BASIC PROGRAMMING EXERCISES (CONDITIONAL STATEMENTS, LOOPS, METHODS, AND ARRAYS)

COMP-202A, Fall 2009, All Sections

INSTRUCTIONS

- Attempt each question on paper before trying to implement it in a Java program.
- Attempt to implement the solution to a problem only once you think you have a solution written on paper.
- Every loop **MUST** terminate as soon as it can; for example, if a question asks you to write a method which determines whether a given value occurs in a given array, the loop **MUST** terminate as soon as it finds a value in the given array which is equal to the given value, without looking at the rest of the values. However, you **MUST NOT** use the `break` statement to exit loops.
- Each question asks you to write a class which defines one method. After completing each question, add a `main()` method to the class you wrote for that question. This `main()` method should ask the user to enter values for each of the parameters of the method you wrote for that question, and read these values from the keyboard. Then, have your `main()` method call the method you wrote for that question, pass the values the user entered as parameters to this method, and display the results this method returns.
- To read an array of `char` from the keyboard, use

  ```
  keyboard.nextLine().toCharArray();
  ```

  where `keyboard` is the name of the `Scanner` variable you initialized to read from the keyboard. Likewise, to read a single `char` from the keyboard, use

  ```
  keyboard.nextLine().charAt(0);
  ```

  where, again, `keyboard` is the name of the `Scanner` variable you initialized to read from the keyboard.

PROBLEMS

1. Write a Java class called `Factorial`. This class defines a method called `factorial()` which takes as its only parameter an `int` called `n`, and returns an `int` representing the factorial of `n`. The factorial of an integer n, denoted $n!$, is defined as $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$ (but note that $0!$ is 1). You **MAY** assume that `n` is a non-negative integer.
2. Write a Java class called `Fibonacci`. This class defines a method called `fibonacci()` which takes as its only parameter an `int` called `n`, and returns an `int` representing the $\texttt{n}^{\text{th}}$ Fibonacci number. The

n^{th} Fibonacci number, denoted f_n, is defined as follows:

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

You **MAY** assume that `n` is a non-negative integer.

3. Write a Java class called `PrimalityChecker`. This class defines a method called `isPrime()`, which takes as its only parameter an `int` called `n`, and returns a `boolean` which is `true` if and only if `n` is a prime number, `false` otherwise. A prime number is an integer which cannot be divided evenly by any integer except 1 and itself. You **MAY** assume that `n` is a non-negative integer.

4. Write a Java class called `GoldbackChecker`. This class defines a method called `checkGoldbach()`, which takes as its only parameter an `int` called `n`, and returns an array of `int`. This method finds two prime numbers whose sum is equal to `n`, and returns these two prime numbers in an array of `int`s of length 2; if two such primes cannot be found, `n` is less than 4, or `n` is odd, your method should return `null`. Your `checkGoldbach()` method **MAY** call the `isPrime()` method you wrote for a previous exercise. Note that every even `int` value greater than `4` can be expressed as the sum of two prime numbers.

5. Write a Java class called `Power`. This class defines a method called `power()` which takes as parameters a `double` called `base` as well as an `int` called `exponent`, and returns a `double` representing the value of `base` raised to the power `exponent`. You **MAY** assume that `exponent` is a non-negative integer value, and that `base` and `exponent` are not both `0`. You **MUST** write the computation yourself; in other words, you **MUST NOT** use the `Math.pow()` method.

6. Write a Java class called `CountDigits`. This class defines a method called `countDigits()` which takes as its only paramter an `int` called `n`, and returns an `int` representing the number of digits in `n`. You **MAY** assume that `n` is a positive integer.

7. Write a Java class called `Contains`. This class defines a method called `contains()`, which takes as parameters an array of `int` called `a` as well as an `int` called `x`, and returns a `boolean` which is `true` if and only if `x` occurs in `a`, `false` otherwise.

8. Write a Java class called `ContainsInRange`. This class defines a method called `contains()`, which takes as parameters an array of `int` called `a`, an `int` called `x`, an `int` called `start`, as well as an `int` called `end`, and returns a `boolean` which is `true` if and only if `x` occurs in `a` at a position which is greater than or equal to `start`, and less than `end`. You **MAY** assume that `start` is less than or equal to `end`, and that both `start` and `end` are greater than or equal to 0, and less than or equal to `a.length`.

9. Write a Java class called `CheckDuplicates`. This class defines a method called `checkDuplicates()`, which takes as its only parameter an array of `int` called `a`, and returns a `boolean` which is `true` if and only if there exists at least one value in `a` which occurs more than once. Your `checkDuplicates()` method **MAY** call the `contains()` method you wrote for a previous exercise.

10. Write a Java class called `Maximum`. This class defines a method called `maximum()`, which takes as its only parameter an array of `int` called `a`, and returns an `int` representing the maximum value which occurs in the array.

11. Write a Java class called `CountOccurrences`. This class defines a method called `countOccurrences()`, which takes as parameters an array of `int` called `a` as well as a `int` called `x`, and returns an `int` representing the number of elements of `a` which are equal to `x`.

