Lab 4 Wallet

Purpose

Purpose of this lab is for you to develop a program where many objects are created from a class. Primitives and objects are passed as parameters to class methods.

Problem specification

Your new `Wallet` class implements a wallet that contains banknotes. A banknote is represented as an `int` for simplicity, `1` for a $1 bill, `5` for a $5 bill, and so on. You are required to use just a simple array of `int` to hold the banknotes. You may NOT use an array list.

Here are some example wallets printed out:

```
Wallet[5, 50, 10, 5]
Wallet[]
Wallet[1, 5, 10, 50, 5]
```

Here's the outline of the `Wallet` class. You will implement each method as described below.

```
public class Wallet
{
    // max possible # of banknotes in a wallet
    private static final int MAX = 10;

    private int contents[];
    private int count;     // number of banknotes stored in contents[]

    public Wallet()
    {
        // your code goes here
    }

    public Wallet(int a[])
    {
        // your code goes here
    }

    public String toString()
    {
        // your code goes here
    }

    public int value()
    {
        // your code goes here
    }
```

```
    public void reverse()
    {
        // your code goes here
    }

    public void add(int banknote)
    {
        // your code goes here
    }

    public void transfer(Wallet donor)
    {
        // your code goes here
    }

    public void sort()
    {
        // your code goes here
    }

    public boolean remove(int banknote)
    {
        // your code goes here
    }

    public boolean sameBanknotesSameOrder(Wallet other)
    {
        // your code goes here
    }

    //EXTRA CREDIT methods follow...

    public Wallet removeBanknotePairs(Wallet w)
    {
        // your code goes here
    }
}
```

Instance variables

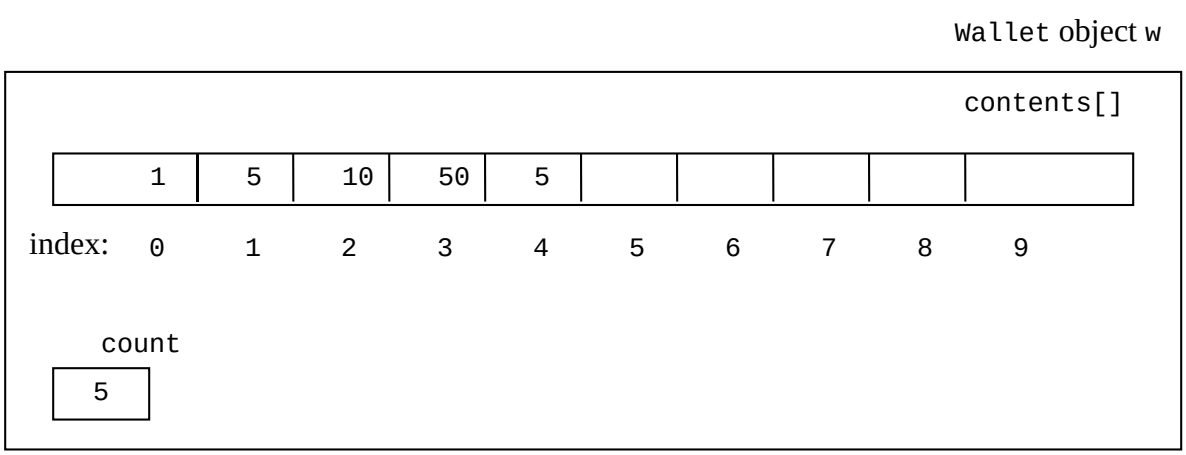
Use exactly and only these instance variables:

`Wallet` uses the `contents[]` array to store `int`s representing banknotes. `count` holds the number of banknotes in `contents[]`, so must be updated every time a banknote is added or removed.

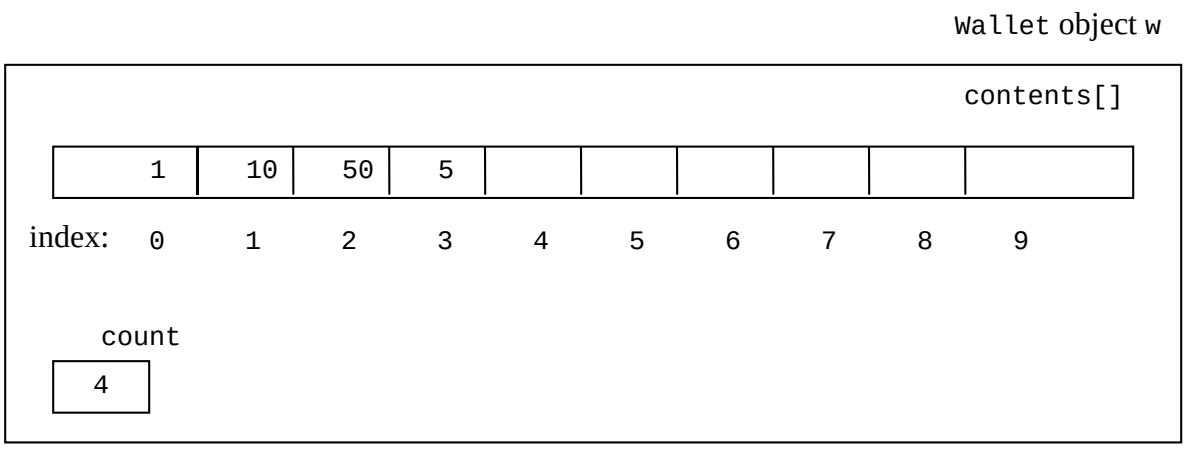
`count` is `0` when a wallet is empty. `MAX` is the maximum number of banknotes a wallet can hold, so count is `MAX - 1` when a wallet is full. `count` will always be the index where the next banknote is to be added.

