**Tema 1: Inspections**

a. Write an application that prompts a user for the number of years the user has until retirement and the amount of money the user can save annually.

If the user enters 0 or a negative number for either value, reprompt the user until valid entries are made. Assume that no interest is earned on the money.

Display the amount of money the user will have at retirement. Save the file as RetirementGoal.java.

b. Modify the RetirementGoal application to display the amount of money the user will have if the user earns 5% interest on the balance every year. Save the file as

RetirementGoal2.java.

11. Pickering Manufacturing Company randomly selects one of its four factories to inspect each week. Write an application that determines which factory will be selected each week for the next 52 weeks. Use the Math.random() function explained in Appendix D

to generate a factory number between 1 and 4; you use a statement similar to:

factory = 1 + (int) (Math.random() \* 4);

After each selection, display the factory to inspect, and after the 52 selections are complete, display the percentage of inspections at each factory for the year. Run the

application several times until you are confident that the factory selection is random.

Save the file as Inspections.java.

12. Assume that the population of Mexico is 114 million and that the population

increases 1.01 percent annually. Assume that the population of the United States is

312 million and that the population is reduced 0.15 percent annually. Write an

application that displays the populations for the two countries every year until the

population of Mexico exceeds that of the United States, and display the number of

years it took. Save the file as Population.java.

**Tema 2: Purchase**

Create a class named Purchase. Each Purchase contains an invoice number,

amount of sale, and amount of sales tax. Include set methods for the invoice number and sale amount. Within the set() method for the sale amount, calculate the sales tax as 5% of the sale amount. Also include a display method that displays a purchase’s details. Save the file as Purchase.java. b. Create an application that declares a Purchase object and prompts the user for purchase details. When you prompt for an invoice number, do not let the user proceed until a number between 1,000 and 8,000 has been entered. When you prompt for a sale amount, do not proceed until the user has entered a nonnegative value. After a valid Purchase object has been created, display the object’s invoice number, sale amount, and sales tax. Save the file as

CreatePurchase.java.

**Tema 3: Babbysitting**

Create a BabysittingJob class for Georgette’s Babysitting Service. The class contains

fields to hold the following:

l A job number that contains six digits. The first two digits represent the year, and

the last four digits represent a sequential number. For example, the first job in

2014 has a job number of 140001.

l A code representing the employee assigned to the job. Assume that the code will

always be 1, 2, or 3.

l A name based on the babysitter code. Georgette has three babysitters: (1) Cindy,

(2) Greg, and (3) Marcia.

l The number of children to be watched. Assume that this number is always greater

than zero.

l The number of hours in the job. Assume that all hour values are whole numbers.

l The fee for the job. Cindy is paid $7 per hour per child. Greg and Marcia are paid

$9 an hour for the first child, and $4 per additional hour for each additional child.

For example, if Greg watches three children for two hours, he makes $17 per hour

for two hours, or $34.

Create a constructor for the BabysittingJob class that accepts arguments for the job

number, babysitter code, number of children, and hours. The constructor determines

the babysitter name and fee for the job. Also include a method that displays every

BabysittingJob object field. Save the file as BabysittingJob.java.

Next, create an application that prompts the user for data for a babysitting job. Keep

prompting the user for each of the following values until they are valid:

l A four-digit year between 2013 and 2025 inclusive

l A job number for the year between 1 and 9999 inclusive

l A babysitter code of 1, 2, or 3

l A number of children for the job between 1 and 9 inclusive

l A number of hours between 1 and 12 inclusive

When all the data entries are valid, construct a job number from the last two digits

of the year and a four-digit sequential number (which might require leading zeroes).

Then, construct a BabysittingJob object, and display its values. Save the file as

CreateBabysittingJob.java.

**Tema 4: Quiz**

a. Write an application that creates a quiz. The quiz should contain at least five

questions about a hobby, popular music, astronomy, or any other personal interest.

Each question should be amultiple-choice question with at least four answer options.

When the user answers the question correctly, display a congratulatory message.

