

CMSC 131 Quiz 6 Worksheet

The sixth Quiz of the course will be on **Wednesday** April 6 in lab session. The following list provides more information about the quiz:

- You will have 15 minutes to complete the quiz.
- It will be a written quiz (not using any computer).
- It will be closed-book, closed-notes, and no calculator is allowed.
- Answers must be neat and legible. We recommend that you use pencil and eraser.
- The quiz will be based on the exercises you will find below. The quiz will ask you to write a class for a particular application.

The following exercises cover the material to be included in Quiz #6. Solutions to these exercises will not be provided, but you are welcome to discuss your solutions with TAs and instructors during office hours.

When asked for a “piece of code” you do not need to provide an entire class definition or even an entire method. Just present the Java statements and any variable declarations, as needed, to make your solution clear.

Problem 1

Write a method that implements the functionality of `Double.parseDouble`. That is, a method that transforms a string into a double. Examples include “14.234”, “-13”, “+.234”, and “0.0100”.

Problem 2

A Stack is a data structure frequently used in Computer Science. In this structure elements are inserted and removed from the same end (similar to a stack of clean plates in a cafeteria). Implement a class called `MyStack` which represents a Stack of String objects. The specifications associated with the class are:

Instance variable

```
String[] array; // Array used to represent the stack.  
                // Feel free to add any other variables you may need.
```

Methods

- Constructor – Creates a stack with no elements. Hint: create an array with size 0.
- push – Adds an element to the top of the stack. It takes as parameter a reference to a string. The method will create a duplicate of the string and will add the copy to the stack. The array must be resized to accommodate the new element.

- pop – Removes an element from the top of the stack. It takes no parameters. The method removes the string at the top of the stack and resizes the array. The method returns a reference to the string that was removed.
- top – Returns a reference to the element at the top of the stack without removing the element.
- empty – Returns true if the stack is empty and false otherwise.

The following illustrates how to use the class MyStack:

```
MyStack myStack = new MyStack();
myStack.push("Hello");
myStack.push("Everybody");
System.out.println(myStack.top()); // It will print "Hello".
myStack.pop();
System.out.println(myStack.top()); // It will print "Everybody".
```

Problem 3

A student class has the following definition:

```
public class Student {
    private String name;
    private int id;

    public Student(String sname, int sid) {
        name = sname;
        id = sid;
    }

    public Student(Student student) {
        name = new String(student.name);
        id = student.id;
    }

    public int getId() {
        return id;
    }

    public String toString() {
        return name + " " + id;
    }
}
```

Define a class called Roster which has the following specifications:

Instance variable

```
Student[] array; // array used to represent the roster
                // feel free to add any other variables you may need
```

Methods

- Constructor – Creates a roster with no students. Hint: Create an array with size 0.
- add – Takes a student reference as parameter. It adds a copy of the student to the array so that the array is **sorted by student id**.
- remove – The method takes an id as parameter. The students with that id will be removed from the array. The array must be resized after the element is removed.
- getStudentsDeep – Returns a deep copy of the students' array.
- getStudentsShallow – Returns a shallow copy of the students' array.
- getStudentsHalf – Returns a half-deep copy of the students' array.
- toString – It will return a string with the students in the rosters or the string “NO STUDENTS”.

Problem 4

Write a method that initializes an integer array with the even random numbers that are present in an integer range. The prototype for the method will be:

```
public static int[] initRandomEven(int startRange, int endRange);
```

For example, given the following call:

```
int[] data = initRandomEven(2, 9);
```

the data array could have the elements: {4, 8, 2, 6}. Note that all the even numbers in the range must appear and cannot appear twice. Your method must be as efficient as possible. Use the Random class nextInt() method to generate your random values.

Problem 5

Write a static method that, given two strings x and y, determines whether x is a substring of y. For example, "and" is a substring of "random".

Problem 6

Note: This problem was last semester's quiz. The solution has been provided at the end.

Problem Description

A class called **WaitList** represents a set of students waiting to get into a course. The class relies on a Student class defined as follows:

```
/* YOU MAY NOT MODIFY THIS CLASS */
public class Student {
    private String name;
    private int id;

    public Student(String sname, int sid) {
        name = sname;
        id = sid;
    }

    public Student(Student student) {
        name = new String(student.name);
        id = student.id;
    }

    public int getId() {
        return id;
    }

    public String toString() {
        return name + " " + id;
    }
}
```

For this quiz you will implement the **WaitList** class. The class specifications are:

Instance Variable

```
Student[] array; // Array that represents the list of students
```

Methods

- **Constructor** – Creates a waitlist with no students.
- **add** – This method takes as parameter a reference to the student to be added. The method makes a copy of the student object and adds the copy to the end of the array representing the waitlist. The size of the array will be increased by one.
- **toString** – Returns a string with the name and id of every student in the waitlist. Information about each student will appear on a line by itself.

One Possible Solution

```
public class WaitList {
    private Student[] array;

    public WaitList() {
        array = new Student[0];
    }

    public void add(Student student) {
        Student[] newArray = new Student[array.length+1];
        for (int i=0; i<array.length; i++) {
            newArray[i] = array[i];
        }
        newArray[array.length] = new Student(student);
        array = newArray;
    }

    public String toString() {
        String result = "";
        for (int i=0; i<array.length; i++) {
            result += array[i] + "\n";
        }

        return result;
    }

    public static void main(String[] args) {
        WaitList waitList = new WaitList();

        waitList.add(new Student("John", 23));
        waitList.add(new Student("Mary", 24));
        System.out.println(waitList);
    }
}
```