Banknotes in a wallet are maintained in the order of arrival. A new banknote is simply added to the end of `contents[]` and `count` is incremented by 1.

When a banknote is removed from a wallet, the first occurrence of the value is removed from `contents[]` and `count` is decremented by 1. Importantly, all banknotes above the removed value must be moved 'down' so that no 'holes' open in the array. For example, if wallet `w` is `Wallet[1, 5, 10, 50, 5]`:



- then after `w.remove(5)`, `w` will be `Wallet[1, 10, 50, 5]`:



Methods

You will complete the method bodies as follows:

`public Wallet()`
initialize the wallet to empty

- allocate memory for the `contents[]` array
- initialize `count`

`public Wallet(int a[])`
initialize the wallet using the array of `int` named `a[]`
an overloaded constructor

- allocate memory for the `contents[]` array
- initialize `contents[]` from `a[]`
- initialize `count`

`public String toString()`
return a textual representation of the wallet, in the standard format. For example:
`Wallet[5, 50, 10, 5]`

- use a `StringBuffer` to build the returned value
- convert the `StringBuffer` to a `String` and return the `String`

`public int value()`
calculate the value of the banknotes in the wallet. For example, if wallet is:
`Wallet[5, 50, 10, 5]`, value is `70`

- must use `count` to do this, to traverse ONLY the banknotes in the wallet
- (cannot use `contents.length` since this traverses every element in the array including elements that may not have been explicitly initialized, which is dangerous)
- return this number of dollars

`public void reverse()`
reverse the order of banknotes in a wallet e.g.
`Wallet[1, 5, 10, 50, 5]` when reversed becomes `Wallet[5, 50, 10, 5, 1]`
(IMPORTANT NOTE: do not create a new `Wallet` object!)

- suggested algorithm:
  start an index pointing at the first banknote in `contents[]`
  start another index pointing at the last of the banknotes in `contents[]`
  while these indexes do not meet or cross over
    swap values at the indexes (use a temp variable to do this)
    move two indexes towards one another

`public void add(int banknote)`
add `banknote` to the wallet

- `banknote` is the banknote to add e.g. `50`, `5`, etc

- add `banknote` to the end of `contents[]` and update `count`

`public void transfer(Wallet donor)`
transfer the contents of one wallet (the `donor` wallet) to the end of another, the receiver. (The receiver will be the `Wallet` object calling the method (i.e. the invoking object)). Leave the `donor` wallet empty. For example:

if the receiver is `Wallet[5, 10, 50, 50]`
and `donor` is `Wallet[5, 5, 10, 1, 50, 5]`
then after the transfer:
receiver will be `Wallet[5, 10, 50, 50, 5, 5, 10, 1, 50, 5]`
and `donor` will be `Wallet[]`

- should call the `add()` method as you implement this
- to set a wallet to empty, simply set its `count` to `0`

`public void sort()`
sort the banknotes in a wallet into ascending order e.g.
`Wallet[5, 50, 10, 5, 1]` when sorted becomes `Wallet[1, 5, 5, 10, 50]`
(IMPORTANT NOTE: do not create a new `Wallet` object!)

- suggested algorithm:
  - we saw in Week 7 a sort method that works for an array of `Integer` objects containing `MAX` elements
  - take this method and modify syntax to work for `contents[]` and `count`

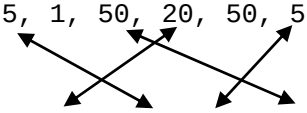
`public boolean remove(int banknote)`
remove first occurrence of `banknote` from the wallet. Return `true` if `banknote` was removed, `false` otherwise

- (this `banknote` may not be in the wallet)
- if `banknote` is removed, must update `contents[]` so that no holes appear, and `count`

`public boolean sameBanknotesSameOrder(Wallet other)`
return whether two wallets have exactly the same banknotes in exactly the same order

