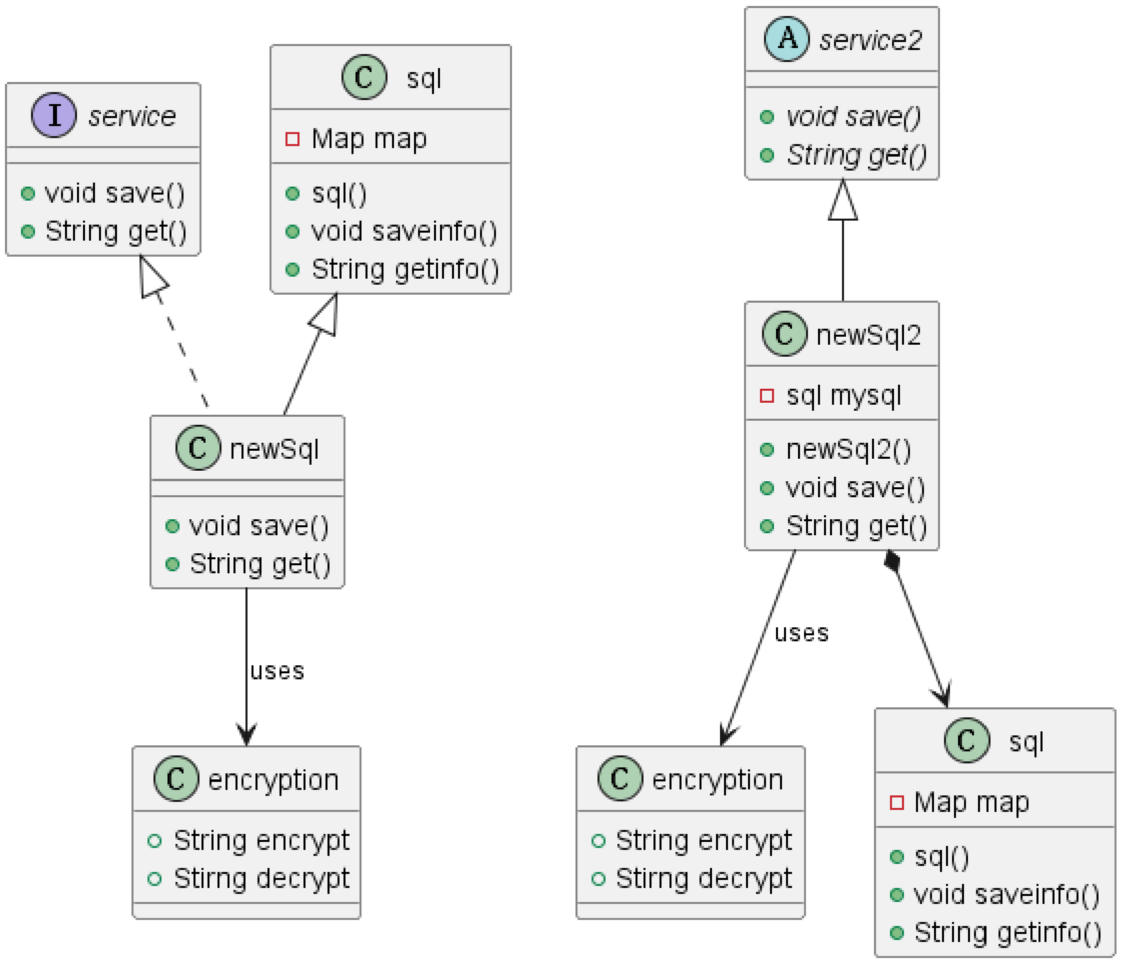
- Test.java

```

public class Test {
    public static void main(String[] args) {
        Pages pages = new Pages();
        pages.add(new Page("page 1"));
        pages.add(new Page("page 2"));
        pages.add(new Page("page 3"));
        pages.add(new Page("page 4"));
        Iterator it = pages.iterator();
        while(it.hasNext()){
            Page page= (Page)it.next();
            System.out.println(page.getContent());
        }
    }
}
```

-

## 2. 适配器模式

### 类图



### 实现代码

- encryption.java

```java
public class encryption {
    public String encrypt(String info) {
        return "#"+info+"#";
    }

    public String decrypt(String info) {
        return info.replace("#", "");
    }
}
```