If the user responds to a question incorrectly, display an appropriate message as well

as the correct answer. At the end of the quiz, display the number of correct and

incorrect answers and the percentage of correct answers. Save the file as Quiz.java.

b. Modify the Quiz application so that the user is presented with each question

continually until it is answered correctly. Remove the calculation for percentage

of correct answers—all users will have 100 percent correct by the time they

complete the application. Save the file as Quiz2.java.

**Tema 5: Game of Count 21**

Two people play the game of Count 21 by taking turns entering a 1, 2, or 3, which is

added to a running total. The player who adds the value that makes the total exceed

21 loses the game. Create a game of Count 21 in which a player competes against the

computer, and program a strategy that always allows the computer to win. On any

turn, if the player enters a value other than 1, 2, or 3, force the player to reenter the

value. Save the game as Count21.java.

1. Carly’s Catering provides meals for parties and special events. In previous chapters,

you developed a class that holds catering event information and an application that

tests the methods using four objects of the class. Now modify the EventDemo class

to do the following:

l Continuously prompt for the number of guests for each Event until the value

falls between 5 and 100 inclusive.

l For one of the Event objects, create a loop that displays “Please come to my

event!” as many times as there are guests for the Event.

Save the modified file as EventDemo.java.

**Tema 6: Rental Demo**

Sammy’s Seashore Supplies rents beach equipment to tourists. In previous

chapters, you developed a class that holds equipment rental information and an

application that tests the methods using four objects of the class. Now modify the

RentalDemo class to do the following:

l Continuously prompt for the number of minutes of each Rental until the value

falls between 60 and 7,200 inclusive.

l For one of the Rental objects, create a loop that displays “Coupon good for 10

percent off next rental” as many times as there are full hours in the Rental.

Save the modified file as RentalDemo.java.

|  |  |
| --- | --- |
| **Tema7:** | **(Complex Numbers)**  Create a class called Complex for performing arithmetic with complex numbers. Complex numbers have the form    where i is    Write a program to test your class. Use floating-point variables to represent the private data of the class. Provide a constructor that enables an object of this class to be initialized when it is declared. Provide a no-argument constructor with default values in case no initializers are provided. Provide public methods that perform the following operations:   1. Add two Complex numbers: The real parts are added together and the imaginary parts are added together. 2. Subtract two Complex numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand. 3. Print Complex numbers in the form (a, b), where a is the real part and b is the imaginary part. |
| **Tema 8:** | **(Set of Integers)**  Create class IntegerSet. Each IntegerSet object can hold integers in the range 0100. The set is represented by an array of booleans. Array element a[i] is true if integer i is in the set. Array element a[j] is false if integer j is not in the set. The no-argument constructor initializes the Java array to the "empty set" (i.e., a set whose array representation contains all false values).  Provide the following methods: Method union creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third set's array is set to TRue if that element is true in either or both of the existing setsotherwise, the element of the third set is set to false). Method intersection creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set's array is set to false if that element is false in either or both of the existing setsotherwise, the element of the third set is set to true). Method insertElement inserts a new integer k into a set (by setting a[k] to true). Method deleteElement deletes integer m (by setting a[m] to false). Method toSetString returns a string containing a set as a list of numbers separated by spaces. Include only those elements that are present in the set. Use --- to represent an empty set. Method isEqualTo determines whether two sets are equal. Write a program to test class IntegerSet. Instantiate several IntegerSet objects. Test that all your methods work properly. |
| **Tema 9:** | **(Date Class)**  Create class Date with the following capabilities:   1. Output the date in multiple formats, such as 2. MM/DD/YYYY 3. June 14, 1992 4. DDD YYYY 5. Use overloaded constructors to create Date objects initialized with dates of the formats in part (a). In the first case the constructor should receive three integer values. In the second case it should receive a String and two integer values. In the third case it should receive two integer values, the first of which represents the day number in the year. [Hint: To convert the string representation of the month to a numeric value, compare strings using the equals method. For example, if s1 and s2 are strings, the method call s1.equals( s2 ) returns true if the strings are identical and otherwise returns false.] |
|  | [Page 414] |
| **Tema 10:** | **(Rational Numbers)**  Create a class called Rational for performing arithmetic with fractions. Write a program to test your class. Use integer variables to represent the private instance variables of the classthe numerator and the denominator. Provide a constructor that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form. The fraction  2/4  is equivalent to 1/2 and would be stored in the object as 1 in the numerator and 2 in the denominator. Provide a no-argument constructor with default values in case no initializers are provided. Provide public methods that perform each of the following operations:   1. Add two Rational numbers: The result of the addition should be stored in reduced form. 2. Subtract two Rational numbers: The result of the subtraction should be stored in reduced form. 3. Multiply two Rational numbers: The result of the multiplication should be stored in reduced form. 4. Divide two Rational numbers: The result of the division should be stored in reduced form. 5. Print Rational numbers in the form a/b, where a is the numerator and b is the denominator. 6. Print Rational numbers in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.) |

