CS230: **Solutions** to Practice Problems for Midterm

NOTE: *We did **not** include comments or javadoc in the solutions. On the exam, it's best to include these (unless otherwise specified), as this will help us give you partial credit.*

Problem 1. You are given the following method definition:

```
public static boolean isVowel(char c){
     c = Character.toLowerCase(c);
     return((c == 'a')||(c == 'o')||(c == 'e')||(c == 'i')||(c == 'u'));
}
```

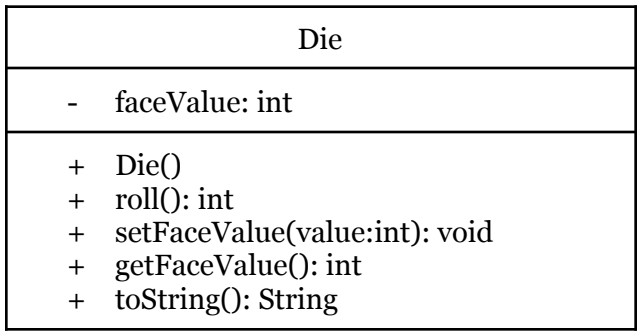
Using the `isVowel()` method, write another method, **`countConsonants()`**, that takes as input a String and returns the number of consonants in it. A consonant is considered to be any letter that is NOT a vowel. For example, **`countConsonants("Vowel")`** should return 3.

```
// Write any assumption you are making. No need to add javadoc for this method


public static int countConsonants(String s) {
  int total = 0;
  for (int i = 0; i < s.length(); i++) {
    if (!isVowel(s.charAt(i))) {
      total++;
    }
  }

  return total;
}
```

Problem 2. Recall the `Die` class from earlier in the semester. We've included its UML diagram here:



This represents a normal six-sided die, so the `roll()` method returns an integer from 1-6. Write a code fragment that creates two `Die` and prints how many rolls it takes to get a "7".

```
Die die1 = new Die();
Die die2 = new Die();

int roll1;
int roll2;
int num_rolls = 0;

do {
  roll1 = die1.roll();
  roll2 = die2.roll();
  num_rolls++;
} while (roll1 + roll2 != 7);

System.out.println("It took " + num_rolls + " rolls to get a 7.")
```

Problem 3. You are given a 2-dimensional array of integers that represents a grid of integers. Write a method that finds the sum of the elements located on the boundary. The boundary of the matrix includes the first and the last row, and the first and the last column. For example, consider the following array:

```
int[][] grid = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12},
    {13, 14, 15, 16}
};
```

For `grid`, the method should return: 1 + 2 + 3 + 4 + 8 + 12 + 16 + 15 + 14 + 13 + 9 + 5 = 102.

```
public static int sumBoundary(int[][] grid) {
  int columns = grid[0].length;
  int rows = grid.length;

  int total = 0;

  for (int i = 0; i < columns; i++) {
    total += grid[0][i];
    total += grid[rows - 1][i];
  }

  for (int i = 1; i < rows - 1; i++) {
    total += grid[i][0];
    total += grid[i][columns - 1];
  }

  return total;
}
```

Problem 4. In this problem, you will design a class named Book, representing a book in a library.

4A. Create the Book class. Each book has the following properties:

- title (a string)
- author (a string)
- isbn (a ***unique*** integer identifying the book)
- isCheckedOut (a boolean indicating whether the book is currently checked out)

Further, for this problem, implement the following methods:

1. A constructor that takes in the title and author, and initializes all the fields, including the book's *unique* isbn.
2. A method `checkoutBook` that marks the book as "checked-out" from the library. If the book is already checked out, your method should throw an `Exception`.
3. A method `returnBook` that marks the book as "returned" to the library. If the book wasn't checked out to begin with, your method should throw an Exception.
4. A `toString` method.

```
public class Book {
    private String title;
    private String author;
    private int isbn;
    private static int globalISBN = 0;
    private boolean isCheckedOut = false;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isbn = globalISBN;
        globalISBN++;
    }

    public void checkoutBook() throws Exception {
        if (this.isCheckedOut) {
            throw new Exception("Book already checked out!");
        }
        this.isCheckedOut = true;
    }

    public void returnBook() throws Exception {
        if (!this.isCheckedOut) {
            throw new Exception("Book not checked out!");
        }
        this.isCheckedOut = false;
    }

    public String toString() {
        return ("Book, " + this.title + ", by " + this.author
                + " (" + this.isbn + "): checked out = " + this.isCheckedOut);
    }
}
```

4B. Implement a main method in which you:

1. Create three Book objects.
2. Check out one of the books.
3. Print out the details of all the books.

```
public static void main(String[] args) {
  Book b1 = new Book("Shadow and Bone", "Leigh Bardugo");
  Book b2 = new Book("Siege and Storm", "Leigh Bardugo");
  Book b3 = new Book("Ruin and Rising", "Leigh Bardugo");

  try {
    b2.checkoutBook();
  } catch (Exception e) {
    System.out.println("Book already checked out!");
  }

  System.out.println(b1);
  System.out.println(b2);
  System.out.println(b3);
}
```

Problem 5. In this problem, you will implement a `BingoCard` class to support a Bingo game. In a Bingo game, one gets a 5-by-5 board with numbers from 0 to 24 (inclusive) randomly placed (without repetition on the board). For example:

0	9	10	4	15
14	13	1	7	8
2	21	12	17	18
22	3	23	20	16
11	24	19	6	5

As numbers are called out, you mark the numbers on your grid. When you've marked 5 in a row or column, you call "Bingo!" and win a prize. For example, the numbers "2, 21, 12, 17, 18" make up a Bingo.

In this problem, you will implement `BingoCard` class.

5A. Implement the `BingoCard` class so that it contains the following:

- `board`: A 5-by-5 board game, represented by an integer array.
- When creating the instance of the class, the array is initialized with numbers between 0 and 24 (inclusive), with **no repetitions.** The numbers should be in **random locations** in the array. *Hint: you can start by placing the numbers 0-24 in the array in order, and then use the Random class to get a random ordering somehow.*

```
public class BingoCard {
    private int[][] board;

    public BingoCard() {
        this.board = new int[5][5];

        // Initialize the array to numbers 0 through 24
        int count = 0;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                this.board[i][j] = count;
                count++;
            }
        }

        Random generator = new Random();
        int x, y, tmp;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                // Pick an entry (x, y) at random
                x = generator.nextInt(5);
                y = generator.nextInt(5);

                // Swap it with the entry at (i, j)
                tmp = this.board[x][y];
                this.board[x][y] = this.board[i][j];
                this.board[i][j] = tmp;
            }
        }
    }
}
```

5B. Add the instance method `mark` to your class. Given a number between 0-24 (inclusive), this method, marks it on the board. *Hint: you can use a denote an element on the board as "marked" by using a value outside the range of 0-24.*

```
public void mark(int n) {
    // Look for the entry in the array that stores 'n'
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (this.board[i][j] == n) {
                // Set the entry to "marked" via -1
                this.board[i][j] = -1;
                return;
            }
        }
    }
}
```

5C. Add the instance method `hasBingo` to your class. This method should return true if the board has 5 elements marked in a row (ignore columns and diagonals for this question).

```
public boolean hasBingo() {
    boolean bingo;

    for (int i = 0; i < 5; i++) {
        bingo = true;

        for (int j = 0; j < 5; j++) {
            if (this.board[i][j] != -1) {
                bingo = false;
                break;
            }
        }

        if (bingo) {
            return true;
        }
    }

    return false;
}
```