- newSql.java

```java
public class newSql extends sql implements service {

    @Override
    public void save(int id, String info) {
        // TODO Auto-generated method stub
        encryption encrypt = new encryption();
        info = encrypt.encrypt(info);
        this.saveInfo(id, info);
    }

    @Override
    public String get(int id) {
        // TODO Auto-generated method stub
        String info = this.getInfo(id);
        encryption encrypt = new encryption();
        info = encrypt.decrypt(info);
        return info;
    }

}
```

- newSql2.java

```

public class newSql2 extends service2{
    private sql mysql;

    public newSql2(sql cursql) {
        // TODO Auto-generated constructor stub
        mysql = cursql;
    }

    @Override
    public void save(int id, String info) {
        // TODO Auto-generated method stub
        encryption encrypt = new encryption();
        encrypt.encrypt(info);
        mysql.saveInfo(id, info);
    }

    @Override
    public String get(int id) {
        // TODO Auto-generated method stub
        String info = mysql.getInfo(id);
        encryption encrypt = new encryption();
        info = encrypt.decrypt(info);
        return info;
    }

}
```

- service.java

```

public interface service {
    public void save(int id,String info);
    public String get(int id);
}
```

- service2.java

```java

public abstract class service2 {
    public abstract void save(int id,String info);
    public abstract String get(int id);
}
```

- sql.java

```java
import java.util.HashMap;
import java.util.Map;

public class sql{
    private Map<Integer,String> map;
    public sql() {
        // TODO Auto-generated constructor stub
        map = new HashMap<Integer,String>();
    }

    public void saveInfo(int id,String info) {;
        map.put(id, info);
    }

    public String getInfo(int id) {
        return map.get(id);
    }
}
```

- Test2_1.java

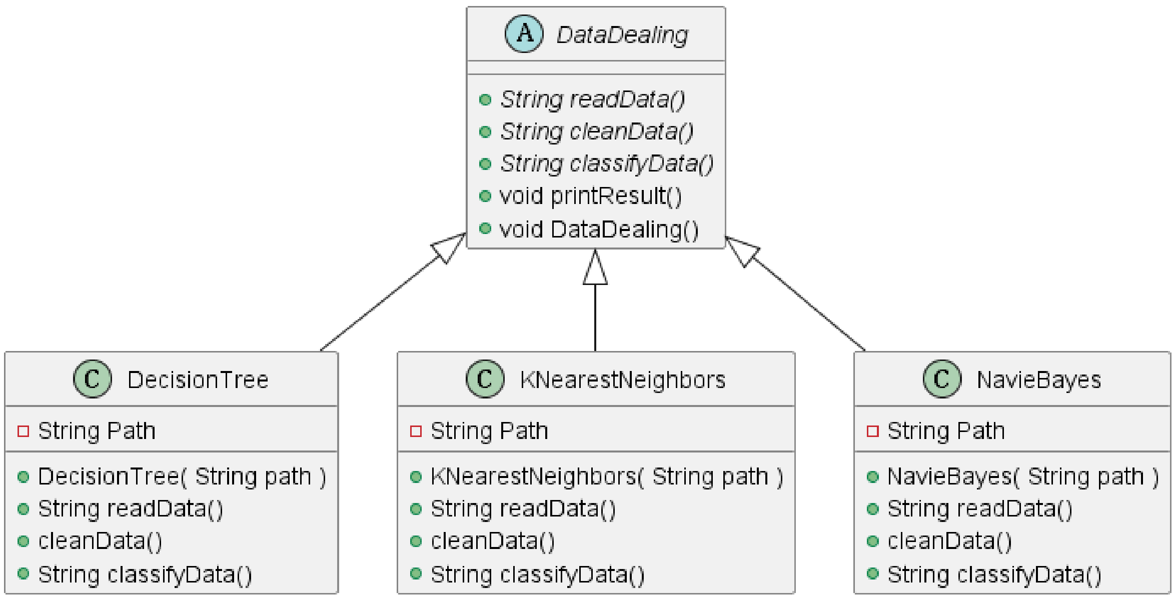
```java

public class Test2_1 {
    public static void main(String[] args) {
        service s = new newSql();
        s.save(1, "abc");
        System.out.print(s.get(1));
    }
}
```