**Tema 11: Tema 7: Rrjeti social**

Shkruani nje program qe realizon nje rrjet social me afishim si me poshte:  
  
Miresevini ne rrjetin social CPLUSPLUS!  
MENU 1:  
Shtypni:  
  1: Hyj ne llogari  
  2: Regjistrohu  
  3: Dil nga programi  
Nese perdoruesi jep 1 shfaqet MENU 2:

MENU 2:  
Jepni emrin (lexohet nje string dhe testohet nese eshte ne listen e anetareve nese kemi hyrje ne llogari. Nese kemi regjistrim dhe anetari eshte i regjistruar atehere te lexohet nje emer i ri.)  
Jepni fjalekalimin (lexohet nje string dhe testohet nese eshte i sakte nese kemi hyrje ne llogari. Nese kemi regjistrim passwordi duhet te jete minimumi 6 karaktere)  
  
Me pas afishohet:

Hyrje e suksesshme

dhe kalohet tek MENU3:

MENU3:  
  1: Shih Kerkesat  
  2: Shih Miqte  
  3: Shto Mik  
  4: Hiq mik  
  5: Dil nga llogaria  
Nese Perdoruesi jep 1 afishohet:  
Ju zgjodhet 1: Shih Kerkesat dhe me pas shfaqet perseri MENU3:  
Ne te njejten menyre procedohet per zgjedhjen 2.

Nese perdoruesi zgjedh 3 ose 4 afishohet MENU 4:

MENU 4:

Jepni emrin e mikut qe do shtoni /hiqni  
(lexohet nje emer dhe kontrollohet ne listen e miqve)

Kerkesa u dergua / Miku u fshi  
Nese perdoruesi jep nje vlere te ndryshme nga 1-5 lexohet nje numer i ri deri sa te jete ne kete rang.

Nese perdoruesi zgjedh  5 shfaqet perseri MENU 1:  
  1: Hyj ne llogari  
  2: Regjistrohu  
  3: Dil nga programi  
Nese perdoruesi jep 2 shfaqet perseri MENU 2: dhe me pas afishohet

Regjistrimi u krye me sukses!

Shfaqet perseri MENU 1.  
Nese perdoruesi jep 3 programi perfundon.

**Tema 12: Projektimi i nje qarku me metoden “Sum of product”**

Ne kete program kerkohet qe te lexohen nga tastiera shprehjet Boolean te formes “Sum of product” dhe te afishohen kombinimet e duhura te variablave (A, B, C, D) ne menyre qe te krijoje nje funksion qe vlera e tij te jete 1.

Shenim:

Per te dhene komplementin e pare te variablit, emertimi behet me germe te vogel.

Per mbylljen e leximit te variablave jepni karakterin X

Psh per funksionin:

X= ABC+Abc+aBC+ABc

Duhet te japim nga tastiera

ABC

Abc

aBC

ABc

Q

Ne kete rast shprehja per kombinimin A=1, B=1 dhe C=1 do kishte formen:

111+100+011+110=1

Programi duhet te afishoje vlerat e A B dhe C si dhe shprehjen perkatese.

Te modifikohet programi per nje numer te dhene variablash nga 2 ne 8.

**Tema 13: Projektimi i nje qarku me metoden “Product of sum”**

Ne kete program kerkohet qe te lexohen nga tastiera shprehjet Boolean te formes “Product of sum” dhe te afishohen kombinimet e duhura te variablave (A, B, C, D) ne menyre qe te krijoje nje funksion qe vlera e tij te jete 0.

