对象序列化实验报告

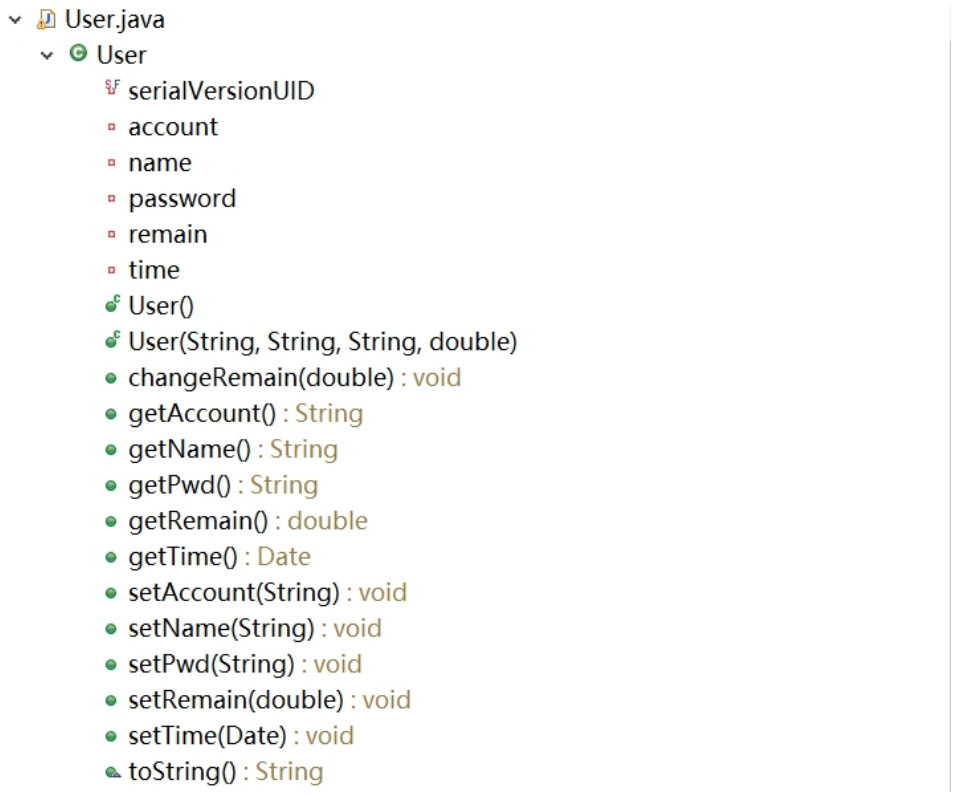
马斓轩 1813076

一、实验内容

通过对象序列化设计一个银行帐号信息保存和读取功能。并对密码和余额做加密。

二、算法设计

- 用户类（User）设计



- ✧ 帐号中包括：帐号（account）、户名（name）、密码（password），余额（remain）、创建时间（time）
- ✧ 包含功能：修改余额（changeRemain）
- ✧ 其他：重新 toString 方法，方便输出。

```
@Override
public String toString() {
    return "account=" + this.getAccount() + ", name=" + this.getName() + ", password=" + this.getPwd()
            + ", remain=" + this.getRemain() + ", time=" + this.getTime() + "";
}
```

- 银行类（Bank）设计
  - ✧ 对象序列化：创建文本文档 user.txt,创建输出流，向文件 user 中输出，并用 ObjectOutputStream 对其封装。witeObject()中的 list 是用来存储对象的动态数组。

```
File file = new File("D:", "user.txt");
try {
    file.createNewFile();
} catch (IOException e1) {
    e1.printStackTrace();
}
if (file.exists()) {
    ObjectOutputStream oos=null;
    try {
        oos = new ObjectOutputStream(new FileOutputStream(file));
        oos.writeObject(list);
    }catch(IOException e) {
        e.printStackTrace();
    }finally {
        try {
            oos.close();
        }catch(IOException e) {
            e.printStackTrace();
        }
    }
```

  - ✧ 反序列化：创建输入流，从 user.txt 文件中获取信息，存储到动态数组 list2 中，在控制台循环输出。

```
ObjectInputStream ois=null;
try {
    ois= new ObjectInputStream(new FileInputStream(file));
    @SuppressWarnings("unchecked")
    ArrayList<User> list2=(ArrayList<User>) ois.readObject();
    for (int i = 0; i < list2.size(); i++) {
        User temp=list.get(i);
        System.out.println(temp);
    }
}catch (Exception e) {
    e.printStackTrace();
}finally{
    try {
        ois.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

✧ 加密：RSA 加密是非对称加密,加密密钥分为公钥和私钥。这里使用私钥加密公钥解密。

```
static void rsa(String src) throws Exception {

    //1.初始化密钥
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(1024);//密钥长度为64的整数倍，最大是65536
    KeyPair keyPair = keyPairGenerator.generateKeyPair();
    RSAPublicKey rsaPublicKey = (RSAPublicKey) keyPair.getPublic();
    RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) keyPair.getPrivate();
    byte[] pk =  rsaPublicKey.getEncoded();
    byte[] sk = rsaPrivateKey.getEncoded();
    System.out.println("RSA公钥：" + parseByte2HexStr(pk));
    System.out.println("RSA私钥：" + parseByte2HexStr(sk));//可以将其保存到本地文件中

