proprep

Workbook

Table of Contents

Further Programming Concepts

Arrays

Questions

1) Create a new project, and within it a class called arraysApp. Inside the class create a method called outputOddNumbers()
   a. Inside the method, declare an array of bytes with the following elements:
      i. 1, 2, 4, 5, 10, 11, 12, 14, 17, 20, 21
   b. Using a for loop, iterate through the array outputting to the screen each element of the array on a new line
   c. Modify the code inside the loop so that numbers are output only if they are odd

2) Create another method in the same class, called topThree()
   a. Inside the method, declare an array of bytes called scores with the following elements:
      i. 55, 75, 12, 67, 23, 95, 23, 61, 82, 43, 37
   b. Declare a second, empty array of bytes, called results with a size of 3
   c. Use a loop to iterate through the scores array, comparing each element value with what is stored in results[0]. If the current value during iteration is higher than what is already stored in results[0], replace results[0] with the current value
   d. results[0] should now contain the largest value in the array. After the loop, output to the screen the message "The highest score is:" followed by the contents of results[0]
   e. Add a second loop after the above code which finds the second highest value by comparing the current element with results[1], AND results[0]. If it is higher than results[1] but lower than results[0], replace results[1] with the current value.
   f. results[1] should now contain the second largest value in the array. Output to the screen the message "The 2nd highest score is:" followed by the contents of results[1]
   g. Add a final loop after all previous code, which finds the third highest value and stores it into results[2]
   h. results[2] should now contain the third largest value in the array. Output to the screen the message "The third highest score is:" followed by the contents of results[2]
   i. Change the first element in the scores array from 55 to 97 and check your program still works, with new numbers appearing as the top 3 scores

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

2

© All rights in this workbook reserved to proprep™

3) Create a new method called `common()`

a. Declare two arrays, `arrayA` and `arrayB`, containing the following values

    i.   `arrayA`: 4, 7, 10, 16, 25, 28, 40, 53, 79, 92, 111

    ii. `arrayB`: 2, 5, 6, 10, 12, 19, 28, 33, 44, 51, 87, 92

b. Use a loop to iterate through `arrayA`, checking whether each element is present in `arrayB`. When an element is found in both arrays, output the value alongside the message "common value found".

4) Create a method called `lookup()`

a. Declare two arrays:

    i.   an integer array called `gradeBoundaries`, with initial values: 35,43,50,65,77

    ii. a char array called `grades`, with initial values: 'U', 'E', 'D', 'C', 'B', 'A'

b. Declare three variables:

    i.   an integer called `mark`, with an initial value of 22

    ii.   a boolean called `found`, with an initial value of false

    iii. an integer called `position`, with an initial value of 0

c. Create a do…while loop which ends when `position` is equal to the length of the `gradeBoundaries` array or when `found` is equal to true

d. Inside the do…while loop, see if the value of `mark` is less than the value of the element in the `gradeBoundaries` array currently being pointed to by the index variable `position`

    i.   If it is, assign the value `true` to the `found` variable

    ii. If it is not, increment the `position` variable

e. Outside the loop print the element in the `grades` array that is pointed to by the index variable `position`

f. Test the code outputs the character 'U'

g. Change the value of `marks` to 43, and 77 to confirm grades of D and A are output respectively.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

3

© All rights in this workbook reserved to proprep™

5) Below is pseudo code for an algorithm called a Bubble sort:

numbers= [9, 5, 4, 15, 3, 8, 11]
numItems = len (numbers)
for i = 0 to numItems -2
    for j = 0 to (numItems - i - 2)
        if (numbers[j] > numbers[j+1]) then
            temp = numbers [j]
            numbers[j] = numbers[j + 1]
            numbers[j + 1] = temp
        end if
    next j
    output (numbers)
    output ("\n")
next i

a. Turn this pseudo code into a Java method called bubbleSort
b. The line `output(numbers)` will need to print the entire array on one line using a loop, and the line `output ("\n")` will output a new line character
c. Change the numbers array by adding the number 2 as the last element after the number 11, and test the bubbleSort method again. How many times does the number 2 get swapped?

6) Create a method called twoD()
a. Declare a two dimensional integer array called `grid`, with 5 rows and 7 columns
b. Using a pair of nested loops, assign to each element of the array a value according to this formula: `element value = column index + (row index * 7)`
c. After the nested loops in part b which write values to the array, create another pair of nested loops.
    i. In the inner loop output each element from the array on the same line using `System.out.print`
    ii. After each element is output, output the tab character `"\t"` again on the same line using `System.out.print`
    iii. After the inner loop, but inside the outer loop, output a new line character
d. Test your program outputs the numbers 0 to 34
e. Make a change to the program so it outputs the numbers 1 to 35 instead
f. Change the program again, so that when the initial size of the array is changed, from 5 rows, 7 columns to 8 rows, 5 columns, the numbers 1 to 40 are printed.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

4

© All rights in this workbook reserved to proprep™

7) Create a method called `transpose()`

a. Declare a 2D char array initialised with the following values:
   {'h', 'i', 's', 't', 's'}, {'e', 's', 'e', ' ', 'a'},
   {'r', ' ', 'c', 'm', 'g'}, {'e', 'a', 'r', 'e', 'e'},   {' ', ' ', 'e', 's', '.'}

b. Declare a pair of integer variables, `row` and `column`, both initialised to `0`

c. Create a pair of nested loops, with the inner loop iterating `column`, and the outer loop iterating `row`

d. Inside the inner loop, output the element pointed to by `row` and `column`, using System.out.print so all characters appear on the same line

e. Test your program outputs the characters as seen above in part a in the order that they appear in the array by row

f. Swap the order of the loops created in c) so that column is iterated in the outer loop, and row is iterated in the inner loop. Test again, can you explain the output you see?