- Test2_2.java

```java

public class Test2_2 {
    public static void main(String[] args) {
        service2 s = new newSql2(new sql());
        s.save(1, "abc");
        System.out.print(s.get(1));
    }
}
```

## 3. 模板方式模式和适配器模式

### 类图



### 实现代码

- DataDealing.java

```java
public abstract class DataDealing {
    public abstract String readData();

    public abstract String cleanData(String rawData);

    public abstract String classifyData(String cleanedData);

    public final void printResult(String result){
        System.out.println(result);
    }

    public final void dataDealing(){
        String rawData = readData();
        String cleandData = cleanData(rawData);
        String result = classifyData(cleandData);
        printResult(result);
    }

}
```

- DecisionTree.java

```java
public class DecisionTree extends DataDealing{
    private String path;

    public DecisionTree(String path){
        this.path = path;
    }

    @Override
    public String readData() {
        return "data from :"+this.path;
    }

    @Override
    public String cleanData(String rawData) {
        return rawData;
    }

    @Override
    public String classifyData(String cleanedData) {
        return "classified by " + this.getClass().getName() + "\n" + cleanedData
+"\n";
    }

}
```

- KNearestNeighbour.java

```java
public class KNearestNeighbour extends DataDealing {
    private String path;

    public KNearestNeighbour(String path){
        this.path = path;
    }

    @Override
    public String readData() {
        return "data from :"+this.path;
    }

    @Override
    public String cleanData(String rawData) {
        return rawData;
    }

    @Override
    public String classifyData(String cleanedData) {
        return "classified by " + this.getClass().getName() + "\n" + cleanedData
+"\n";
    }
}
```

- NavieBayes.java

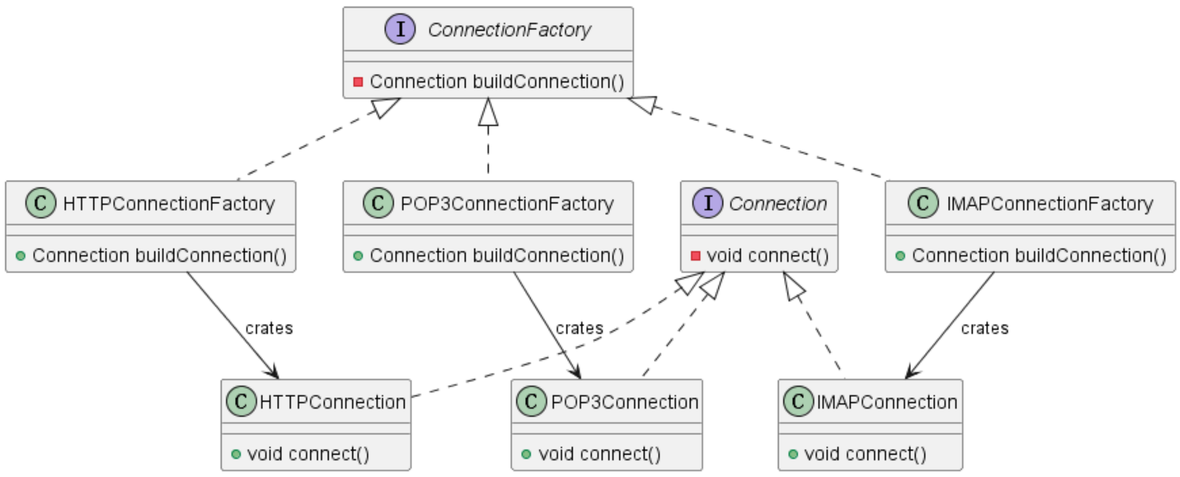
```java
public class NavieBayes extends DataDealing{
    private String path;
    public NavieBayes(String path){
        this.path=path;
    }

    @Override
    public String readData() {
        return "data from :"+this.path;
    }

    @Override
    public String cleanData(String rawData) {
        return rawData;
    }

    @Override
    public String classifyData(String cleanedData) {
        return "classified by " + this.getClass().getName() + "\n" + cleanedData
+"\n";
    }

}
```