Shenim:

Per te dhene komplementin e pare te variablit, emertimi behet me germe te vogel.

Per mbylljen e leximit te variablave jepni karakterin X

Psh per funksionin:

Y= (A+B+C) \* (a+b+C) \* (A+B+c)

Duhet te japim nga tastiera

ABC

abC

ABc

X

Ne kete rast shprehja per kombinimin A=0, B=0 dhe C=0 do kishte formen:

(0+0+0)\*(1+1+0)\*(0+0+1)=0

Programi duhet te afishoje vlerat e A B dhe C si dhe shprehjen perkatese.

Te modifikohet programi per nje numer te dhene variablash nga 2 ne 8.

**Tema 14: Vala Digjitale**

Jepet nje numer binar me n bit dhe shpejtesia e perhapjes se sinjalit (ne bit per sec)

Te afishohet vala dixhitale per kete numer grafikisht si dhe te llogaritet dhe afishohet koha e nje biti.

P.sh nese numri i dhene eshte 0011101 te afishohet:

**\_\_|¯¯¯|\_|¯|**

Per afishimin e simbolit ¯ te perdoret numri ne kodin ASCII 175

Pas afishimit programi duhet te lexoje nje numer te ri binar.

Te merren parasysh rastet kur perdoruesi nuk jep numer binar duke afishuar mesazhe gabimi.

Nese jepet -1 programi mbyllet.

**Tema 15: (Airline Reservations System)**

A small airline has just purchased a computer for its new automated reservations system. You

have been asked to program the new system. You are to write a program to assign seats on each flight of the airline’s only plane

(capacity: 10 seats).

Your program should display the following menu of alternatives— **Please type 1 for "First Class"** and **Please**

**type 2 for "Economy"**. If the person types **1**, your program should assign a seat in the first class section (seats 1-5). If the person

types **2**, your program should assign a seat in the economy section (seats 6-10). Your program should print a boarding pass

indicating the person’s seat number and whether it is in the first class or economy section of the plane.

Use a single-subscripted array to represent the seating chart of the plane. Initialize all the elements of the array to 0 to indicate

that all seats are empty. As each seat is assigned, set the corresponding elements of the array to 1 to indicate that the seat is no

longer available.

Your program should, of course, never assign a seat that has already been assigned. When the first class section is full, your

program should ask the person if it is acceptable to be placed in the nonsmoking section (and vice versa). If yes, then make the

appropriate seat assignment. If no, then print the message **"Next flight leaves in 3 hours."**

**Tema 16: (Turtle Graphics)**

The Logo language, which is particularly popular among personal computer users, made the concept of

*turtle graphics* famous. Imagine a mechanical turtle that walks around the room under the control of a JAVA program. The turtle

holds a pen in one of two positions, up or down. While the pen is down, the turtle traces out shapes as it moves; while the pen is up,

the turtle moves about freely without writing anything. In this problem, you will simulate the operation of the turtle and create a

computerized sketchpad as well.

Use a 20-by-20 array **floor** that is initialized to zeros. Read commands from an array that contains them. Keep track of the

current position of the turtle at all times and whether the pen is currently up or down. Assume that the turtle always starts at position

0,0 of the floor with its pen up. The set of turtle commands your program must process are as follows:

Suppose that the turtle is somewhere near the center of the floor. The following “program” would draw and print a 12-by-12

square and end with the pen in the up position:

**2**

**5,12**

**3**

**5,12**

**3**

**5,12**

**3**

**5,12**

**169**

As the turtle moves with the pen down, set the appropriate elements of array **floor** to **1**’s. When the **6** command (print) is given,

wherever there is a **1** in the array, display an asterisk or some other character you choose. Wherever there is a zero, display a blank.

Write a program to implement the turtle graphics capabilities discussed here. Write several turtle graphics programs to draw

interesting shapes. Add other commands to increase the power of your turtle graphics language.

