

**מס' נבחן**

**בחינות**

**שם הקורס: מבוא למדעי המחשב**

**קוד הקורס: 10006**

|  |  |  |
| --- | --- | --- |
| **הוראות לנבחן:**   * **חומר עזר שימושי לבחינה:**   **כל חומר עזר מודפס** | **בחינת סמסטר: ב'**  **השנה: תש"פ**  **מועד:** | |
| * **אין להשתמש בטלפון סלולארי** * **אין להשתמש במחשב אישי או נייד** * **אין להשתמש בדיסק און קי ו/או מכשיר מדיה אחר** * **אין להפריד את דפי שאלון הבחינה** | **תאריך הבחינה:**  **שעת הבחינה:**  **משך הבחינה: 240 דקות**  **השאלון לא ייבדק בתום הבחינה**  **מרצים: קרן כליף ואלכס טילקין** | |
| **מבנה הבחינה והנחיות לפתרון:** | |

* במבחן 4 שאלות, יש לענות על כולן.
* על הקוד להיות פשוט, מסודר וברור. קוד מסורבל יגרור הורדת ניקוד.
* יש להשתמש בפונקציות שנלמדו, ואין "להמציא את הגלגל מחדש", אלא אם נאמר במפורש אחרת
* **אין להשתמש בחומר שלא נלמד בקורס זה**
* יש להקפיד על כל כללי הכתיבה שנלמדו, ובפרט על קריאות, מודלוריות, שימוש בקבועים ופשטות.
* מותר ומומלץ לכתוב פונקציות עזר במקרה הצורך!

**בהצלחה!**

כל הזכויות שמורות ©. מבלי לפגוע באמור לעיל, אין להעתיק, לצלם, להקליט, לשדר, לאחסן מאגר מידע, בכל דרך שהיא, בין מכאנית ובין אלקטרונית או בכל דרך אחרת כל חלק שהוא מטופס הבחינה

**שאלה 1 (45 נקודות):**

איגוד הגלישה העולמי (World Surf League -WSL) אשר השכיר את שירותיכם על מנת שתתחזקו עבורו מערכת לניהול אליפות העולם בגלישה.

להלן תיאור של מחלקות הקיימות במערכת ומחלקות **שעליכם להוסיף או לעדכן**, בהמשך יוצג main לדוגמה ופלט תוצאת הרצתו.

1. נתונה בהמשך באופן חלקי המחלקה עבור גלשן (SurfBoard):
   1. תכונות המחלקה: צבע הגלשן, אורכו וסוג.
   2. קונסטרקטור המקבל את כל התכונות ומאתחל את האובייקט.
   3. **(5 נק') הוסיפו** למחלקה תמיכה במספר סידורי אוטומטי שינתן עם יצירת האובייקט. המספר הסידורי יתחיל מ- 1000.
   4. מתודה toString המחזירה מחרוזת עם נתוני הגלשן כפי שמופיע בפלט בהמשך.
2. נתונה באופן חלקי מחלקה עבור גולש (Surfer):
   1. תכונות המחלקה הן שם הגולש, מאיזו מדינה הוא, הדירוג העולמי שלו ומערך הגלשנים שיש לו. המספר המקסימלי של גלשנים לגולש הוא 20. מותר להוסיף תכונות נוספות במידת הצורך.
   2. נתונה המתודה המקבלת מספר סידורי של גלשן, ובמידה לגולש ישנה הפניה לגלשן עם מספר סידורי זה, המתודה תחזיר הפניה לגלשן, אחרת תחזיר null.
   3. נתונה המתודה המקבלת סוג של גלשן, ומחזירה true האם לגולש יש לפחות גלשן אחד מסוג זה, אחרת המתודה מחזירה null.
   4. נתונה מתודה toString המחזירה מחרוזת עם נתוני הגולש.
   5. **(5 נק') כתבו** קונסטרקטור המקבל את שם הגולש, המדינה אליה הוא משוייך והדירוג הנוכחי שלו, ותאתחל את כל שדות האובייקט.
   6. **(5 נק') כתבו** מתודה המוסיפה לגולש גלשן. המתודה תוסיף את הגלשן במידה ויש מקום במערך הגלשנים ובמידה ולגולש עדיין אין גלשן עם מספר סידורי זה. במקרה של הכנסה המתודה תחזיר true, אחרת תחזיר false.
   7. **(5 נק') כתבו** מתודה המחזירה את הגלשן הארוך ביותר שיש לגולש. במידה ולגולש עדיין אין גלשנים, יש להחזיר null.
3. נתונה באופן חלקי מחלקה עבור WSL (עבור איגוד הגלישה העולמי).
   1. תכונות המחלקה: מערך הגולשים הרשומים באיגוד ומיקום האליפות האחרונה שהאיגוד קיים.
   2. נתון קונסטרקטור המקבל את המיקום האחרון שבו התקיימה האליפות.
   3. נתונה מתודה המעדכנת את המיקום האחרון בו התקיימה האליפות.
   4. נתונה מתודה המוסיפו גולש לאיגוד.
   5. נתונה מתודה toString המחזירה מחרוזת עם נתוני האיגוד.
   6. **(5 נק') כתבו** מתודה המחזירה את הגולש שיש לו את הגלשן הכי ארוך.
   7. **(5 נק') כתבו** מתודה המקבלת סוג מסויים של גלשן, ומחזירה לכמה גולשים יש גלשן מסוג זה.
   8. **(10 נק') כתבו** מתודה המחזירה את סוג הגלשן שיש להכי הרבה גולשים.
   9. **(5 נק') כתבו** מתודה המקבלת מספר סידורי של גלשן ומחזירה את הגולש אליו שייך. במידה ואינו שייך לאף גולש המתודה תחזיר null.