Problem 6. Consider the program below, and then answer the questions. For partial credit, show your work (draw a memory diagram and execute the program line by line).

```
public class TestArray {
    public static int[] changeArray (int[] arr){
        int[] otherArr = {2, 4, 6, 8};
        for(int i = 1; i < arr.length; i++){
            if (arr[i] % 2 == 0) {
               arr[i] = otherArr[i];
            }
        }
        return otherArr;
    }

    public static void printArray (int[] arr){
        for(int i = 0; i < arr.length; i++){
            System.out.print(arr[i] + ", ");
        }
        System.out.println();
    }

    public static void main(String[] args){
        int[] array1 = {10, 11, 12, 13};
        int[] array2 = changeArray(array1);

        System.out.println("contents of array1:");
        printArray(array1); // Q2A

        System.out.println("contents of array2:");
        printArray(array2); // Q2B
    }
}
```

2A. What would appear in the terminal after running the line marked as `Q2A`?

- ☐ `10, 11, 12, 13,`
- ☑ **`10, 11, 6, 13,`**
- ☐ `2, 11, 6, 13,`
- ☐ `2, 11, 12, 13,`

2B. What would appear in the terminal after running the line marked as `Q2B`?

- ☐ `10, 11, 6, 13,`
- ☐ `2, 11, 6, 13,`
- ☐ `2, 4, 6, 13,`
- ☑ **`2, 4, 6, 8,`**

Problem 7. Implement a method, **`sumOfEvenColumns()`**, that takes as input a 2-dimensional array of integers and returns the sum of all numbers in the **even-index columns**. For example, consider the following array:

4	7	-8	0	10
-20	2	1	1	68
2	-10	9	-19	16

The method should return: 4 - 20 + 2 - 8 + 1 + 9 + 10 + 68 + 16.

```
// No need to write javadoc

public static int sumOfEvenColumns(int[][] a) {
    int sum = 0;

    for (int row = 0; row < a.length; row++) {
        for (int col = 0; col < a[0].length; col += 2) {
            sum += a[row][col];
        }
    }

    return sum;
}
```