**Tema 17: (Knight’s Tour)**

One of the more interesting puzzlers for chess buffs is the Knight’s Tour problem, originally proposed by

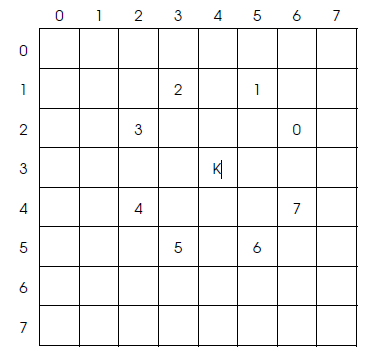
the mathematician Euler. The question is this: Can the chess piece called the knight move around an empty chessboard and touch

each of the 64 squares once and only once? We study this intriguing problem in depth here.

The knight makes L-shaped moves (over two in one direction and then over one in a perpendicular direction). Thus, from a

square in the middle of an empty chessboard, the knight can make eight different moves (numbered 0 through 7) as shown in Fig.

4.25.



a) Draw an 8-by-8 chessboard on a sheet of paper and attempt a Knight’s Tour by hand. Put a **1** in the first square you

move to, a **2** in the second square, a **3** in the third, etc. Before starting the tour, estimate how far you think you will get,

remembering that a full tour consists of 64 moves. How far did you get? Was this close to your estimate?

b) Now let us develop a program that will move the knight around a chessboard. The board is represented by an 8-by-8

double-subscripted array **board**. Each of the squares is initialized to zero. We describe each of the eight possible

moves in terms of both their horizontal and vertical components. For example, a move of type 0, as shown in Fig. 4.25,

consists of moving two squares horizontally to the right and one square vertically upward. Move 2 consists of moving

one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves

upward are indicated with negative numbers. The eight moves may be described by two single-subscripted arrays,

**horizontal** and **vertical**, as follows:

**horizontal[ 0 ] = 2**

**horizontal[ 1 ] = 1**

**horizontal[ 2 ] = -1**

**horizontal[ 3 ] = -2**

**horizontal[ 4 ] = -2**

**horizontal[ 5 ] = -1**

**horizontal[ 6 ] = 1**

**horizontal[ 7 ] = 2**

**vertical[ 0 ] = -1**

**vertical[ 1 ] = -2**

**vertical[ 2 ] = -2**

**vertical[ 3 ] = -1**

**vertical[ 4 ] = 1**

**vertical[ 5 ] = 2**

**vertical[ 6 ] = 2**

**vertical[ 7 ] = 1**

Let the variables **currentRow** and **currentColumn** indicate the row and column of the knight’s current position.

To make a move of type **moveNumber**, where **moveNumber** is between 0 and 7, your program uses the statements

**currentRow += vertical[ moveNumber ];**

**currentColumn += horizontal[ moveNumber ];**

Keep a counter that varies from **1** to **64**. Record the latest count in each square the knight moves to. Remember to test

each potential move to see if the knight has already visited that square, and, of course, test every potential move to

make sure that the knight does not land off the chessboard. Now write a program to move the knight around the chessboard.

Run the program. How many moves did the knight make?

c) After attempting to write and run a Knight’s Tour program, you have probably developed some valuable insights. We

will use these to develop a *heuristic* (or strategy) for moving the knight. Heuristics do not guarantee success, but a carefully

developed heuristic greatly improves the chance of success. You may have observed that the outer squares are

more troublesome than the squares nearer the center of the board. In fact, the most troublesome, or inaccessible, squares

are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave

open those that are easiest to get to, so when the board gets congested near the end of the tour, there will be a greater

chance of success.

We may develop an “accessibility heuristic” by classifying each of the squares according to how accessible they

are and then always moving the knight to the square (within the knight’s L-shaped moves, of course) that is most inaccessible.