להלן main לדוגמה:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

SurfBoard sb1 = **new** SurfBoard("Black", 6.0f, SurfBoard.BoardType.***Fish***);

SurfBoard sb2 = **new** SurfBoard("White", 6.8f, SurfBoard.BoardType.***Fun***);

SurfBoard sb3 = **new** SurfBoard("Pink", 5.6f, SurfBoard.BoardType.***Hybrid***);

SurfBoard sb4 = **new** SurfBoard("Yellow", 5.4f, SurfBoard.BoardType.***ShortBoard***);

SurfBoard sb5 = **new** SurfBoard("Red", 5.4f, SurfBoard.BoardType.***ShortBoard***);

SurfBoard sb6 = **new** SurfBoard("Blue", 5.4f, SurfBoard.BoardType.***ShortBoard***);

Surfer s1 = **new** Surfer("Kelly Slater", "US", 8);

Surfer s2 = **new** Surfer("Kolohe Andino", "US", 4);

Surfer s3 = **new** Surfer("Italo Ferreira", "US", 1);

s1.addSurfBoard(sb1);

s1.addSurfBoard(sb2);

s2.addSurfBoard(sb3);

s2.addSurfBoard(sb4);

s3.addSurfBoard(sb5);

s3.addSurfBoard(sb6);

WSL wsl = **new** WSL("Bells Beach, Melborne, Australia");

wsl.addSurfer(s1);

wsl.addSurfer(s2);

wsl.addSurfer(s3);

System.***out***.println(wsl.toString());

System.***out***.println("====================================");

Surfer surferBySerialNumber = wsl.findSurferByBoardSerialNumber(1005);

System.***out***.println("The surfer with surfboard with the serial number 1005:");

System.***out***.println(

surferBySerialNumber != **null** ? surferBySerialNumber.getName() : "-");

System.***out***.println("====================================");

surferBySerialNumber = wsl.findSurferByBoardSerialNumber(1200);

System.***out***.println("The surfer with surfboard with the serial number 1200:");

System.***out***.println(

surferBySerialNumber != **null** ? surferBySerialNumber.getName() : "-");

System.***out***.println("====================================");

Surfer surfer = wsl.findSurferWithLongestSurfBoard();

System.***out***.println("The surfer who has the longest surfboard is:");

System.***out***.println(surfer.getName());

System.***out***.println("====================================");