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

5

© All rights in this workbook reserved to proprep™

Strings and String Processing

Questions

1) Create a new project and within it a class called `stringApp`

a. Create a new method called `reverseString()`

b. Declare a string variable and initialise it with the literal string `"stressed"`

c. Use a loop to iterate through each character starting from the back of the string, outputting each character as it is encountered, all on one line using `System.out.print`

d. Test the program by running it, if it works correctly it will print the input string reversed. Try some other input strings like `"spam"` and `"peels"`, all three should output familiar words.

2) Create a new method called `robot()`

a. Declare:

    i. a string variable called `commands` and initialise it with the literal string `"FFLFFRFFLFFFFRRFFH"`

    ii. a char variable called `currentCommand` with no initial value

b. Using a loop to iterate through each character in the string, extract the current character into `currentCommand`

c. Inside the loop, display a message as follows depending on the contents of `currentCommand`:

    i. F – Forward

    ii. R – Right turn 90 degrees

    iii. L – Left turn 90 degrees

    iv. H – Halt

d. Test your code produces the expected output

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

6

© All rights in this workbook reserved to proprep™

3) Create a new method called `extractInitials()`

a. Declare a string variable `fullName` containing the initial value `"Robert Womack"`

b. Declare an integer variable `position`, with 0 as the initial value

c. Declare a string variable `initials` with an empty string "" as the initial value

d. Add the first character of `fullName` to the `initials` variable using the concatenation operator

e. Use a loop to iterate through each character in the string

   i. each time a space or a hyphen is found, extract the next character and add it to `initials`

   ii. Keep going until no more characters are left in the `fullName` string

f. Output the contents of the `fullName` and `initials` strings separated by a suitable character or message

g. Test the program with the following initial values for `fullName`

   i. "Robert Womack"

   ii. "Penny Martha Lane"

   iii. "David Michael-Cox"

   iv. "George Frederick Ernest Albert Saxe-Coburg-Gotha"

4) Create a new method called `parseAssembly()`

a. Declare:

   i. a string variable with the initial value `"LDA 101 STA 100 NOP LDA 102 NOP HLT"`

   ii. a string variable called `strMatch` with the initial value "LDA"

b. Split the input string on spaces, into a new array so that each substring is stored in a new array element. For example "LDA" will be stored in array element 0, `101` in array element 1 etc.

c. Using a loop, inspect each element in the array to see if it contains the same value as `strMatch`. If it does output the following all on the same line:

   i. the contents of `strMatch`

   ii. a space character

   iii. the contents of the next element in the array

   iv. a new line character

d. Test your program outputs LDA 101 and LDA 102 and nothing else

e. Change the initial value of `strMatch` to `"STA"` and check the output is STA 100

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

7

© All rights in this workbook reserved to proprep™

5) Create a method called `isGraphicFileType()`

a. Declare:

    i. a String array called `files` containing the following values:
"MyThesis.doc", "profilePic.jpg", "budget.xls", "favjpg.pdf", "funny.gif", "temp", "logo.png", "photo.jpeg"

    ii. an String variable called `filetype`

    iii. an integer variable called `position`

b. Iterate through the elements of the array outputting the value of each element without adding a newline character at the end

c. Inside the loop below the output line in part b,

    i. use a suitable string method to find whether the string contains a "." and to store the position where it is found into `position`

    ii. If the "." was found, extract a substring into `filetype` so `filetype` contains the characters after the "."

    iii. Compare the substring with the file extensions "jpg", "jpeg", "gif" and "png". If there is a match, output `"\t * is a graphic file type *"` without a newline

    iv. Output a newline character inside the loop, but outside the selection structure in parts ii-iii

d. Test the method outputs all the filenames, with an indication that the file is a graphic file type for only the relevant files.

e. Change the file type for "logo.png" to "logo.pdf" and confirm the difference in output.

6) Create a method called `isPalindrome()`

a. Declare a string variable called `inString` with the initial value "Did Hannah see bees? Hannah did."

b. Create two more string variables called `rawString` and `reverseRawString`, both with initial values of ""

c. Copy the contents of `inString` to `rawString` using regular expressions and Java string functions so that:

    i. any upper case letters in `rawString` are forced to lower case

    ii. only letters are in `rawString` and all other characters are ignored

d. Using a loop, output the `rawString` characters into `reverseRawString` in reverse order, i.e. the first character in `reverseRawString` will be the last character in `rawString`

e. Compare `rawString` to `reverseRawString`

    i. if they are equal output `inString` followed by the message " is a palindrome"

    ii. if they are not equal output `inString` followed by the message " is not a palindrome"

f. Test your program works, it should find that the input string is a palindrome. Remove one letter from the initial value to see if it now finds it is not a palindrome