    //2.1私钥加密，公钥解密【加密】
    PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(rsaPrivateKey.getEncoded());
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    PrivateKey privateKey = keyFactory.generatePrivate(pkcs8EncodedKeySpec);
    Cipher cipher = Cipher.getInstance("RSA");
    cipher.init(Cipher.ENCRYPT_MODE, privateKey);
    byte[] result = cipher.doFinal(src.getBytes());
    System.out.println("RSA私钥加密：" + parseByte2HexStr(result));
}

private static String parseByte2HexStr(byte[] buf) {
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < buf.length; i++) {
        String hex = Integer.toHexString(buf[i] & 0xFF);
        if (hex.length() == 1) {
            hex = '0' + hex;
        }
        sb.append(hex.toUpperCase());
    }
    return sb.toString();
}
```

在主函数中对密码和余额进行加密：

```
for(int i=0;i<list.size();i++) {
    try {
        rsa(list.get(i).getPwd());
        rsa(list.get(i).getRemain()+"");
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}
```

- 案例测试与结果

  创建动态数组 list，直接在其中构造对象。

```
List<User> list =new ArrayList<User>();
list.add(new User("1111111111","a","zgjzzm",860.2)) ;
list.add(new User("2222222222","b","mlx000",222.3)) ;
```

控制台输出结果：

```
RSA公钥：30819F300D06092A864886F70D0101010500038180003081890281810086DD008C4FF3A4F32F1A76422072D03610879DFC5691956E198A3F7C6D
RSA私钥：308202760201003000D06092A864886F70D0101010500048202603082025C020100028181008 6DD008C4FF3A4F32F1A76422072D03610879DFC56
RSA私钥加密：3E2D848BBBD0EFDFFA014C35E45C511BC321DA05784AA36AE023B95C883D0095E913C2747643CCA5324CD9CF013053D196077ABE53D8C8F957
RSA公钥：30819F300D06092A864886F70D0101010500038180003081890281810093C1C2A3FE41AC90A68BBCF0E2FD850AC8956E287C9391820EEEBA3D576
RSA私钥：308202760201003000D06092A864886F70D0101010500048202603082025C020100028181009 3C1C2A3FE41AC90A68BBCF0E2FD850AC8956E287C9
RSA私钥加密：460C66DDD8A8896DC8F930E3356965D07F8277E0DBB4615A55F66E325C02B18B95C4C7BE2D1609340CC8D17CE6A4C4200ADEF429EF97285625
RSA公钥：30819F300D06092A864886F70D0101010500038180003081890281810 0A2CDC9C7D8FFEEDC2116B61E2D5AE87259AA36E33881AC13F4ED3BAB025
RSA私钥：308202750201003000D06092A864886F70D0101010500048202 5F3082025B0201000281810 0A2CDC9C7D8FFEEDC2116B61E2D5AE87259AA36E3388
RSA私钥加密：775DFB94A8A118BD8124BF9DF9AA0742E1D0F554AC674A0E2AFA914CB221FFB644663ADF6B15B8B774C3E2CD0B4B5364BAC0882BB3AB4D9CCC
RSA公钥：30819F300D06092A864886F70D0101010500038180003081890281810 0C306FF2774946DFAFCB0EDA7F8155DC231181F416DEB736FD29418FC167
RSA私钥：308202760201003000D06092A864886F70D0101010500048202603082025C020100028181 00C306FF2774946DFAFCB0EDA7F8155DC231181F416DE
RSA私钥加密：39AE0C9F56D06D070CC62F65FEAC3B347B460441F39E76600808DFA259D72467DB3793ADF74B8A607E25E2C9B63382A06EFF688728D5250500
account=1111111111, name=a, password=zgjzzm, remain=860.2, time=Fri May 29 15:52:29 CST 2020
account=2222222222, name=b, password=mlx000, remain=222.3, time=Fri May 29 15:52:29 CST 2020
```

三、思考与拓展

构造对象时使用的是直接构造，考虑用 Scanner 进行输入。在主函数中构造动态数组 info，按行接受控制台输入。对于每一行，用 createUser 进行字符串分离，并构造对象，放入 list 中。

```
List<User> list =new ArrayList<User>();
Scanner in=new Scanner(System.in);
ArrayList<String> info=new ArrayList<>();
while(in.hasNext()) {
    info.add(in.nextLine());
}
for(int i=0;i<info.size();i++) {
    User user=new User();
    user=createUser(info.get(i));
    list.add(user);
}
```

```
public static User createUser(String str) {
    User user=new User();
    //账户
    String varia="";
    for(int i=0;i<10;i++) {
        varia+=str.charAt(i);
    }
    user.setAccount(varia);
    //户名
    varia="";
    int tag=10;
    for(int i=10;i<str.length();i++) {
        if(str.charAt(i)==' ') {
            if(!varia.isEmpty()) {
                user.setName(varia);
                tag=i;
                varia="";
                break;
            }
            else {
                continue;
            }

        }
        else {
            varia+=str.charAt(i);
        }
    }
```

```
//密码
for(int i=tag;i<str.length();i++) {
    if(str.charAt(i)==' ') {
        if(!varia.isEmpty()) {
            user.setPwd(varia);
            tag=i;
            varia="";
            break;
        }
        else {
            continue;
        }

    }
    else {
        varia+=str.charAt(i);
    }
}
//余额
for(int i=tag;i<str.length();i++) {
    if(str.charAt(i)==' ') {
        if(!varia.isEmpty()) {
            double value = Double.valueOf(varia.toString());
            user.setRemain(value);
        }
        else {
            continue;
        }

    }
    else {
        varia+=str.charAt(i);
    }
}
return user;
```

数字签名是带有密钥（公钥、私钥）的消息摘要算法。主要作用是验证数据的完整性、认证数据来源、抗否认。在数字签名的实现中我们使用私钥签名、公钥验证。常用的数字签名算法包括 RSA、DSA、ECDSA。