SurfBoard.BoardType boardType = wsl.findTheMostUsedBoardType();

System.***out***.println("The most used board type is " + boardType.toString());

}

}

להלן פלט עבור main זה:

Last location was Bells Beach, Melborne, Australia, and the surfurs were:

Name: Kelly Slater(US), World Rank: 8, Surboards:

color: Black, length: 6.0, Serial Number: 1000, Board Type: Fish

color: White, length: 6.8, Serial Number: 1001, Board Type: Fun

Name: Kolohe Andino(US), World Rank: 4, Surboards:

color: Pink, length: 5.6, Serial Number: 1002, Board Type: Hybrid

color: Yellow, length: 5.4, Serial Number: 1003, Board Type: ShortBoard

Name: Italo Ferreira(US), World Rank: 1, Surboards:

color: Red, length: 5.4, Serial Number: 1004, Board Type: ShortBoard

color: Blue, length: 5.4, Serial Number: 1005, Board Type: ShortBoard

====================================

The surfer with surfboard with the serial number 1005:

Italo Ferreira

====================================

The surfer with surfboard with the serial number 1200:

-

====================================

The surfer who has the longest surfboard is:

Kelly Slater

====================================

**השלימו את המחלקה SurfBoardעפ"י הנדרש בהגדרות למעלה:**

**public** **class** SurfBoard {

**public** **enum** BoardType{***ShortBoard***, ***Fish***, ***Fun***, ***LongBoard***, ***Hybrid***, ***Gun***, ***Sup***};

**private** **static** **int** *serialNumberGenerator* = 1000;

**private** String color;

**private** **float** length;

**private** **int** serialNumber;

**private** BoardType boardType;

**public** SurfBoard(String color, **float** length, BoardType boardType) {

setColor(color);

**this**.length = length;

**this**.boardType = boardType;

serialNumber = *serialNumberGenerator*++;

}

@Override

**public** String toString() {

StringBuffer sb = **new** StringBuffer();

sb.append("color: " + color + ", ");

sb.append("length: " + length + ", ");

sb.append("Serial Number: " + serialNumber + ", ");

sb.append("Board Type: " + boardType.toString());

**return** sb.toString();

}

**public** **void** setColor(String color) {

**this**.color = color;

}

**public** String getColor() {

**return** color;

}

**public** **float** getLength() {

**return** length;

}

**public** **int** getSerialNumber() {

**return** serialNumber;

}

**public** BoardType getBoardType() {

**return** boardType;

}

}

**השלימו את המחלקה Surferעפ"י הנדרש בהגדרות למעלה:**

**public** **class** Surfer {

**private** String name;

**private** String country;

**private** **int** worldRank;

**private** SurfBoard[] surfBoards;

**private** **int** numberOfBoards;

**private** **final** **int** MAX\_NUMBER\_OF\_SURBOARDS = 20;

**public** String getName() {

**return** name;

}

@Override

**public** String toString() {

StringBuffer sb = **new** StringBuffer();

sb.append("Name: " + name + "(" + country + "), ");

sb.append("World Rank: " + worldRank + ", ");

sb.append("Surboards:\n");

**for** (**int** index = 0; index < numberOfBoards; index++) {

sb.append("\t" + surfBoards[index].toString() + "\n");

}

**return** sb.toString();

}

**public** **boolean** hasSurfBoardType(SurfBoard.BoardType boardType) {

**for** (**int** index = 0; index < numberOfBoards; index++) {

**if**(surfBoards[index].getBoardType() == boardType) {

**return** **true**;

}

}

**return** **false**;

}

**public** SurfBoard findSurfBoardById(**int** serialNumber) {

**for** (**int** index = 0; index < numberOfBoards; index++) {

**if**(surfBoards[index].getSerialNumber() == serialNumber) {

**return** surfBoards[index];

}

}

**return** **null**;

}

**public** Surfer(String name, String country, **int** worldRank) {

**this**.name = name;

**this**.country = country;

**this**.worldRank = worldRank;

numberOfBoards = 0;

surfBoards = **new** SurfBoard[MAX\_NUMBER\_OF\_SURBOARDS];

