Chapter 7: Arrays & More
Lab Exercises and MiniProject

Creating and reversing arrays

Write a program that asks the user for a integer. The program should create an array as big as the entered value. Then the user will be asked to enter values until the array will be filled. (Use a while loop and store these values in array.

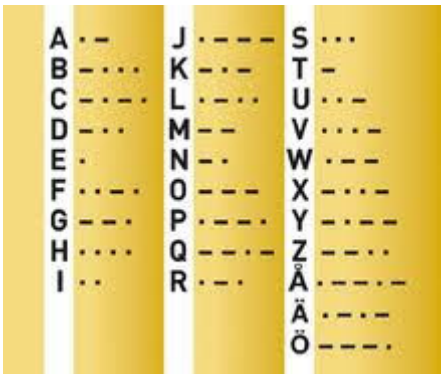
1. Print the array. Use a for-loop.

2. Then reverse the array elements so that the first element becomes the last element, the second element becomes the second to last element, and so on, with the old last element now first.

3. Do not just reverse the order in which they are printed; actually change the way they are stored in the array. You can solve this problem by creating a new temporary array and copy all data from the original array to the temporary.

The Morse Alphabet

In May of 1844 Samuel F.B. Morse sent the message" What hath God wrought!" by telegraph from Washington to Baltimore, heralding the beginning of the age of electronic communication. To make it possible to communicate information using only the presence or absence of a single tone, Morse designed a coding system in which letters and other symbols are represented as coded sequences of short and long tones, traditionally called dots and dashes. In Morse code, the 26 letters of the alphabet are represented by following codes. Only upper case letters are used in Morse.



You can easily store these codes in a program by declaring an array with 29 elements and store the sequence of characters corresponding to each letter as Strings. Name this array to morseArray.

1. Write a program that reads a string from the user and translates each letter in the string to its equivalent in Morse code, using periods do represent dots and hyphens to represent dashes. Separate the words by using System.out.println() whenever you encounter a space in the input, but ignore all other punctuation characters.
Tips! Se below some logical structure for such a programme

```
String morse= "";
char ch=0;
int counter =0;
while (  counter< message.length())
{

ch=message.charAt( counter);
switch ( ch)
{
  case 'A':  morse= morse +  morseArray [0]  // if the first
place correspond to
                    break;

 case 'B':  morse=
                  break;

 case 'C':

}
 Counter++;

}
…..
```

MyCrypto

One very simple encrypting method is to keep all the letters in the alphabet in an array and the letters you want to use as encrypted letter in another array.

For example, if this is the first array containing the alphabet

A	B	C	D	E	F	G	….	…	Z

and the next array containing the "encrypted" alphabet, just put the letters in the wished order

K	P	L	A	M	B	Z	….	…	R

then if you read the word "CAB" from the user then this should be translate to "LKP".

1. In a class called MYCrypto, write a program and use the description above to encrypt messages from the user.

2. Now modify your program as follow:

Move your code from main() to a method called

public static String encrypt (String inMessage) { …. }

The method takes as input a String, your message to be encrypt, and returns the encrypted message. Do you need to have the array declaration in main() or in the method?

Test your method in main method.

3. Now write a method to decrypt message

public static String decrypt (String inMessage) { your code here }

Prove it works by using the method in your main programme.

A Very Simple Bank Program

Copy to this working directory the Account class from the last lab.

```
public class Account
{
  private double balance;
  private String name;
 private long acctNum;

  //-----------------------------------------------
  //Constructor -- initializes balance, owner, and account number
  //-----------------------------------------------
  public Account(double initBal, String owner, long number)
  {
    balance = initBal;
    name = owner;
    acctNum = number;
  }

  //-----------------------------------------------
  // Checks to see if balance is sufficient for withdrawal.
  // If so, decrements balance by amount; if not, prints message.
  //-----------------------------------------------
  public void withdraw(double amount)
  {
    if (balance >= amount)
       balance=balance- amount;

    else
       System.out.println("Insufficient funds");
  }

  //-----------------------------------------------
  // Adds deposit amount to balance.
  //-----------------------------------------------
  public void deposit(double amount)
  {
     if( amount >0)
     balance=balance+ amount;
```

```
  }

  //-----------------------------------------------
  // Returns balance.
  //-----------------------------------------------
  public double getBalance()
  {
    return balance;
  }


  //-----------------------------------------------
  // Returns a string containing the name, account number, and balance.
  //-----------------------------------------------
  public String toString()
  {
    return name+ " " + acctNum+ " " + balance;
  }

  public long getAcctNum()
  {
    return acctNum;
  }
}
```