Problem 8. In this problem, we ask you to draw memory diagrams representing the program at various stages of its execution.

```
public class InterestingCode {

  public static void mystery1(String[] arr) {
      arr[2] = "Taro";
      arr[3] = arr[1];
  }

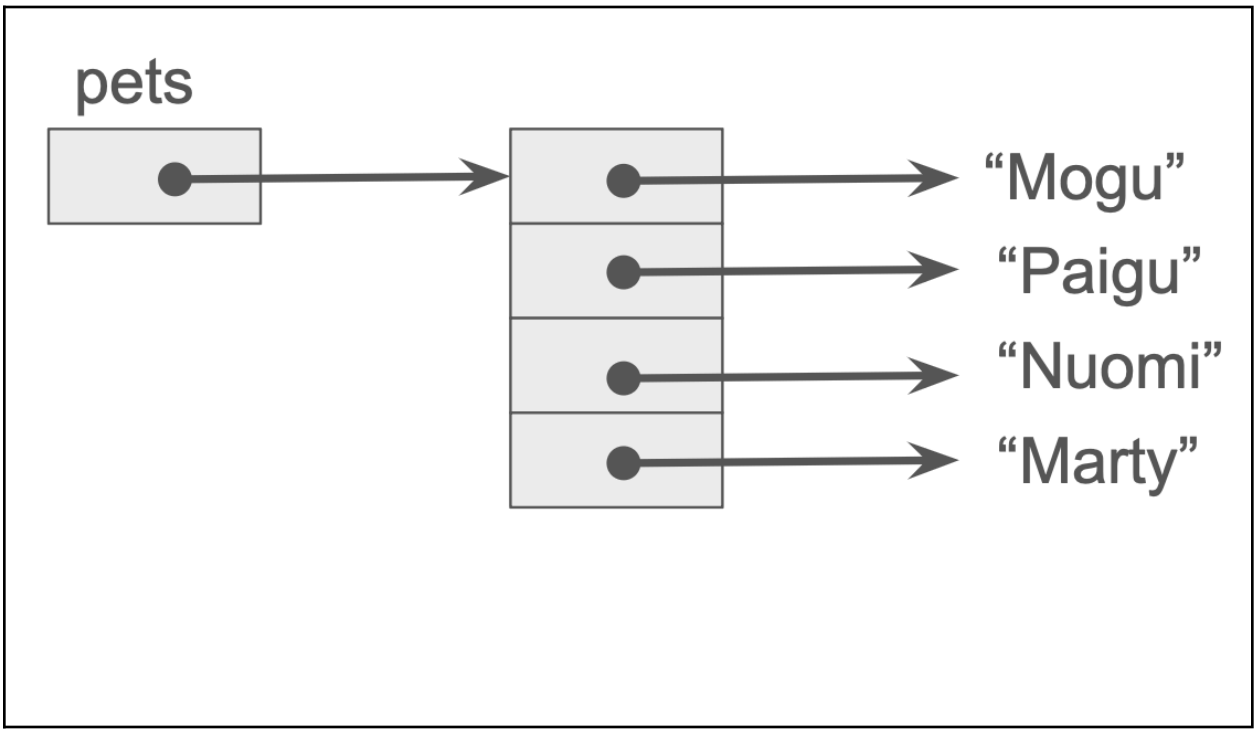
  public static void mystery2(String[] arr) {
      arr[0] = arr[1];
  }

  public static void main(String[] args) {
    String[] pets = {"Mogu", "Paigu", "Nuomi", "Marty"}; // Q2A

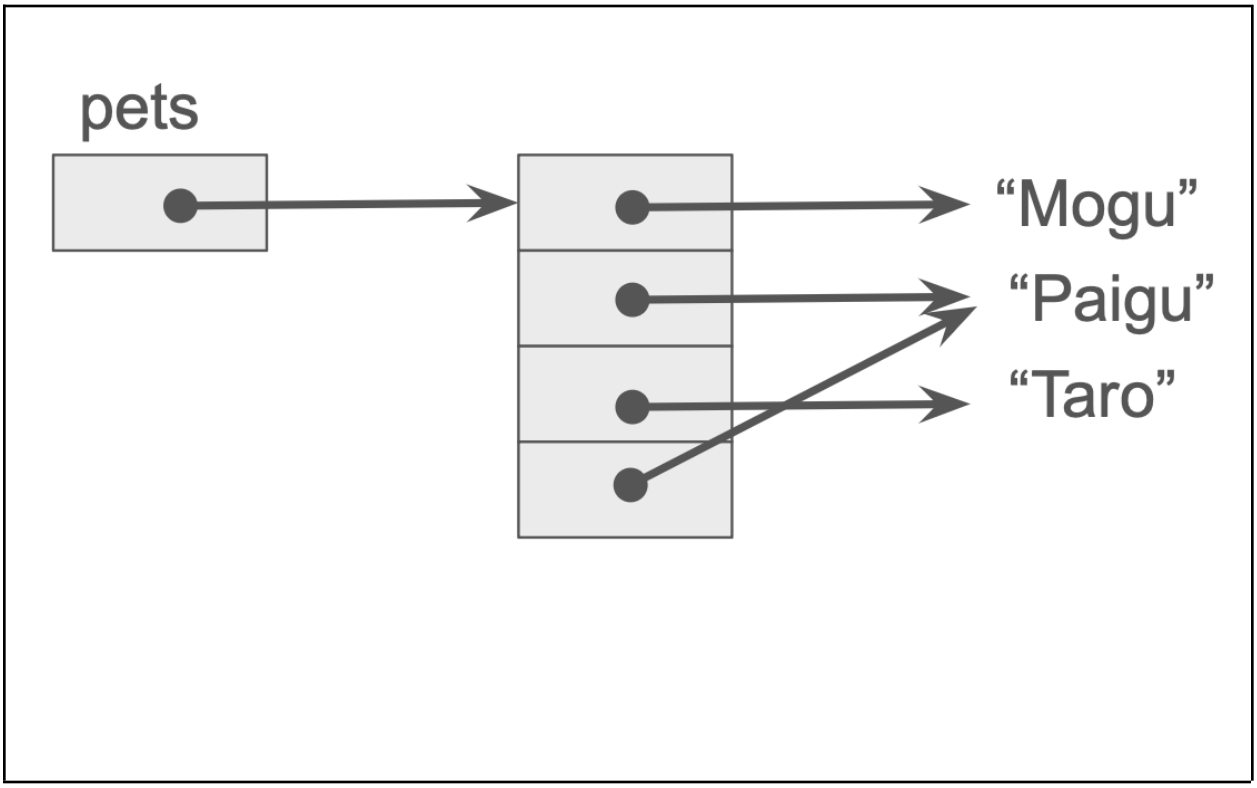
    mystery1(pets); // Q2B

    mystery2(pets); // Q2C
  }
}
```