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

8

© All rights in this workbook reserved to proprep™

7) Create a new method called `findSongs()`

a. Declare:

    i. a string variable called `sourceString` with an initial value as follows: "<album>Greatest<song trackid=1>Song for You</song><song trackid=2>Sad Song</song><song>Another Song</song></album>"

    ii. a string array called `songs` with 10 uninitialised elements

    iii. integer variables called `index`, `start` and `end`, initialised to 0

    iv. a boolean variable called `finished`, initialised to `false`

b. Using a loop which terminates when `finished` is `true`, parse `sourceString` to extract the song name between `<song>` and `</song>`. Note some songs have extra text between `<song and >`

    i. Use `start` and end to store the beginning and end of the substring to be extracted

    ii. each substring extracted should be stored into the next element in the `songs` array using `index`

    iii. when there are no more songs to be found, set `finished to true`

c. Outside the parse loop, use another loop to output the contents of the `songs` array, with each element appearing on a new line in the output.

d. Test your method stores three strings into the `songs` array

e. Using the format "`<song>`*name of song*`</song>`" add two more songs at the end of the initial value of `sourceString` and test whether these additional songs appear in the `songs` array and therefore the output

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

9

© All rights in this workbook reserved to proprep™

Functions

Questions

1) Create a new project and within it a class called `functionsApp`

a. Within your class create a method called `checkLengthOK` that accepts one parameter, which is a `String`, and returns a `boolean`

b. The method should return `true` if the string passed in is 8 or more characters long, otherwise it should return `false`

c. Test your method works with the input string `"computer"` returning `true` and input string `"compute"` returning `false`

2) Pythagoras' theorem states we can find the length of a side h in a right angled triangle given the length of the two other sides a and o, according to the formula: $h^2 = a^2 + o^2$

a. Use the BlueJ Codepad to experiment with the Java Math library functions `pow()` and `sqrt()` which allow you to square a number and take the square root of a number.
   i. `Math.pow(3,2)` returns 3 squared which is 9
   ii. `Math.sqrt(25)` returns 5, the square root of 25

b. Create a method that accepts two `doubles` and returns a `double`

c. Using the two input values as a and o, use Pythagoras to calculate and return the length of h by taking the square root of $a^2 + o^2$

d. Test that input arguments of 4 and 3 return the value 5

3) Create a method called `linearSearch` that accepts an array of integers to search and a single integer to search for, and returns an integer

a. Use a loop to iterate through each element in the array

b. If the integer is found in the array, the method returns the position it was found at, if not found it returns -1

c. Test with inputs of:
   i. {1,3,5,7,9} and the value 1
   ii. {1,3,5,7,9} and the value 2
   iii. {2,4,6,8,10} and the value 10

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

10

4) A quadratic equation takes the form $ax^2 + bx + c = 0$

a. The `String.valueOf()` function will convert any number inside the brackets to a string representation. Try this with various sizes of integer and real numbers in the CodePad, e.g
   i. `String.valueOf(123)`
   ii. `String.valueOf(3.14)`

b. Write a method called `quadratic` that takes in three doubles, a, b and c and returns a string

c. Return the value(s) of x as a string using formula $x = (-(b) \pm sqrt(b^2 - 4ac)) / 2a$
   i. Note the ± symbol means there will be two solutions, but if $b^2 - 4ac$ equals zero then both solutions will be the same so we can just return one value
      - If there are two solutions, return both solutions as a string in the format "x = xxx or x = yyy" where xxx and yyy represent the value of x for each of the two solutions
      - If there is only one solution output the solution as a string using the format "x = xxx"
   ii. If $b^2 - 4ac$ is less than zero then return the string "Complex roots not supported"
   iii. If the value of a or c passed in was 0 return the string "Invalid input"

d. Test that
   i. inputs of a = 1, b = -6 and c = 5 give the result "x = 1 or x = 5"
   ii. inputs of a = 2, b = 4 and c = 2 give the result "x = -1"
   iii. inputs of a = 3, b = 5 and c = 3 give the result "Complex roots not supported"
   iv. inputs of any value of b and either a=0 or c=0 give the result "Invalid input"

5) This exercise involves bringing together several methods to solve one overall problem

a. Use the Codepad to experiment with the methods available to check characters
   i. `Character.isLowerCase('a')`
   ii. `Character.isDigit('2')`

b. Use the Codepad to see how a string can be checked for a specific character like so:
   i. `String listofChars="$%^*"`
   ii. `listofChars.contains(String.valueOf('%'))`

c. Create a method called `hasUpperCase` which accepts a string argument and returns a boolean
   i. Loop through each character in the input string looking for upper case letters
   ii. Return `true` only if the string contains at least one character which is an upper case letter, otherwise return `false`

d. Create another method called `hasLowerCase` which returns `true` only if a string passed into it contains at least one character which is a lower case letter

e. Create another method called `hasDigit` which returns `true` only if a string passed into it contains at least one character which is a digit between 0 and 9

f. Create another method called `hasSymbol` which returns `true` only if a string passed into it contains at least one symbol from any of the characters `"!$%^&*().?"`

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

proprep

11

© All rights in this workbook reserved to proprep™

g. Create a final method, `validatePassword` which is passed a string, and returns `true` only if the string contains at least one upper case character, one lower case character, one digit and one symbol. It should use the three methods you created in parts a-d

h. Test with input passwords of:
    i. "Secret7!" – should return true
    ii. "ABCd$" – should return false
    iii. "Weak1" – should return false
    iv. "password1!" – should return false