- Test3_1.java

```
public class Test3_1 {
    public static void main(String[] args) {
        DataDealing d1 = new NavieBayes("Test1");
        DataDealing d2 = new DecisionTree("Test2");
        DataDealing d3 = new KNearestNeighbour("Test3");
        d1.dataDealing();
        d2.dataDealing();
        d3.dataDealing();
    }
}
```

## 4. 工厂方法模式

### 类图



### 实现代码

- Connection.java

```
public interface Connection {
    void connect();
}
```

- ConnectionFactory.java

```java
public interface ConnectionFactory {
    Connection buildConnection();
}
```

- HTTPConnection.java

```java
public class HTTPConnection implements Connection{
    @Override
    public void connect() {
        System.out.println("use "+this.getClass().getName());
    }
}
```

- HTTPConnectionFactory.java

```java
public class HTTPConnectionFactory implements ConnectionFactory {

    @Override
    public Connection buildConnection() {
        return new HTTPConnection();
    }
}
```

- IMAPConnection.java

```java
public class IMAPConnection implements Connection{
    @Override
    public void connect() {
        System.out.println("use "+this.getClass().getName());
    }
}
```

- IMAPConnectionFactory.java

```java
public class IMAPConnectionFactory implements ConnectionFactory {

    @Override
    public Connection buildConnection() {
        return new IMAPConnection();
    }
}
```

- POP3Connection.java

```java
public class POP3Connection implements Connection {
    @Override
    public void connect() {
        System.out.println("use "+this.getClass().getName());
    }
}
```

- POP3ConnectionFactory.java

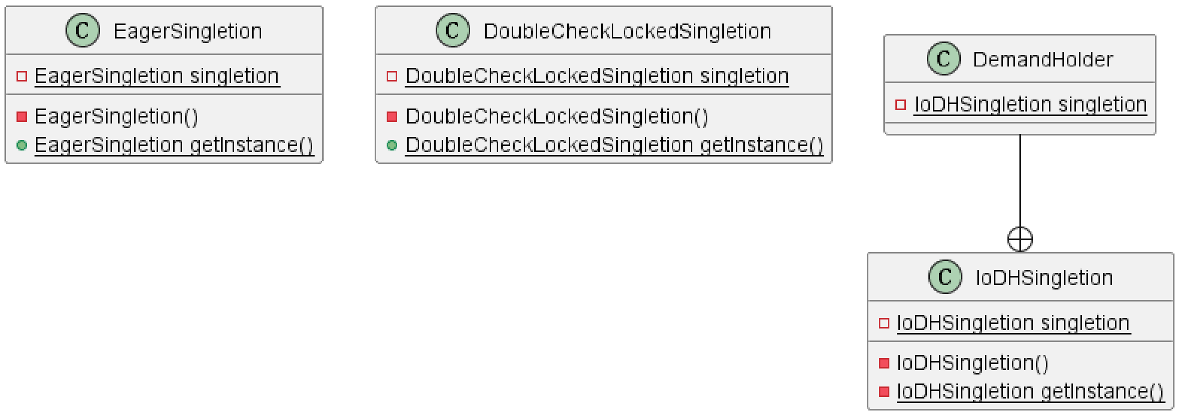
```
public class POP3ConnectionFactory implements ConnectionFactory {
    @Override
    public Connection buildConnection() {
        return new POP3Connection();
    }
}
```

- Test4.java

```
public class Test4 {
    public static void main(String[] args) {
        ConnectionFactory pop3ConnectionFactory = new POP3ConnectionFactory();
        ConnectionFactory imapConnectionFactory = new IMAPConnectionFactory();
        ConnectionFactory httpConnectionFactory = new HTTPConnectionFactory();
        Connection pop3Connection = pop3ConnectionFactory.buildConnection();
        Connection imapConnection = imapConnectionFactory.buildConnection();
        Connection httpConnection = httpConnectionFactory.buildConnection();
        pop3Connection.connect();
        imapConnection.connect();
        httpConnection.connect();
    }
}
```

