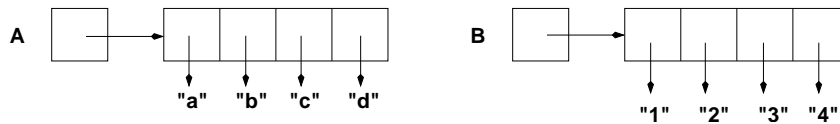


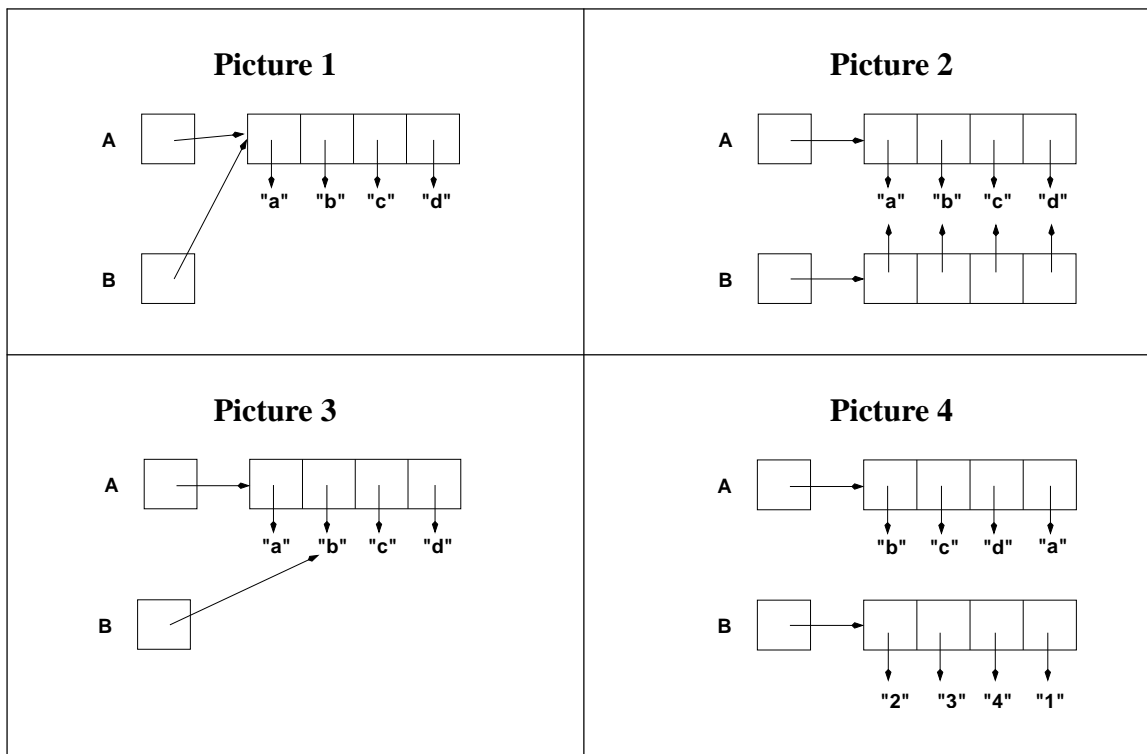
WES-CS GROUP MEETING #10

Exercise 1: Arrays and Aliasing

Assume that you have two *String* arrays, A and B, initialized as shown below.



For each of the pictures shown below, write code that would change arrays A and B from their initial values as shown above, to the new value shown in the picture. If there is no such code, explain why.



Exercise 2: Arrays of Objects

Assume that the following *Person* class has been defined.

```
public class Person {
    private String name;
    private int birthYear;
    private String eyeColor;

    // constructor
    public Person(String nme, int brthYr, String eye) {
        name = nme;
        birthYear = brthYr;
        eyeColor = eye;
    }

    // accessor methods
    public String getName() { return name; }
    public int getBirthYear() { return birthYear; }
    public String getEyeColor() { return eyeColor; }
}
```

First, choose four people in your group and draw an array called *people* that contains four *Person* objects representing those four people (use strings with all capital letters for the names and eye colors).

Now (in groups of two or three) play a game of *concentration* using the green and gold, or blue and yellow, or pink and white cards. Each green/blue/pink card has a Java expression, and each gold/yellow/white card has a value. Start with all of the cards upside down. When it's your turn, you turn over one card of each color. If they match you keep those two cards, and keep going; otherwise, your turn is over.

The game ends when you run out of green/blue/pink cards or the remaining gold/yellow/white cards don't match any green/blue/pink cards.

Whoever has the most cards wins!

Exercise 3: 2D Arrays

A program that lets two people play a game of tic-tac-toe (by drawing the current board, asking the players for the next move, etc.) might use a 2-dimensional array of characters (containing spaces, 'X's and 'O's) to represent the board. Let's assume that we have a TicTacToe class, partially defined as follows.

```
public class TicTacToe {
    /*****
     * fields
     *****/
    private char[][] board; // should contain only spaces, Xs, Os
    private int boardSize;

    /*****
     * constructor
     *****/
    public TicTacToe (int size) {
        boardSize = size;

        // PART (a)
    }

    /*****
     * public methods
     *****/
    public boolean boardFull() {
        // PART (b)
    }

    public boolean hasWon(char player) {
        // PART (c)
    }
}
```

Part (a)

Complete the TicTacToe constructor so that it initializes the board field to represent an empty, square board of the given size.

Part (b)

Complete the `boardFull` method so that it returns `true` if the board has no empty spaces, and otherwise return `false`.

Part (c)

Complete the `hasWon` method so that it returns `true` if some row, or some column, or one of the two diagonals of the board is completely filled with the given character (which should be either an 'X' or an 'O').

To write this method so that it is easy to understand, you may want to use some private methods to handle the different kinds of winning positions.

Exercise 4: Logical Thinking

Play the game *Connect-4*. Try to develop a strategy. For example, find a configuration that isn't itself a win for you, but that guarantees that you will win on your next move, no matter what your opponent does.

Green/Blue/Pink Cards

```
people[1].getBirthYear()
```

```
(people[0].getEyeColor().compareTo(people[3].getEyeColor())) < 0
```

```
people[people.length-1].getBirthYear() > 1985
```

```
people[2].getName().size() < 6;
```

```
(people[2].getName().substring(0,1)).equals((people[3].getEyeColor().substring(0,1)))
```

```
people[1].getEyeColor().charAt(1)
```

Gold/Yellow/White Cards

1987

1986

1988

1989

true

false

true

false

true

false

true

false

L

R

A

1985