6) Java can take input from the keyboard using methods in the `java.util` library

a. At the top of the file, above the class definition, add the line:
    i. `import java.util.Scanner;`

b. Create a new method which returns no values, and has no arguments called `passwordChecker`

c. Add the following lines into the `passwordChecker` method:
    i. `Scanner in = new Scanner(System.in);`
    ii. `String password;`

d. Use a do…while loop to repeatedly read a string from the keyboard to store into the `password` variable using:
    i. `System.out.print("Enter a password to check:");`
    ii. `password = in.next();`

e. Still inside the loop, call the `checkLengthOK()` and `validatePassword()` methods using `password` as the argument to both.
    i. If `true` is returned by both methods output the message `"Password valid"`
    ii. Otherwise output the message `"Password invalid"`

f. End the loop when the user types "end" as the password

g. Test your program by running main, and entering:
    i. " Shed7!" – which should be invalid
    ii. "Richest1%" – which should be valid
    iii. "Longest12" – which should be invalid
    iv. "end" which should end the program

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

12

7) For this exercise we will create three methods to check random letters from a given password.

a. Create a new method `getRandomPosition` which takes a string as the argument and returns an integer which is a random number between 0 and the length of the string.
   i. Use the Codepad to see how the `Math.random()` method returns a random number which is greater than 0 and less than 1.0
   ii. In your method, multiply the random number generated by `Math.random()` by the length of the string and then cast it to an integer
   iii. Return this integer value

b. Now create a second method `checkRandomLetter` which takes a `String` as the argument and returns a `boolean`
   i. Use the `getRandomPosition` function to get a random position in the string and store it to a local variable `position`
   ii. Output a message telling the user to type in a character from the string position indicated by `position` – bear in mind humans like counting from 1 not zero!
   iii. Use a `Scanner` to get input from the keyboard:
   - `Scanner in = new Scanner(System.in);`
   - `String userInput;`
   - `userInput = in.next();`
   iv. Compare the character at `position` within the string to the first character of the string typed in at the keyboard
   v. If the two characters match, return `true`, otherwise return `false`

c. Create a final method called `authenticate()` which has a string `password` as a parameter and returns a `boolean`
   i. It should call `checkRandomLetter` three times, passing it `password` as an argument.
   ii. If the characters at all three random positions are correctly given, return the boolean value `true`.
   iii. Return `false` as soon as one character is entered incorrectly

d. Test the program using the `authenticate` method
   i. correctly input 3 characters
   ii. input the first character incorrectly
   iii. input the last character incorrectly

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

13

proprep

© All rights in this workbook reserved to proprep™

Exceptions

Questions

1) Create a new project and within it a class called `ExceptionsApp`
  a. Create a method called `lookupCalories` which returns a `double` and accepts an `int` as a parameter called `index`
  b. Declare an array of `doubles` called table containing the following values:
     {0, 0.238892, 0.477783, 0.716675, 0.955566, 1.194458, 1.433349, 1.672241}
  c. Using `try`, `catch` blocks, return the value pointed to by `index`, or return 0 in the `catch` block if a `RuntimeException` is thrown along with a message `"Invalid lookup value – 0 to 7 only!"`
  d. Test your method with the following values:
       i. 1 – should return 0.238892
       ii. 0 – should return 0
       iii. 7 – should return 1.672241
       iv. 8 – should return 0 and report an error to the terminal

2) Create a new method called `addHexNumbers` which takes two `Strings` as arguments and also returns a `String` which is the sum of the two arguments, all interpreted and returned as hex
  a. Use the Codepad to experiment with:
       i. `Integer.parseInt()` using two parameters, the first is a `String` containing the number to be parsed, and the second is an `int` for the number base, i.e. 2 for binary, 8 for octal and 16 for hex
       ii. `Integer.toHexString()` which takes one parameter, an integer, which it will convert to a string representation in hex
  b. Use exception handling to trap any `NumberFormatExceptions`, and to print:
       i. A message saying `"Please enter digits only"`
       ii. The value that generated the exception using the `getMessage()` method that is available to all exceptions
  c. If there is no exception return a string representation of the two numbers added together in hex, otherwise return the literal string `"error"`
  d. Test with the following:
       i. "23", "79" – returns "9c"
       ii. "23", "e" – returns "31"
       iii. "23", "g" – returns "error" and prints `"Please enter digits only"`

3) Create a new method called `divideNumbers` which again takes two `Strings` as arguments and also returns a `String`

a. Return as a `String` representation in decimal the first argument *divided* by the second argument. Interpret the inputs as `ints` in decimal

b. Use exception handling to trap:

    i. An `ArithmeticException` – returning the string `"Divide by zero not allowed in integer arithmetic"`

    ii. A `NumberFormatException` – returning the string `"Inputs must be numbers in decimal only"`

c. Create another method called `testDivide()` with no arguments or return values

    i. Use `Scanner` to get two integers from the keyboard

    ii. Call the `divideNumbers()` method, passing the two numbers to the method

    iii. Output the returned String to the screen

    iv. Continue doing this until the two inputs are entered as "-1" and "-1"

d. Run `testDivide()` and test with input values of:

    i. 12 and 3 – outputs 4

    ii. 3.14 and 10 – outputs `"Inputs must be numbers in decimal only"`

    iii. 10 and 0 – outputs `"Divide by zero not allowed in integer arithmetic"`

e. Remove one of the catch blocks, and run an appropriate test again. What difference does the exception block make to the flow of the program?