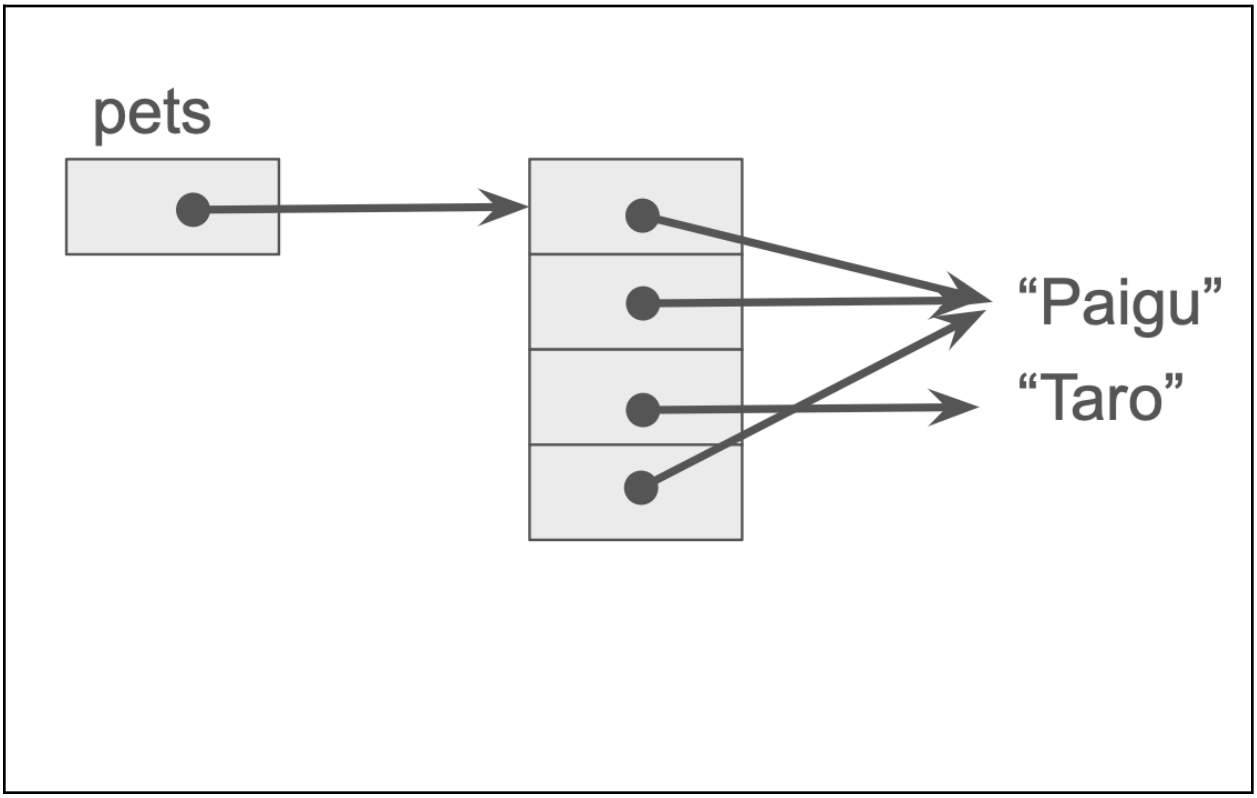
8A. Draw a memory diagram depicting `pets` after line `Q2A` finishes.