12. Write a Java class called `MostOftenOccurring`. This class defines a method called `occursMostOften()`, which takes as its only parameter an array of `int` called `a` and returns an `int` representing the value which occurs most often in `a`. If there is a tie, display the value which occurs first in `a` among those that occur most often. Your `occursMostOften()` method **MAY** call the `countOccurrences()` method you wrote for the previous exercise.

13. Write a Java class called `ToUpperCase`. This class defines a method called `toUpperCase()`, which takes as its only parameter an array of `char` called `s`, and returns an array of `char`. The array that the method returns contains exactly the same characters as `s` in the same order, except that all lower-case letters occurring in `s` are replaced by their upper-case equivalents in the array returned by the method.

14. Write a Java class called `OccursHere`. This class defines a method called `occursHere()`, which takes as parameters two arrays of `char` called `superString` and `subString`, as well as a positive integer `position`, and returns a `boolean`. This `boolean` is `true` if and only if all the characters in `subString` occur consecutively in `superString` starting at position `position`, in the same order as the one in which they appear in `subString`, `false` otherwise.

15. Write a Java class called `IndexOf`. This class defines a method called `indexOf()`, which takes as parameters two arrays of `char` called `superString` and `subString`, and returns an `int`. This `int` represents the smallest index within `superString` at which the characters of `subString` appear consecutively in `superString`, in the same order as the one in which they appear in `subString`. This method returns `-1` if the characters of `subString` never appear consecutively in `superString`. Your `indexOf()` method **MAY** call the `occursHere()` method you wrote for a previous exercise.

16. Write a Java class called `Substring`. This class defines a method called `substring()`, which takes as parameters an array of `char` called `s`, an `int` called `beginIndex`, as well as an `int` called `endIndex`, and returns a new array of `char` which consists only of the characters of `s` between positions `beginIndex` (inclusive) and `endIndex` (exclusive). You **MAY** assume that `beginIndex` is greater than or equal to 0, that `endIndex` is less than or equal to the length of `s`, and that `beginIndex` is less than or equal to `endIndex`.

17. Write a Java class called `RemoveAll`. This class defines a method called `removeAll()`, which takes as parameters an array of `int` called `a` as well as an `int` called `x`, and returns a new array of `int` which consists of all the elements in `a`, in the order in which they appear in `a`, but with all occurrences of `x` removed. The size of the array returned by this method **MUST** be equal to the number of elements it contains; that is, it must be equal to `a.length - n`, where `n` is the number of occurrences of `x` in `a`. Your `removeAll()` method **MAY** call the `countOccurrences()` method you wrote for a previous exercise.

18. Write a Java class called `NumberConsecutive`. This class defines a method called `countOccurrences()` that takes as parameters an array of `int` called `a`, an `int` called `x`, as well as an `int` called `position`, and returns an `int` representing the number of occurrences of `x` in `a` starting at position `position`. For example, if `a` is {`1, 2, 2, 2, 2, 2, 1, 1, 1`}, `x` is 2, and `position` is `1`, then your method will return `5`, as there are 5 consecutive occurrences of 2 in `a` starting at position `1`. You **MAY** assume that `position` is greater than or equal to `0` and less than `a.length`

19. Write a Java class called `MaxConsecutive`. This class defines a method called `maxOccurrences()` that takes as parameters an array of `int` called `a` as well as an `int` called `x`, and returns an `int` representing the position in `a` at which the longest series of consecutive occurrences of `x` occurs. If `x` never occurs in `a`, then the method should return `-1`. If there is a tie between two positions, you should return the smallest position. Your `maxOccurrences()` method **MAY** call the `countOccurrences()` method of the `NumberConsecutive` class you wrote for a previous exercise.

20. Write a Java class called `Reverse`. This class defines a method called `reverse()`, which takes as its only parameter an array of `int` called `a`, and returns a new array of `int` which contains all the elements in `a`, but in the reverse order.

21. Write a Java class called `ReverseInPlace`. This class defines a method called `reverse()`, which takes as its only parameter an array of `int` called `a`, and returns `void`. This method changes `a` so that when the method returns, its elements occur in the reverse order.

22. Write a Java class called `Palindrome`. This class defines a method called `isPalindrome()`, which takes as its only parameter an array of `char` called `s`, and returns a `boolean` which is `true` if and only if the characters in `s` form a palindrome. A palindrome is a series of characters which reads the same

forwards and backwards, such as `"laval"` or `"stressed desserts"`.

23. Write a Java class called `Equals`. This class defines a method called `equals()`, which takes as parameters two arrays of `int` called `a1` and `a2`, and returns a `boolean` which is `true` if and only if `a1` and `a2` are equal. For the purposes of this question, two arrays of `int` are equal if they are of the same length and the same values appear in the same positions.

24. Write a Java class called `Subset`. This class defines a method called `isSubset()`, which takes as parameters two arrays of `int` called `big` and `small`, and returns a `boolean` which is `true` if and only if all the elements in `small` also occur in `big`. Your `isSubset()` method **MAY** call the `contains()` method you wrote for a previous exercise.

25. Write a Java class called `CountWords`. This class defines a methods called `countWords()`, which takes as its only parameter an array of `char` called `s`, and returns an `int` representing the words in `s`. A word is delimited by one or many space characters.