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

15

© All rights in this workbook reserved to proprep™

4) This exercise makes use of the `lookupCalories` method you created in an earlier exercise

a. Create a new method called `enterCalories()` which takes no arguments and returns no values

b. Add the following as the first line in the method:
`Scanner in = new Scanner(System.in);`

c. Create a `try` block

d. Use a suitable `Scanner` method to get a single integer num from the keyboard inside the `try` block

e. Pass num to the `lookupCalories` method and output num followed by the message `"kJ in kcal is: "` followed by the value returned from `lookupCalories`

f. After the `try` block, create a finally block, which contains two lines:
   i. Output the message "Closing scanner object"
   ii. Close the Scanner object `in` using the `close()` method

g. After the `finally` block, output the message `"Ending program"`

h. Test with values:
   i. 5 – returns 1.194458
   ii. 8 – returns 0 and displays a message
   iii. e – fails to return a value

i. Why did test ii) emit "Ending program" but test iii) did not?

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

16

© All rights in this workbook reserved to proprep™

5) The keyword `throw` in Java can be used by your code to throw an exception under particular conditions

   a. Create a method called `checkString()` which returns a `boolean` and accepts a single `String` argument

   b. Inside the method, use selection statements to check the argument passed in

       i. If it's the literal `null`, throw a `NullPointerException` with the message `"* Invalid input null *"`

       ii. If it's a string of length zero, throw an `IllegalArgumentException` with the message `"* Invalid input empty string *"`

       iii. Outside both selection statements, at the end of the method, return the value `true` if the length of the string is between 8 and 16 characters long, otherwise return `false`

   c. Create a second method called `testStrings()` with no return value or arguments

   d. Within the method, create a `String` array called `strings`, initialised with the values: `{"12345678901234567", "12345678", "1234", "", null, "1234567890123456"}`

   e. Create a `for` loop with an index variable `index` used to iterate through each element of the `strings` array

       i. Within the loop, create `try catch` blocks

       ii. In the try block call the `checkString` method, passing it an element from the `strings` array using the current value of `index`

       iii. Depending on the value returned by `checkString`, display the element value followed by `" – OK"` when `true` is returned, or `" – failed "` when `false` is returned

       iv. In the catch block, catch all exceptions and output the exception details using `getMessage()` to retrieve the detail

   f. Outside the loop, output a message saying `"\nProcessed "` followed by the value of index followed by `" of "` followed by the size of the array

   g. Test the code produces two "failed" messages, two "OK" messages and two error messages

   h. What advantage has `throw` given us in this program?

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

17

6) Compiled Java code runs on a virtual CPU called a Java Virtual Machine (JVM). This allows the code to be run on different CPUs without recompiling. The JVM has a certain amount of memory available which it allocates as requested during runtime

  a. Use the BlueJ Codepad to execute the lines:

     i. `Runtime.getRuntime().freeMemory()`
     ii. `Runtime.getRuntime().totalMemory()`
     iii. `Runtime.getRuntime().maxMemory()`

  b. Create a method called `JVMinfo()` with no return value or parameters

  c. The methods in part a) return a `long` reporting the relevant amount of memory in bytes. Inside the `JVMinfo()` method, use those methods to output messages telling us how much free memory is available, and how much total memory is being currently used by the JVM, both in megabytes

  d. Create another method called `allocate()` with no parameters or return values

     i. Use a loop to repeatedly get an `int` from the keyboard and store it into a variable num and which ends when a number below 1 is entered
     ii. Inside the loop, call the `JVMinfo` method
     iii. Use a `try` and `catch` block inside the loop to catch an `OutOfMemoryError`
     iv. Inside the `try` block attempt to create a new array of `Strings` using the following line: `String[] arr = new String[num * 1024 * 1024];`
     v. In the `catch` block, output to the `System.err` stream using `System.err.println` to send a message saying "Free memory exceeded, maximum available is: ", and output the maximum amount of memory available to the JVM using `Runtime.getRuntime().maxMemory()`

  e. Use another `try` block outside the loop that catches all other exceptions using the `Exception` type and outputs the message `"Invalid input"` to the `System.err` stream

  f. Test `allocate()` with the following inputs:

     i. 10
     ii. 0
     iii. 200
     iv. 500
     v. e

  g. Why does the program continue for test iv) but not for test v) ?

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

File Input and Output

Questions

1) Create a new project and within it a class called `filesApp`. Add a line to import the `java.io` classes

a. Create a method called `writeTimesTable` which accepts a single int argument n and returns a `boolean`

b. Within a `try` block add the following lines:
   i. `FileWriter outfile = new FileWriter("table.txt");`
   ii. `outfile.write("Here is the times table:\n");`

c. Add a loop that iterates a variable `multiplier` from 1 to 12

d. Output to the file the contents of the `multiplier` variable followed by the string `" x "` followed by the contents of the variable n, followed by the string `" = "`

e. Output the product of `multiplier` and n, followed by the literal `"\n"`

f. Outside the loop, close the file using `outfile.close()`

g. As the last line of the `try` block, return the boolean value `true`

h. Add a `catch` block which catches an `IOException`, using `getMessage` to output any error message detail, and then finally returning the boolean value `false`

i. Test your program by running it, and check the contents of the file `table.txt` which will have been saved in the same folder as your project

j. Change the file permissions of the file `table.txt` to read only and run your program again, and check that `false` is returned and check the terminal for the output message.