We label a double-subscripted array **accessibility** with numbers indicating from how many squares

each particular square is accessible. On a blank chessboard, each center square is rated as **8**, each corner square is

rated as **2** and the other squares have accessibility numbers of **3**, **4** or **6** as follows:

**2 3 4 4 4 4 3 2**

**3 4 6 6 6 6 4 3**

**4 6 8 8 8 8 6 4**

**4 6 8 8 8 8 6 4**

**4 6 8 8 8 8 6 4**

**4 6 8 8 8 8 6 4**

**3 4 6 6 6 6 4 3**

**2 3 4 4 4 4 3 2**

Now write a version of the Knight’s Tour program using the accessibility heuristic. At any time, the knight should

move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied

squares. Therefore, the tour may begin in any of the four corners. (*Note:* As the knight moves around the chessboard,

your program should reduce the accessibility numbers as more and more squares become occupied. In this way, at any

given time during the tour, each available square’s accessibility number will remain equal to precisely the number of

squares from which that square may be reached.) Run this version of your program. Did you get a full tour? Now modify

the program to run 64 tours, one starting from each square of the chessboard. How many full tours did you get?

d) Write a version of the Knight’s Tour program which, when encountering a tie between two or more squares, decides

what square to choose by looking ahead to those squares reachable from the “tied” squares. Your program should move

to the square for which the next move would arrive at a square with the lowest accessibility number.

**Tema 18. (Eight Queens)**

Another puzzler for chess buffs is the Eight Queens problem. Simply stated: Is it possible to place eight

queens on an empty chessboard so that no queen is “attacking” any other, i.e., no two queens are in the same row, the same column,

or along the same diagonal? Use the thinking developed in Exercise 4.24 to formulate a heuristic for solving the Eight Queens problem.

Run your program. (*Hint:* It is possible to assign a value to each square of the chessboard indicating how many squares of an

empty chessboard are “eliminated” if a queen is placed in that square. Each of the corners would be assigned the value 22, as in Fig.



Once these “elimination numbers” are placed in all 64 squares, an appropriate heuristic might be: Place the next queen in the

square with the smallest elimination number. Why is this strategy intuitively appealing?

**Tema 19. (Bucket Sort)**

A bucket sort begins with a single-subscripted array of positive integers to be sorted and a double-subscripted

array of integers with rows subscripted from 0 to 9 and columns subscripted from 0 to *n* - 1, where *n* is the number of values in the

array to be sorted. Each row of the double-subscripted array is referred to as a bucket. Write a function **bucketSort** that takes an

integer array and the array size as arguments and performs as follows:

a) Place each value of the single-subscripted array into a row of the bucket array based on the value’s ones digit. For example,

97 is placed in row 7, 3 is placed in row 3 and 100 is placed in row 0. This is called a “distribution pass.”

b) Loop through the bucket array row by row, and copy the values back to the original array. This is called a “gathering

pass.” The new order of the preceding values in the single-subscripted array is 100, 3 and 97.

c) Repeat this process for each subsequent digit position (tens, hundreds, thousands, etc.).

On the second pass, 100 is placed in row 0, 3 is placed in row 0 (because 3 has no tens digit) and 97 is placed in row 9. After the

gathering pass, the order of the values in the single-subscripted array is 100, 3 and 97. On the third pass, 100 is placed in row 1, 3

is placed in row zero and 97 is placed in row zero (after the 3). After the last gathering pass, the original array is now in sorted

order.

Note that the double-subscripted array of buckets is 10 times the size of the integer array being sorted. This sorting technique

provides better performance than a bubble sort, but requires much more memory. The bubble sort requires space for only one additional

element of data. This is an example of the space–time trade-off: The bucket sort uses more memory than the bubble sort, b ut

performs better. This version of the bucket sort requires copying all the data back to the original array on each pass. Another possibility

is to create a second double-subscripted bucket array and repeatedly swap the data between the two bucket arrays.

**Tema 20 : (Selection Sort)** A selection sort searches an array looking for the smallest element in the array. Then, the smallest element

is swapped with the first element of the array. The process is repeated for the subarray beginning with the second element of the

array. Each pass of the array results in one element being placed in its proper location. This sort performs comparably to the bubble

sort—for an array of *n* elements, *n* - 1 passes must be made, and for each subarray, *n* - 1 comparisons must be made to find the

smallest value. When the subarray being processed contains one element, the array is sorted. Write recursive function **selectionSort** to perform this algorithm.