The next class SimpleBank is a very simple bank program. Copy the code in java file. Save it. Compile and run the program. Hopefully it works but …nothing happened when you choose from the menu. This is because we don't have any code to execute for each case. This you have to do. Write the code needed in all cases as described in the menu.

```
   import java.util.Scanner;

   public class SimpleBank
  {
     static Scanner scan = new Scanner(System.in);


     static  Account [] bank =new Account[100];

      static int position =0;  // used to refer to right "position"  in
the bank..


  //-----------------------------------------------------------
      public static void main(String[] args)
     {


        printMenu();
        int choice = scan.nextInt();

        while
```

```
      (choice != 0)
      {
         dispatch(choice);
         printMenu();
         choice = scan.nextInt();
      }
   }

    public static void dispatch(int choice)
   {



      switch(choice)
      {
         case 0:  {
            // Ask the user for a name accountnumber
           // create a new account and added to the bank array.
          break;


              }
        case 1:{

              break;
            }
         case 2:                {

            break;
            }

          case 3:               {

            break;
            }

          case 4               {

            break;
            }


         default:
            System.out.println("Sorry, invalid choice");
      }
   }

//----------------------------
// Print the user's choices
//----------------------------
    public static void printMenu()
   {
      System.out.println("\n  Welcome to MY BANK ");
      System.out.println("   ====");
      System.out.println("0: Create new Account ");
      System.out.println("1: Deposit" );
      System.out.println("2: Withdraw");
      System.out.println("3: Show account information");
      System.out.println("4 Show bank information");
```

```
            System.out.print("\nEnter your choice: ");
        }
    }
```

Miniproject

Caesar

Caesar's code or **Caesar shift**, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a shift of 3, `A` would be replaced by `D`, `B` would become `E`, and so on. The method is named after Julius Caesar, who used it to communicate with his generals.

To decode the message each letter is shifted the same number of characters backwards.

An improvement can be made to this encoding technique if we use a repenting key. Instead of shifting each character by a constant amount we can shift each character by a different about using a list (array) of key values. If the message is longer than the list of key values we just start using the keys over again from the beginning. For example if the key values are 3 1 7 4 2 5 , then the first character is shifted by three, the second character by one, the third by seven, etc.

n	o	v	a	n	j	g	h	l		m	u		u	r	x	l	v
3	1	7	4	2	5	3	1	7		4	2		5	3	1	7	4
k	n	o	w	l	e	d	g	e		i	s		p	o	w	e	r

The first row in the encoded message, the second are the keys, the third is raw is the decoded message.

Write methods to encrypt and to decrypt characters. Only printable characters will be encrypted and decrypted (not space character).

1. Use the ascii code of the character and the key to encrypt / decrypt messages. Start by implementing following methods in a class called Caesar.

public static char encrypt_character (char tkn, int key) {….your code… }

public static char decrypt_character (char tkn, int key) {…your code…}

I create for you a graphical user interface in the class CaesarFrame (download it in your lab directory), to be used together with your methods, as you saw in my lecture. Try to read and understand how user interface works.

The only think you have to do is to implement the **encrypt_message (),** end **decrypt_message() in the class CaesarFrame**.

For that you should use the **encrypt_character, and decrypt_character.**

Tips! Create an array to keep the keys. Use a variable called currentKey to keep track of the last used key. After using the last key in the array, you should start with the first key again.

Extra projektuppgift (ej obligatoriskt)

1. Try to improve the project so the encrypted text should not include other characters then letters. This means that (ascii + key) should always be in the interval 65-90.
2. Find a solution to include the Swedish letters and numbers.