2) Files can be appended to using a FileWriter argument set to `true`

a. In the Codepad, run the following code and at relevant points check the folder you saved your project in to see the impact:
   i. `import java.io.*;`
   ii. `String nl = System.getProperty("line.separator");`
   iii. `File myFile = new File("testfile.txt");`
   iv. `FileWriter fw = new FileWriter(myFile, true);`
   v. `fw.write("Some stuff ");`
   vi. `fw.write("Some more stuff");`
   vii. `fw.close()`
   viii. `FileWriter fw2 = new FileWriter(myFile, true);`
   ix. `fw2.write(nl + "*END*");`
   x. `fw2.close()`

b. Create a method called `writeTimesTableAppend` based on the previous exercise with an `int` argument n, and `boolean` return value which appends a times table to a file `table.txt` rather than overwrite the file each time.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

proprep

19

© All rights in this workbook reserved to proprep™

   c. Adapt the code to ensure you use the end of line character(s) relevant to any operating system that the program might be run on.

   d. Test your code as follows:
   - i. Delete the `table.txt` file if it exists.
   - ii. Run `writeTimesTableAppend` with an argument of 9.
   - iii. Run `writeTimesTableAppend` again with an argument of 4 – check the file contains the 9 times table followed by the 4 times table.
   - iv. Run `writeTimesTable` (from the previous exercise) with an argument of 3
   - v. What has happened to the contents of the file and why?
   - vi. Run `writeTimesTableAppend` with an argument of 6 and check the contents are what you expect.

3) The Java `File` class allows a programmer to create file paths that can be used to work with files and folders, and that copes with differences between operating systems

   a. In the Codepad, run the following code and at the end check the folder you saved your project in to see the impact:
   - i. `import java.io.*;`
   - ii. `File.separator`
   - iii. `File.separatorChar`
   - iv. `File myFolder = new File("testFolder");`
   - v. `myFolder.exists()`
   - vi. `myFolder.mkdir();`
   - vii. `myFolder.exists()`
   - viii. `File myFile = new File("testFile.txt");`
   - ix. `myFile.exists()`
   - x. `myFile.createNewFile();`
   - xi. `myFile.exists()`

   b. Create a method called `createFileFolder` that returns a `boolean` and accepts two `Strings`, named `folder` and `filename` as arguments.

   c. Within the method, create a new folder:
   - i. Create a new path to the new folder using the `folder` parameter.
   - ii. Use the `mkdir()` method to create the new folder – if the value returned back from `mkdir` is `false`, exit from the method by returning `false.`

   d. Create a new file inside the folder you just created:
   - i. Construct another new path using both the `folder` parameter, a method from part aii or aiii, and the `filename` parameter.
   - ii. Use the `createNewFile()` method to create a new file with the given name in the given folder.
   - iii. If the value returned from `createNewFile` is `true`, return `true` otherwise return `false.`

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

20

© All rights in this workbook reserved to proprep™

e. Test your code with the following folder and filenames and check the files and folders have been created inside your current BlueJ project folder:

    i. Folder name "folder1" and filename "file1.txt".
    ii. Folder name "folder2" and filename "file1.txt".
    iii. Folder name "folder1" and filename "file1.txt".

4) A Scanner can be used to get input from the keyboard or from a file

a. Using the Codepad, try the following:

    i. `import java.io.*;`
    ii. `import java.util.Scanner;`
    iii. `File fileToOpen = new File("README.txt");`
    iv. `Scanner in = new Scanner(fileToOpen);`
    v. `in.hasNext()`
    vi. `in.next()`
    vii. `in.next()`
    viii. `in.next()`
    ix. `in.nextLine()`
    x. `in.nextLine()`
    xi. `in.close()`
    xii. `in.next()`

b. Create a new method called countWords which returns an int, and accepts a `String` argument called filename.

c. Create a path to the file passed in as `filename` .

d. Create a Scanner object using the path created in part c).

e. Create a `String` variable `data`, and an `int` variable `count` which is initialised to 0.

f. Using a while loop which terminates when there are no more tokens to read:

    i. read the next token into `data` using `next()`.
    ii. force all characters inside `data` to lower case.
    iii. replace all characters apart from lower case letters with the empty string `""`.
    iv. Increment `count` if data contains a word (i.e. doesn't contain the empty string).

g. When the loop terminates, close the file and return `count.`

h. If an exception occurred return -1.

i. Test with `README.txt` as the input filename – it should return 52.

j. Test with `missing.txt` as the input filename – it should return -1.

k. Create your own text file with a few test words in it across several lines and check the numbers of words counted by your program is correct.

5) A common file related scenario is to open one file, read data from it and write output to a different file. The file read from can then be deleted and the output file can replace it.

a. Use the Codepad to try the following commands out:

    i. `import java.io.*;`

    ii. `File file1 = new File("results.txt");`

    iii. `File file2 = new File("temp.txt");`

    iv. `file1.createNewFile();`

    v. `file2.createNewFile();`

    vi. `PrintWriter out = new PrintWriter(file2);`

    vii. `out.println("Adele,100");`

    viii. `out.close();`

    ix. `file1.delete();`

    x. `file2.renameTo(file1);`

b. Create a method called `highScore` which returns a `boolean` and accepts a `String` argument `name`, followed by an `int` argument `score`. This method will add the score and name to the file `highscores.txt` as long as it is one of the top 10 scores, or it will always add the score if there are less than 10 scores currently in the file. Each line of the file contains a name, a single comma and a score, for example "Billy,95".