## 5. 单例模式

### 类图


C EagerSingletion
EagerSingletion singletion
EagerSingletion()
EagerSingletion getInstance()
C DoubleCheckLockedSingletion
DoubleCheckLockedSingletion singletion
DoubleCheckLockedSingletion()
DoubleCheckLockedSingletion getInstance()
C DemandHolder
IoDHSingletion singletion
C IoDHSingletion
IoDHSingletion singletion
IoDHSingletion()
IoDHSingletion getInstance()

### 实现代码

- DoubleCheckLockingSingletion.java

```
public class DoubleCheckLockingSingletion {
    private static DoubleCheckLockingSingletion singletion = null;
    private DoubleCheckLockingSingletion(){
        System.out.println("create singletion by "+this.getClass().getName());
    }
    public static DoubleCheckLockingSingletion getInstance(){
        if(singletion==null){
            synchronized (DoubleCheckLockingSingletion.class){
                if(singletion==null){
                    singletion = new DoubleCheckLockingSingletion();
                }
            }
        }
        return singletion;
    }
}
```

- EagerSingletion.java

```
public class EagerSingletion {
    private static final EagerSingletion singletion = new EagerSingletion();
    private EagerSingletion(){
        System.out.println("create singletion by "+this.getClass().getName());
    }
    public static  EagerSingletion getInstance(){
        return singletion;
    }
}
```

- IoDHSingletion.java

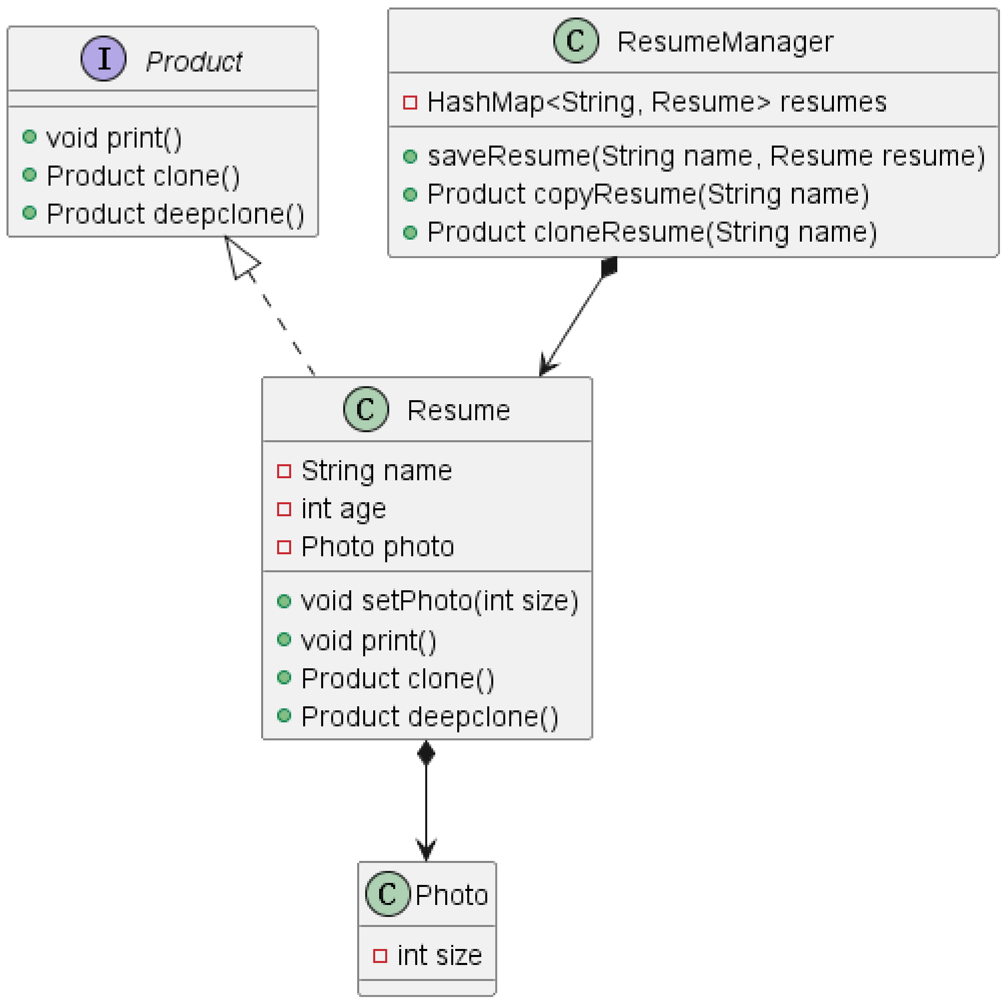
```
public class IoDHSingletion {
    //Initioalization on Demand Holder
    private static IoDHSingletion singletion  = null;
    private IoDHSingletion(){
        System.out.println("create singletion by "+this.getClass().getName());
    }
    private static class DemandHolder{
        private static IoDHSingletion singletion = new IoDHSingletion();
    }

    public static IoDHSingletion getInstance(){
        return DemandHolder.singletion;
    }

}
```