Extra credit
`public Wallet removeBanknotePairs(Wallet w)`
create a new wallet and add to it pairs of banknotes removed from two other wallets. Return the new wallet. For example:

if `w1` is `Wallet[5, 1, 50, 20, 50, 5]`



and `w2` is `Wallet[20, 10, 5, 5, 5, 50, 10]`

then after the method call

```
Wallet w3 = w1.removeBanknotePairs(w2);
```

`w1` will be: `Wallet[1, 50]`
`w2` will be: `Wallet[10, 5, 10]`
`w3` will be: `Wallet[5, 5, 50, 50, 20, 20, 5, 5]`

The `WalletTester` class

The `WalletTester` class tests your new `Wallet` class. Source code for `WalletTester` is given below, and can be downloaded from Blackboard, Course Documents, Week 10 folder, Example programs.

```
/**
 * Test the Wallet class.
 *
 * @author Anthony W. Smith
 * @version 5/31/2028
 */
public class WalletTester
{
    public static void main(String args[])
    {
        // create a new Wallet object using an array
        int a[] = {5, 50, 10, 5};
        Wallet myWallet = new Wallet(a);

        // show the contents of myWallet
        System.out.println("myWallet contains: " +
                                        myWallet.toString());

        // print the value of myWallet
        System.out.println("\nvalue of myWallet is: $" +
                                            myWallet.value());

        // reverse the order of banknotes in myWallet
        myWallet.reverse();
        System.out.println("\nmyWallet reversed contains: " +
                                          myWallet.toString());

        // transfer all the banknotes from myWallet to yourWallet!
        Wallet yourWallet = new Wallet();
```

```
        yourWallet.add(1);
        yourWallet.transfer(myWallet);
        System.out.println("\nnow myWallet contains: " +
                                            myWallet.toString());
        System.out.println("yourWallet contains: " +
                                           yourWallet.toString());

        // sort yourWallet
        yourWallet.sort();
        System.out.println("\nyourWallet sorted is: " +
                                         yourWallet.toString());

        // remove all $5 banknotes from yourWallet
        while (yourWallet.remove(5))
            ;
        System.out.println("\nyourWallet with $5s removed is: " +
                                           yourWallet.toString());

        // check whether two wallets have the same banknotes
        // in the same order
        int b[] = {10, 5, 10};
        Wallet tom = new Wallet(b);

        int c[] = {10, 10, 5};
        Wallet dick = new Wallet(c);

        int d[] = {10, 5, 10};
        Wallet harry = new Wallet(d);

        System.out.println(
             "\ntom has same banknotes in same order as dick: " +
                                  tom.sameBanknotesSameOrder(dick));
        System.out.println(
            "tom has same banknotes in same order as harry: " +
                                 tom.sameBanknotesSameOrder(harry));

        // EXTRA CREDIT – compare two wallets and remove banknote pairs
        int e[] = {5, 1, 50, 20, 50, 5};
        Wallet w1 = new Wallet(e);

        int f[] = {20, 10, 5, 5, 5, 50, 10};
        Wallet w2 = new Wallet(f);

        Wallet w3 = w1.removeBanknotePairs(w2);
        System.out.println("\nw1 is: " + w1.toString());
        System.out.println("w2 is: " + w2.toString());
        System.out.println("w3 is: " + w3.toString());
    }
}
```

Extra credit
When you have completed all other methods, implement and test `removeBanknotePairs()` for 20% extra credit. You must test your method using the `w1` and `w2` wallets given above in the description of `removeBanknotePairs()`.

Hints
- to start this lab, first work carefully through `WalletTester` line by line, writing down on a piece of paper what outputs should be produced...
- this makes sure you understand what every method does, and gives you the expected output you need to test your program
- then design, code and test each method in turn, in the order given above. You will have to comment out in `WalletTester` and `Wallet` the methods you have not yet implemented

Required
- the intent of the lab is that you learn how to program by writing your own methods...
- so (other than some `StringBuffer` methods in your `toString()`) do not use API methods for processing arrays
- `WalletTester` in your submission may not be changed in any way
- every method must have a clear, meaningful Javadoc comment
- automatically and routinely use all the other components of simplicity and clarity, as listed in Blackboard, Course Information, "How labs are graded"
- when you have thoroughly tested your program and are certain it is correct, save your output into the `output.txt` file

Lab 4 submission
- deadline for this lab is 3 weeks, by end of Sunday 4/17
- zip your BlueJ project plus `output.txt` output file and email to me at awsmith@palomar.edu
  - you will lose points if you do not include a file named `output.txt` containing the output of your program

- your email Subject line must say 'CSCI 114 Lab 4' followed by your full name, so that it filters to the correct email folder for grading
  - you will lose points if you format your email Subject incorrectly
  - e.g. my email Subject would be:

    ```
    CSCI 114 Lab 4 Anthony W. Smith
    ```

- this is a graded lab, so a reminder that you may not copy code from other people
- reminder that late labs will be penalized 2 points per week or part of week late