8B. Draw a memory diagram depicting `pets` after line `Q2B` finishes.



8C. Draw a memory diagram depicting `pets` after line `Q2C` finishes.



Problem 9. The CS department is implementing a program to help manage courses in the department. In their program, they've created a `Curriculum` class that maintains an array of `Course` objects:

```
public class Course {
    private String name;
    private int number;

    public Course(String name, int number) {
        this.name name;
        this.number = number;
    }

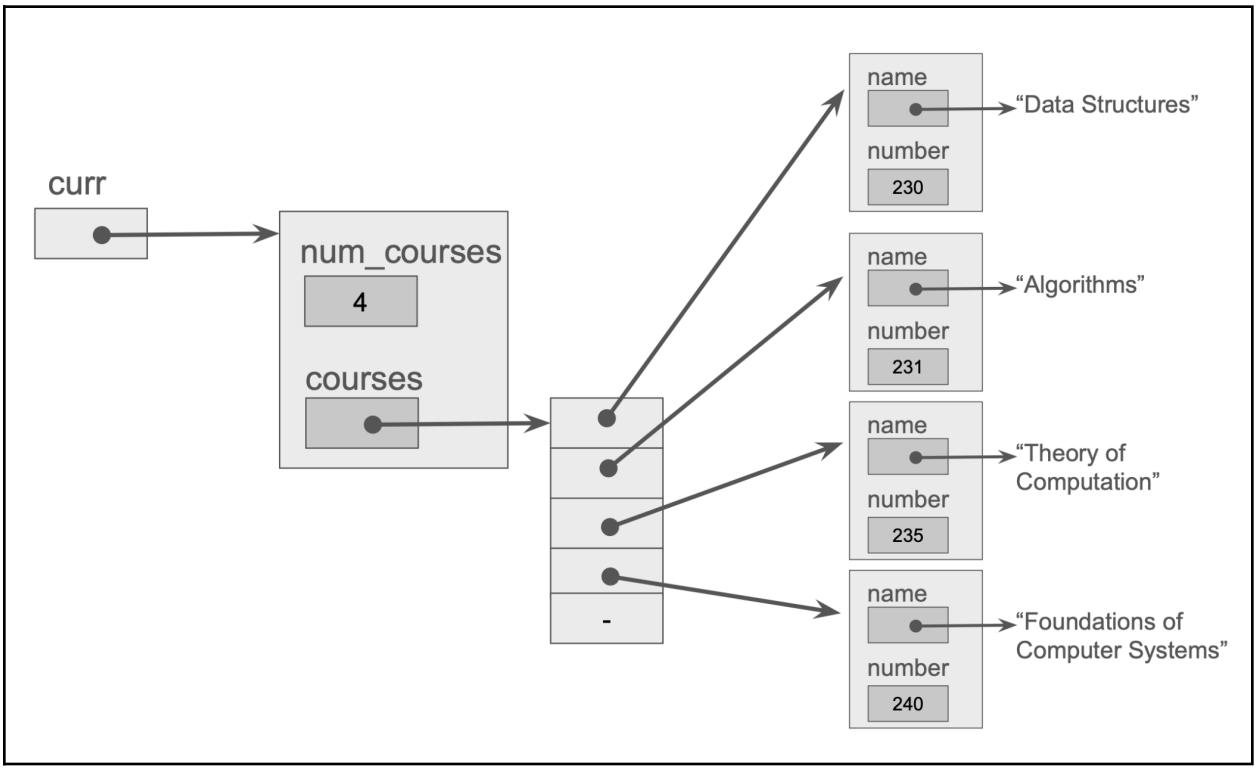
    // Assume this class implements getters and setters
    // as well as an equals method, omitted here for brevity
