```
    Chapter 9:  The Password class

import java.util.*;

public class Password
{
    final static int MIN_SIZE = 6;
    final static int MAX_SIZE = 15;
          static int maxHistory = 4;
          static int expiresNotifyLimit = 3;

    private int maxUses = 120;
    private int remainingUses = maxUses;
    private boolean autoExpires = true;
    private boolean expired = false;

    private ArrayList pswdHistory;


    //Constructors for objects of class Password
    public Password(String newPassword) throws Exception
    {
        pswdHistory = new ArrayList(maxHistory);
        set(newPassword);
    }

    public Password(String newPassword, int numMaxUses) throws
 Exception
    {
        pswdHistory = new ArrayList(maxHistory);
        maxUses = numMaxUses;
        remainingUses = numMaxUses;
        set(newPassword);
    }

    public Password(String newPassword, boolean pswdAutoExpires)
 throws Exception
    {
        pswdHistory = new ArrayList(maxHistory);
        autoExpires = pswdAutoExpires;
        set(newPassword);
    }

    public Password(String newPassword, int numMaxUses, boolean
pswdAutoExpires) throws Exception
```

```java
    {
        pswdHistory = new ArrayList(maxHistory);
        maxUses = numMaxUses;
        remainingUses = numMaxUses;
        autoExpires = pswdAutoExpires;
        set(newPassword);
    }

    public boolean getAutoExpires()
    {
        return autoExpires;
    }

    public void setAutoExpires(boolean autoExpires)
    {
        this.autoExpires = autoExpires;
        if(autoExpires)
            remainingUses = maxUses;
}

    public boolean isExpired()
    {
        return expired;
    }

    public void setExpired(boolean newExpired)
    {
        expired = newExpired;
    }

    public int getExpiresNotifyLimit()
    {
        return expiresNotifyLimit;
    }

    public void setExpiresNotifyLimit(int newNotifyLimit)
    {
        if(newNotifyLimit >= 2 && newNotifyLimit <= 20)
            expiresNotifyLimit = newNotifyLimit;
    }

    public int getMaxHistory()
    {
        return maxHistory;
    }

    public void setMaxHistory(int newMaxHistory)
    {
        int overage = 0;
        if(newMaxHistory >= 1 && newMaxHistory <= 10)
        {
            maxHistory = newMaxHistory;
            overage = getHistorySize() - maxHistory;
            if(overage > 0)                         //if size > max allowed
            {
                do{
                    pswdHistory.remove(0); //then remove overage number
                    overage --;            //of oldest pswds from list
                }while(overage > 0);

                pswdHistory.trimToSize(); //resize capacity to max allowed
            }
        }
    }

    public int getRemainingUses()
    {
        return remainingUses;
    }

    public int getHistorySize()
    {
        return pswdHistory.size();
    }

    public boolean isExpiring()
    {
        boolean expiring = false;

        if(autoExpires && remainingUses <= expiresNotifyLimit)
            expiring = true;

        return expiring;
    }

    //Sets password to a new value ; keeps current & previous
```

```java
values in history up to max number
    public void set(String pswd) throws Exception
    {
        String encryptPswd;
        boolean pswdAdded = true;

        pswd = pswd.trim();                             //remove any leading, trailing white space
        verifyFormat(pswd);                             //verify password was entered correctly
        encryptPswd = encrypt(pswd);                    //convert to encrypted form

        if(!pswdHistory.contains(encryptPswd))  //if pswd not in recently used list
        {
            if(pswdHistory.size() == maxHistory) //if list is at max size
                pswdHistory.remove(0);           //remove oldest password from list

            pswdAdded = pswdHistory.add(encryptPswd); //add new pswd to end of ArrayList

            if(!pswdAdded)                             //should never happen
                throw new Exception("Internal list error - Password not accepted");

            if(expired)                              //if pswd is expired
                expired = false;                     //reset to not expired

            if(autoExpires)                          //if pswd autoexpires,
                remainingUses = maxUses;                 //reset uses to max
        }
        else
            throw new Exception("Password recently used");
        }

    //Validates entered password against most recently saved value
    public void validate(String pswd) throws Exception
    {
        String encryptPswd;
        String currentPswd;
        int currentPswdIndex;

        verifyFormat(pswd);                     //verify password was entered properly
        encryptPswd = encrypt(pswd);            //convert to encrypted form

        if(!pswdHistory.isEmpty())              //at least one password entry in history
        {
            currentPswdIndex = pswdHistory.size()-1;
            currentPswd = (String)pswdHistory.get(currentPswdIndex);

            if(!encryptPswd.equals(currentPswd)) //if not most recent password
                throw new Exception("password is invalid");

            if(expired)
                throw new Exception("Password has expired - please change");

            if(autoExpires)
            {
                --remainingUses;
                if(remainingUses <= 0)
                    expired = true;
            }
        }
        else
            throw new Exception("No password on file - list corrupted!");//should never happen


    }

    //Verifies password has proper format
    private void verifyFormat(String pswd) throws Exception
    {
        boolean numFound = false;

        if(pswd.length() == 0)
            throw new Exception ("No password provided!");

        if(pswd.length() < MIN_SIZE)
            throw new Exception("password must be at least" + MIN_SIZE + "charecters in length");

        if(pswd.length() > MAX_SIZE)
            throw new Exception ("Password cannot be greater the " + MAX_SIZE + " charecters in length");

        // scan through password to find if at least 1 number us used.
        for(int i=0;i< pswd.length() && !numFound; ++i)
            if(Character.isDigit(pswd.charAt(i)))
                numFound = true;

        if(!numFound)
            throw new Exception("Password is invalid - must have at least one number");
        }

    //Encrypts original password returning new encrypted String
    private String encrypt(String pswd)
    {
        StringBuffer encryptPswd;
        int pswdSize = 0;
        int midpoint = 0;
        int hashCode = 0;

        //swap first and last half of password
        pswdSize = pswd.length();
        midpoint = pswdSize / 2;
        encryptPswd = new StringBuffer(pswd.substring(midpoint)      //get last half of pswd
            + pswd.substring(0,midpoint));                           //and concatenate first half

        encryptPswd.reverse(); //reverse order of characters in password

        for(int i = 0; i < pswdSize; ++i)
            encryptPswd.setCharAt(i, (char)
(encryptPswd.charAt(i)& pswd.charAt(i)));

        hashCode = pswd.hashCode(); //hash code for original password
        encryptPswd.append(hashCode);

        return encryptPswd.toString();
    }
}


```