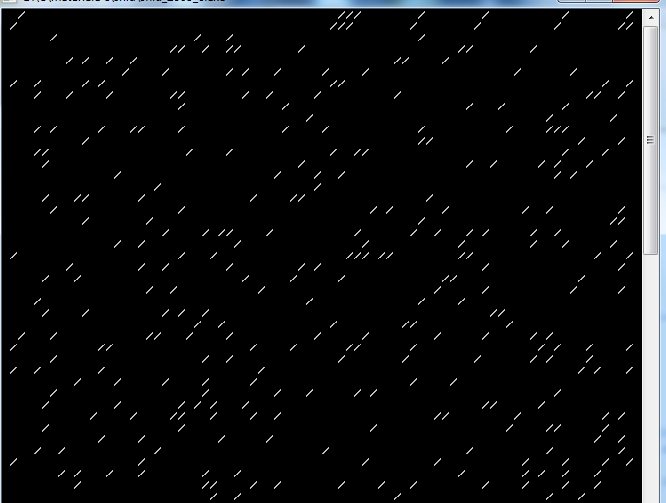
**Tema 21: Rënia e shiut.**

Shkruani nje program qe simulon renien e shiut ne JAVA duke perdorur vektoret dydimensionale.

Te perdoret instruksioni

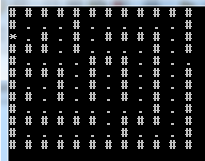
system(“cls”);

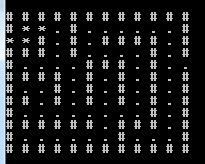
per pastrimin e ekranit kur eshte e nevojshme.

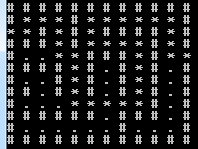


**Tema 22: (Maze Traversal)**

The following grid of hashes (**#**) and dots (**.**) is a double-subscripted array representation of a maze:







In the preceding double-subscripted array, the hashes ( **#**), represent the walls of the maze and the dots represent squares in the possible

paths through the maze. Moves can only be made to a location in the array that contains a dot.

There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming that there is an exit). If

there is not an exit, you will arrive at the starting location again. Place your right hand on the wall to your right and begin walking

forward. Never remove your hand from the wall. If the maze turns to the right, you follow the wall to the right. As long as you do

not remove your hand from the wall, eventually you will arrive at the exit of the maze. There may be a shorter path than the one

you have taken, but you are guaranteed to get out of the maze if you follow the algorithm.

Write recursive function **mazeTraverse** to walk through the maze. The function should receive as arguments a 12-by-12

character array representing the maze and the starting location of the maze. As **mazeTraverse** attempts to locate the exit from

the maze, it should place the character **X** in each square in the path. The function should display the maze after each move so the

user can watch as the maze is solved.

**Tema 23: Kodimi i nje teksti duke perdorur nje fjale si celes kodimi.**

Nese jepet teksti p dhe fjalakyce k (nje string ku A dhe a perfaqesojne 0, kurse Z dhe z perfaqesojne 25), atehere cdo germe, ci, ne tekstin c, llogaritet si me poshte:

ci = (pi + kj) % 26

Shkruani nje program ne JAVA qe lexon nje tekst dhe me pas nje string (celesin e kodimit) dhe afishon tekstin e koduar sipas menyres se mesiperme. Nese teksti permban karaktere te tjera qe nuk jane germa (si numra, simbole, space etj) atehere ato mbeten te pandryshueshme.

Jepni celesin e kodimit

**bacon**

Jepni tekstin qe do kodohet:

**Meet me at the park at eleven am**

Teksti i koduar eshte:

**Negh zf av huf pcfx bt gzrwep oz**

**Tema 24: Tabela e vertetesise**

Shkruani nje program qe afishon ne ekran tabelen e vertetesise per x kombinime (0,1) qe kerkohen ne ekuacion

Ku x merr vlerat nga 2 ne 8. Te afishohen gjithashtu ne dy kolonat djathtas vlerat e AND dhe OR per secilin kombinim.