}

public class Curriculum {
     private int num_courses; //counts the number of courses
     private Course[] courses; // stores the courses

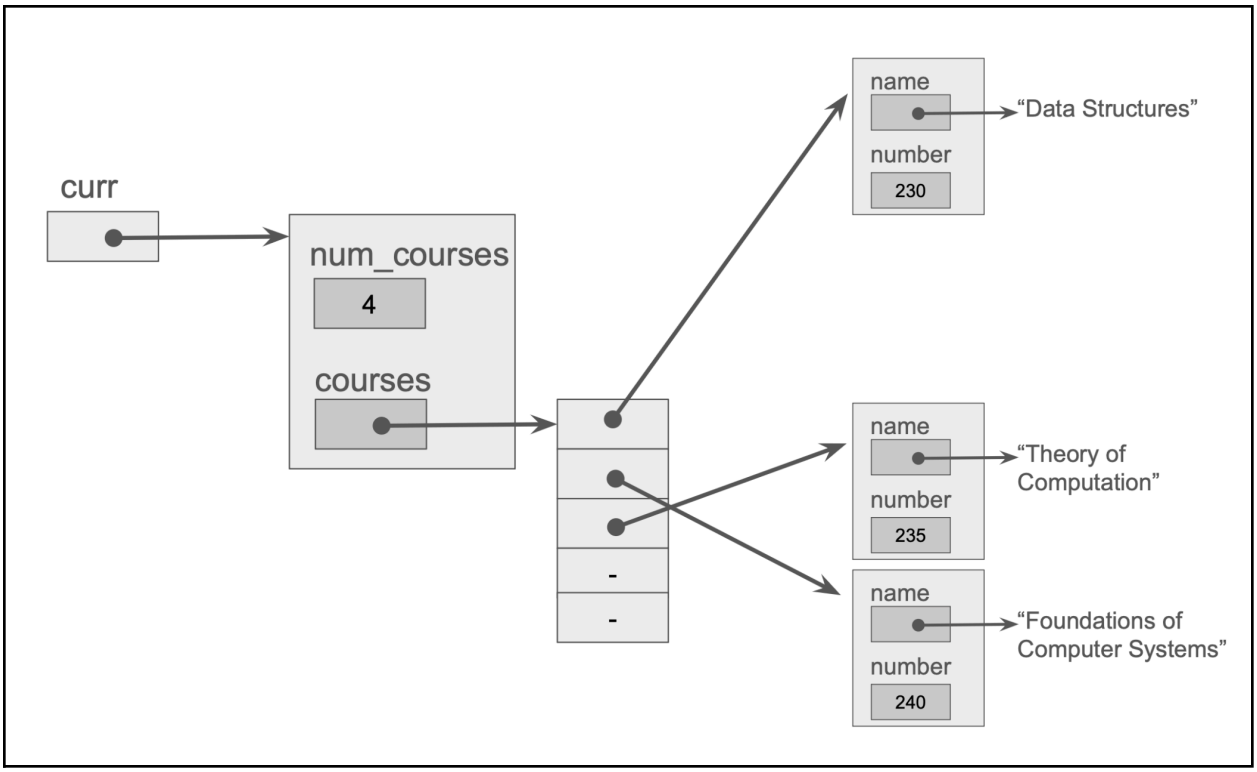
     public Curriculum() {
          this.courses = new Course[5];
          this.num_courses = 0;
     }

     …
}
```

9A. Draw a memory diagram for an instance of the `Curriculum` class that contains 4 courses.



9B. Draw a memory diagram for your instance from part A after the course at *index* 1 has been removed. Make sure there are no "holes" on your array. (You can abbreviate the course object memory diagram to save space).