c. Declare the following:

    i. A `String` called `line` with no initial value

    ii. A `String` array called `temp` with no size or initial values

    iii. A `boolean` called `newHighScore` with an initial value of `false`

    iv. An `int` called `count` with an initial value of 0

    v. A `File` path called `scores` to the file `"highscores.txt"`

    vi. A `File` path called `newScores` to the file `"temp.txt"`

d. If the file `"highscores.txt"` does not exist, create a new file with that name, then create

    i. A Scanner object `in` from the `scores` path.

    ii. A `PrintWriter` object `out` from the `newScores` path.

e. Create a `while` loop which terminates when there are no more lines to be read from the input file `highscores.txt` or when `count` has reached 10.

f. Inside the loop, read a whole line from the input file into the string `line.`

g. Split the `line` string on the `","` character and store the result into the `temp` array.

h. If `highScore` is still `false`, and `score` is greater than the value in `temp[1]` do the following:

    i. Output `name` followed by a "," and then `score` as a line in the `temp.txt` file.

    ii. Set `newHighScore` to `true.`

    iii. Increment `count.`

    iv. If `count` is still less than 10, output the contents of `line` to `temp.txt` and increment `count` again.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

22

© All rights in this workbook reserved to proprep™

    i.   Otherwise, (i.e. if `highScore` was not `false`, or `score` was not greater than `temp[1]`), output the contents of `line` to `temp.txt` and increment `count.`

    j.   Outside the while loop, check if count is `0` or both `newHighScore` is `false` and `count` is less than 10. If so,

        i.   Output name followed by a "," and then `score` as a line in the `temp.txt` file.

       ii.   Set `newHighScore` to `true.`

    k.   Close the `Scanner` and `Printwriter` objects, and hence the files.

    l.   If there has been a new high score set,

        i.   delete the `highscores.txt` file.

       ii.   Rename the `temp.txt` file to `highscores.txt.`

      iii.   Return the value `true.`

   m. Otherwise return `false.`

   n. Test the program as follows:

        i.   Add a score of 10, and name of "Tensing" – should add score to newly created file `highscores.txt`.

       ii.   Add a score of 90, and name of "Topi" – should add this score above the previous one in `highscores.txt`.

      iii.   Add a score of 40, and a name of "Joe" – should appear in the middle of the previous two in `highscores.txt`.

      iv.   Add more scores of below 10, until there are 10 names in the file.

       v.   Add another score which is below the lowest current score in the file, and check that false is returned and the file is unchanged.

      vi.   Add another score which is above 10, and check that the bottom score drops out of the file.

6) Files can be written as binary with random access which allows code to seek to a particular place in the file for reading or writing.

   a.   Use the Codepad to do the following:

        i.   `import java.io.*;`

       ii.   `RandomAccessFile file = new RandomAccessFile ("test.bin", "rw");`

      iii.   `file.writeUTF("Here are some characters");`

      iv.   `file.writeInt(1234);`

       v.   `file.writeFloat(3.14F);`

      vi.   `file.length()`

     vii.   `file.seek(0);`

    viii.   `file.readUTF()`

      ix.   `file.readInt()`

       x.   `file.readFloat()`

      xi.   `file.readUTF()`

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

23

© All rights in this workbook reserved to proprep™

b. Create a method called `addStock` which returns a `boolean` and takes in three arguments: a `String` called name, an `int` called `quantity`, and a `float` called `price.`

c. Create a new file called `stocks.bin` for read and write with random access.

d. Use a loop and the string `concat` method to add spaces to the end of the contents of name while name contains less than 6 characters, so for example an input of `"IBM"` becomes `"IBM   ".`

e. Seek to the end of the file, and write `name`, `quantity` and `price` to the file in that order and using their appropriate datatypes.

f. Close the file and return `true` or `false` depending on the method's success.

g. Create a new method called `getStock` which returns a `String` and accepts a single `int` argument called `position.`

h. The method should seek to the stock in the file pointed at by `position`, using the formula `(position-1)*16` given that each stock will take up 16 bytes in the file and it is desirable to count from 1 rather than zero.

i. The method should return the stock name and value read from the file as a `String` by multiplying the `quantity` by the `price` in the form `"IBM value is 13391.0"`. An exception should return the value `"error".`

j. Test with the following inputs for `addStock`
    i. IBM, 100, 133.91.
    ii. JET2.L, 100, 1061.00.
    iii. GLEN.L, 200, 377.65.
    iv. ASL.F, 200, 120.40.
    v. REE.MC, 300, 19.02.

k. Test with the following inputs for `getStock`
    i. 4 - returns "ASL.F value is 24080.0".
    ii. 5 - returns "REE.MC value is 5706.0".
    iii. 6 - returns "error".

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

24

Recursion

Questions

1) Create a new project and within it a class called `RecursionApp`
a. Create a method called `factorial`, which takes an `int` parameter and returns an `int`.
b. The method should use the following algorithm to return the factorial value:
   `if n=1 then return 1 else return factorial n*(n-1) end if.`
c. Test the method with the following values:
   i. 1 - should return 1.
   ii. 3 - should return 6.
   iii. 5 - should return 120.