P.sh. nese x = 2 do afishohej tabela:

A B AND OR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0 0 0 0

0 1 0 1

1 0 0 1

1 1 1 1

**Tema 25: Kodi ABEL**

Shkruani nje program qe lexon nga tastiera nje shprehje boolean dhe afishon shprehjen e konvertuar ne kodin ABEL

Kodi ABEL perdor simbolet e meposhtme:

a (A e invertuar) !A

A AND B A&B

A OR B A#B

Psh: Nese jepet nga tastiera shprehja

Y= (A + B + C) \* (a + b + C)

duhet te afishohet

Y = (A#B#C)&(!A#!B#C)

**Tema 26: Loja Tic Tac Toe**

Tic Tac Toe

1 2 3

1 | |

---|---|---

2 | |

---|---|---

3 | |

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 2

Kolona : 2

Tic Tac Toe

1 2 3

1 | |

---|----|---

2 | X |

---|----|---

3 | |

Lojtari 2 eshte 'O': jep rreshtin dhe kolonen

Rreshti : 1

Kolona : 1

Tic Tac Toe

1 2 3

1 O | |

---|---|---

2 | X |

---|---|---

3 | |

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 3

Kolona : 3

Tic Tac Toe

1 2 3

1 O | |

---|---|---

2 | X |

---|---|---

3 | | X

Lojtari 2 eshte 'O': jep rreshtin dhe kolonen

Rreshti : 2

Kolona : 3

Tic Tac Toe

1 2 3

1 O | |

---|---|---

2 | X | O

---|---|---

3 | | X

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 1

Kolona : 2

Tic Tac Toe

1 2 3

1 O | X |

---|---|---

2 | X | O

---|---|---

3 | | X

Lojtari 2 eshte 'O': jep rreshtin dhe kolonen

Rreshti : 3

Kolona : 2

Tic Tac Toe

1 2 3

1 O | X |

---|---|---

2 | X | O

---|---|---

3 | O | X

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 2

Kolona : 1

Tic Tac Toe

1 2 3

1 O | X |

---|---|---

2 X | X | O

---|---|---

3 | O | X

Lojtari 2 eshte 'O': jep rreshtin dhe kolonen

Rreshti : 3

Kolona : 1

Tic Tac Toe

1 2 3

1 O | X |

---|---|---

2 X | X | O

---|---|---

3 O | O | X

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 2

Kolona : 2

Zgjidhni nje vend tjeter!

row : 1

column : 3

Tic Tac Toe

1 2 3

1 O | X | X

---|----|---

2 X | X | O

---|----|---

3 O | O | X

Fituat te dy!!!

Doni te luani perseri?(Y - N)

y

Tic Tac Toe

1 2 3

1 | |

---|---|---

2 | |

---|---|---

3 | |

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 1

Kolona : 1

Tic Tac Toe

1 2 3

1 X | |

---|---|---

2 | |

---|---|---

3 | |

Lojtari 2 eshte 'O': jep rreshtin dhe kolonen

Rreshti : 2

Kolona : 2

Tic Tac Toe

1 2 3

1 X | |

---|---|---

2 | O |

---|---|---

3 | |

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 2

Kolona : 1

Tic Tac Toe

1 2 3

1 X | |

---|---|---

2 X | O |

---|---|---

3 | |

Lojtari 2 eshte 'O': jep rreshtin dhe kolonen

Rreshti : 3

Kolona : 1

Tic Tac Toe

1 2 3

1 X | |

---|---|---

2 X | O |

---|---|---

3 O | |

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 1

Kolona : 3

Tic Tac Toe

1 2 3

1 X | | X

---|---|---

2 X | O |

---|---|---

3 O | |

Lojtari 2 eshte 'O': jep rreshtin dhe kolonen

Rreshti : 3

Kolona : 2

Tic Tac Toe

1 2 3

1 X | | X

---|---|---

2 X | O |

---|---|---

3 O | O |

Lojtari 1 eshte 'X': jep rreshtin dhe kolonen

Rreshti : 1

Kolona : 2

Tic Tac Toe

1 2 3

1 X | X | X

---|---|---

2 X | O |

---|---|---

3 O | O |

Player 1 wins

Doni te luani perseri?(Y - N)