9C. Implement a method `removeCourse()`, that takes in a `Course` and removes it from the `Curriculum`. Make sure there are no "holes" in the array after removal.

```
// No need to add javadoc


public void removeCourse(Course c) {
    int index = -1;
    for (int i = 0; i < this.num_courses; i++) {
        if (this.courses[i].equals(c)) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        return;
    }

    this.num_courses--;
    this.courses[index] = this.courses[this.num_courses];
    this.courses[this.num_courses] = null;
}
```

Problem 10. You are hired to write a lottery simulation. In this lottery, there are 50 tickets numbered from 1 to 50, and `numWinners` winners are drawn at random. In this problem, you are asked to implement a Java class called `Lottery` that performs the following tasks:

1. `generateWinningTicketNumbers()`: The method should generate `numWinners` **unique** random ticket numbers between 1 and 50 inclusive. These random numbers are then stored as part of the class's attributes. You can assume `numWinners < 50`.
2. `checkIsWinner()`: This method takes in a ticket number as input and returns true if that ticket number is among the winning ones, false otherwise.

Below is a skeleton for the `Lottery` class:

```
import java.util.Random;

public class Lottery {
    private int[] winningNumbers;

    public Lottery(int numWinners) {
        this.winningNumbers = new int[numWinners];
    }
    …
}
```

10A. Should the `winningNumbers` instance variable have a setter method? Why or why not?

It should not, since we don't want a client of the `Lottery` class to be able to set the winning numbers as they wish. The winning numbers should be set **only** using our method, `generateWinningTicketNumbers`.

10B. Implement `checkIsWinner()`. **No need** to write javadoc.

```
public boolean checkIsWinner(int ticket) {
    for (int i = 0; i < winningNumbers.length; i++) {
        if (this.winningNumbers[i] == ticket) {
            return true;
        }
    }

    return false;
}
```

10C. Write a `main()` method in which you instantiate a new lottery, generate 5 winning tickets, and then check if your ticket, the lucky number 42, is a winning ticket, and print the result. **No need** to write javadoc.

```
public static void main(String[] args) {
    Lottery l = new Lottery(5);
    l.generateWinningTicketNumbers();
    if (l.checkIsWinner(42)) {
        System.out.println("Congratulations!");
    } else {
        System.out.println("Sorry, maybe next time…");
    }
}
```

10D. Implement `generateWinningTicketNumbers()`. Note that the numbers should be **unique**. Hint: can you use any of the methods you've already implemented? **No need** to write javadoc.

```
public void generateWinningTicketNumbers() {
    Random generator = new Random();
    for (int i = 0; i < this.winningNumbers.length; i++) {
        int newTicket;

        do {
            newTicket = generator.nextInt(50) + 1;
        } while (this.checkIsWinner(newTicket));

        this.winningNumbers[i] = newTicket;
    }
}
```

Reference Materials

---

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Selected methods summary:

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(**String** anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(**String** str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(**String** str) Concatenates the specified string to the end of this string.
boolean	**contains**(**CharSequence** s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(**CharSequence** cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(**StringBuffer** sb) Compares this string to the specified StringBuffer.
static **String**	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static **String**	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	**endsWith**(**String** suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(**Object** anObject) Compares this string to the specified object.
boolean	**equalsIgnoreCase**(**String** anotherString) Compares this String to another String, ignoring case considerations.

Java.util

Class Random

java.lang.Object
  java.util.Random

All Implemented Interfaces:

Serializable

An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula.

Constructor Summary

Constructor and Description
Random() Creates a new random number generator.
Random(long seed) Creates a new random number generator using a single long seed.

Selected methods summary:

protected int	**next**(int bits) Generates the next pseudorandom number.
boolean	**nextBoolean**() Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.
void	**nextBytes**(byte[] bytes) Generates random bytes and places them into a user-supplied byte array.
double	**nextDouble**() Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.
float	**nextFloat**() Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.
double	**nextGaussian**() Returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
int	**nextInt**() Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.
int	**nextInt**(int bound) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.