2) Create a method `fibonacciRecursive` which accepts an `int` argument n, and returns an `int` which is the nth term in the Fibonacci sequence
a. The base case should return 0 for n=0 and 1 for n=1.
b. The recursive case should return the sum of recursive calls with arguments of `(n – 1)` and `(n – 2)`.
c. Test with values:
   i. 0 – returns 0.
   ii. 1 – returns 1.
   iii. 5 – returns 5.
   iv. 8 – returns 21.

3) Green Bottles is a children's folk song in which the number of bottles in each verse decreases until there are no more bottles.
a. Create a method called `greenBottles` which takes an `int` parameter and returns an `int`.
b. Output the following using recursion to initiate the next verse each time until the song finishes on 0 bottles:
   `(number) " green bottles, hanging on the wall,"`
   `(number) " green bottles, hanging on the wall,"`
   `"And if 1 green bottle, should accidently fall…\n"`
   `"There'd be " .`
c. Test that an input of 10 produces the correct output for 10 verses of the song.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

25

© All rights in this workbook reserved to proprep™

4) A string can be recursively checked at its two different ends to see if the string value is a palindrome.

a. Create a method called `isPalindromeRecursive` which accepts a `String` argument, and returns a `Boolean.`

b. Within the method, force the string to lower case.

c. Replace all space characters with an empty string `"".`

d. Replace all characters within the string that are not lower case letters with an empty string `"".`

e. If the length of the string is less than or equal to 1, return `true.`

f. Otherwise, if the first character of the string is the same as the last character of the string, call the `isPalindromeRecursive` function again, but pass it a substring of the original string, which extracts all but the first and last characters of the original string.

g. otherwise return the value `false.`

h. Test with the string "Did Hannah see bees? Hannah did." which should return `true.` Remove one letter to see that it returns `false.`

5) A binary search works by dividing a sorted set of numbers into two halves and checking whether a target value being searched for is in the middle of the set. If the middle value is lower than the target value the left hand half and the middle value is discarded. If the middle value is higher than the target value the right hand half and the middle value is discarded. This happens repeatedly until the target value is found or there is only 1 item left. When the target value is found, the position where it was found is returned, otherwise -1 is returned.

a. Create a new method called `binarySearchRecursive` which returns an `int` and takes as arguments an array of `ints` called `arr` and three `ints left`, `right` and `target.`

b. Use a selection structure to test whether `right` is greater than or equal to `left` .

c. If it is not, return -1.

d. If it is:

    i. Declare an integer variable `middle` and assign it the value.

    ii. `left + (( right – left ) / 2).`

    iii. If the value of the array element pointed at by `middle` is equal to the value of `target`, return the value of `middle.`

    iv. Otherwise check if the value of the array element pointed at by `middle` is greater than `target`. If it is, call `binarySearchRecursive` again, but with values for `left` and `right` that will search only the left hand half of the array.

    v. If the value of the array element pointed at by middle is less than target, call binarySearchRecursive again but with values for left and right that will search only the right hand half of the array.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

26

e. Test the method with the following values:

    i. {1,2,3,4,5,6,7,8}, 0, 7, 2 – should return 1

    ii. {2,4,6,8,10,12,14,16}, 0, 7, 3 – should return -1

    iii. {1,2,4,8,16,32,64,128}, 0, 7, 32 – should return 5

6) A merge sort makes use of recursion to break down an unsorted array into individual elements, and merges left right pairs to reconstruct the sorted array.

a. Create a method called `mergeSort` which returns no values but takes an integer array `arr` and the array length `len` as arguments.

b. The base case is to return from the function when `len` is equal to 1.

c. The recursive case does the following:

    i. Declare a local variable `middle` which is initialized with the value `len / 2.`

    ii. Declare an integer array `left` of size `middle.`

    iii. Declare an integer array `right` of size `len-middle.`

    iv. Using two consecutive `for` loops, copy the values from `arr` element 0 up to (but not including) `middle` into array `left`, and the values from `arr` element `middle` up to `len` into array `right`.

    v. After the second of the two `for` loops, make a recursive call to `mergeSort` passing it the `left` array, with a length of `middle.`

    vi. Make a second recursive call to `mergeSort` passing it the `right` array, with a length of `len-middle.`

    vii. Add the following as the final line of the `mergeSort` method:
`merge(arr, left, right, middle, len-middle);.`

d. Create a second method called `merge` which returns no values but accepts as arguments three `int` arrays called `arr, left` and `right`, and two ints called `leftLen` and `rightLen.`

    i. Inside the method create three integer index variables `l, r` and `a` and initialise all to 0.

    ii. Create a `while` loop which continues whilst `l` is less than `leftLen` and `r` is less than `rightLen.`

    iii. Inside the loop compare the values at `left[l]` and `right[r]`. Put whichever is smallest into `arr[a]` and increment either `l` or `r` depending on which was copied to `arr[a]`. Also increment `a.`

    iv. After the first `while` loop, add another while loop to copy any remaining items from `left` to `arr` using `leftLen.`

    v. Add a final while loop to copy any remaining items from `right` to `arr` using `rightLen.`

e. Create a final method with no return values or arguments to test your code called `mergeTest`.

    i. Within it, declare an array initialised with 8 unsorted values of your choosing.

    ii. Call the `mergeSort` function, passing it your array and its length.

    iii. Use a loop to output the contents of your array to check that the array is now sorted in ascending order.

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

For more information and all the solutions, please go to www.proprep.uk
For any questions please contact us at +44-161-850-4375 or info@proprep.com

28

© All rights in this workbook reserved to proprep™