- Test5.java

```
public class Test5 {
    public static void main(String[] args) {
        EagerSingletion eagerSingletion = EagerSingletion.getInstance();
        DoubleCheckLockingSingletion doubleCheckLockingSingletion =
DoubleCheckLockingSingletion.getInstance();
        IoDHSingletion ioDHSingletion = IoDHSingletion.getInstance();
    }
}
```

## 6. 原型模式

### 类图



## 实现代码

- Photo.java

```
public class Photo {
    private int Size;

    public Photo(int size) {
        this.setSize(size);
    }

    public void setSize(int size) {
        Size = size;
    }

    public int getSize() {
        return Size;
    }

    @Override
    public String toString(){
        return "Photo [ Size = " + this.Size + "] ";
    }

}
```

- Product.java

```
public interface Product extends Cloneable{
    public void print();
    public Product clone();
    public Product deepclone();
}
```

- Resume.java

```
import java.io.*;

public class Resume implements Product{

    private String name;
    private int age;
    private Photo photo;

    public void setPhoto(int size) {
        this.photo.setSize(size);
    }

    public Resume(String name, int age, Photo photo){
        this.name=name;
        this.age=age;
        this.photo=photo;
    }

    public Resume(Resume resume){
        this.age= resume.age;
        this.name=new String(resume.name);
        this.photo= new Photo(resume.photo.getSize());
    }

    @Override
    public void print() {
        System.out.println("Resume [ Name = " + this.name + " , Age = " +
this.age + " , " + this.photo + " ]");
    }

    @Override
    public Product clone() {
        Product p = null;
        try{
            p = (Product) super.clone();
        } catch (CloneNotSupportedException e){
            System.err.println("Clone Not Supported");
        }
        return p;
    }

    @Override
    public Product deepclone() {
        Product p = new Resume(this);
        return p;
    }
}
```

- ResumeManager.java

```java
import java.util.HashMap;

public class ResumeManager {
    private HashMap<String,Resume> resumes;
    public ResumeManager(){
        resumes = new HashMap<String,Resume>();
    }

    public void saveResume(String name,Resume resume){
        resumes.put(name,resume);
    }

    public Product copyResume(String name){
        return resumes.get(name).deepclone();
    }

    public Product cloneResume(String name){
        return resumes.get(name).clone();
    }
}
```

- Test6.java

```java
public class Test6 {
    public static void main(String[] args) {
        ResumeManager resumeManager = new ResumeManager();
        Resume r1 = new Resume("A",20,new Photo(16));
        Resume r2 = new Resume("B",21,new Photo(64));
        resumeManager.saveResume("r1", r1);
        resumeManager.saveResume("r2", r2);
        ((Resume)resumeManager.cloneResume("r1")).print();
        ((Resume)resumeManager.cloneResume("r2")).print();
        //--------------------------------------------
        Resume r3 = (Resume) resumeManager.copyResume("r1");
        r3.setPhoto(32);
        r3.print();
        r1.print();
        // -------------------------------------------
        Resume r4 = (Resume) resumeManager.cloneResume("r2");
        r4.setPhoto(32);
        r4.print();
        r2.print();
    }
}
```

## 五、实验小结

```
本次实验学会了迭代器模式，适配器模式，模板方式模式和适配器模式，工厂方法模式，单例模式和原型模
式，在做这些作业的过程中，对java代码更加熟悉了，更深入的了解了面向对象的编程的思想，对代码整体的
设计的把握也更得心应手了
```