}

**public** **boolean** addSurfBoard(SurfBoard surfBoard) {

**if**(numberOfBoards == MAX\_NUMBER\_OF\_SURBOARDS) {

**return** **false**;

}

SurfBoard found = findSurfBoardById(surfBoard.getSerialNumber());

**if** (found != **null**)

**return** **false**;

surfBoards[numberOfBoards++] = surfBoard;

**return** **true**;

}

**public** SurfBoard getLongestSurfboard() {

**if**(numberOfBoards == 0) {

**return** **null**;

}

**int** selectedIndex = 0;

**for** (**int** index = 1; index < numberOfBoards; index++) {

**if**(surfBoards[index].getLength() > surfBoards[selectedIndex].getLength()) {

selectedIndex = index;

}

}

**return** surfBoards[selectedIndex];

**השלימו את המחלקה WSLעפ"י הנדרש בהגדרות למעלה:**

**public** **class** WSL {

**private** Surfer[] surfers;

**private** String locationOfLastSurfingCompetition;

**private** **final** **int** MAX\_NUMBER\_OF\_SURFERS = 50;

**private** **int** currentNumberOfSurfers;

**public** WSL(String locationOfLastSurfingCompetition) {

setLocationOfLastSurfingCompetition(locationOfLastSurfingCompetition);

surfers = **new** Surfer[MAX\_NUMBER\_OF\_SURFERS];

}

**public** String getLocationOfLastSurfingCompetition() {

**return** locationOfLastSurfingCompetition;

}

**public** **void** setLocationOfLastSurfingCompetition(

String locationOfLastSurfingCompetition) {

**this**.locationOfLastSurfingCompetition =

locationOfLastSurfingCompetition;

}

**public** **boolean** addSurfer(Surfer surfer) {

**if**(currentNumberOfSurfers == MAX\_NUMBER\_OF\_SURFERS) {

**return** **false**;

}

surfers[currentNumberOfSurfers++] = surfer;

**return** **true**;

}

@Override

**public** String toString() {

StringBuffer sb = **new** StringBuffer();

sb.append("Last location was " + locationOfLastSurfingCompetition);

sb.append(", and the surfurs were:\n");

**for** (**int** i=0 ; i < currentNumberOfSurfers ; i++)

sb.append(surfers[i].toString());

**return** sb.toString();

}

**public** Surfer findSurferWithLongestSurfBoard() {

**if**(currentNumberOfSurfers == 0) {

**return** **null**;

}

**int** selectedIndex = 0;

**for** (**int** index = 1; index < currentNumberOfSurfers; index++) {

**if**(surfers[index].getLongestSurfboard().getLength() > surfers[selectedIndex].getLongestSurfboard().getLength()) {

selectedIndex = index;

}

}

**return** surfers[selectedIndex];

}

**public** **int** getNumberOfSurfersHavingBoardType(SurfBoard.BoardType boardType) {

**int** counter = 0;

**for** (**int** index = 0; index < currentNumberOfSurfers; index++) {

**if**(surfers[index].hasSurfBoardType(boardType)) {

counter++;

}

}

**return** counter;

}

**public** SurfBoard.BoardType findTheMostUsedBoardType(){

SurfBoard.BoardType[] boardTypes = SurfBoard.BoardType.*values*();

**int** selectedNumberOfSurfers = getNumberOfSurfersHavingBoardType(boardTypes[0]);

SurfBoard.BoardType selectedBoardType = boardTypes[0];

**for** (**int** index = 1; index < boardTypes.length; index++) {

**int** numberOfSurfers = getNumberOfSurfersHavingBoardType(boardTypes[index]);

**if**(numberOfSurfers > selectedNumberOfSurfers) {

selectedBoardType = boardTypes[index];

selectedNumberOfSurfers = numberOfSurfers;

}

}

**return** selectedBoardType;

}

**public** Surfer findSurferByBoardSerialNumber(**int** serialNumber) {

**if**(currentNumberOfSurfers == 0) {

**return** **null**;

}

**for** (**int** index = 0; index < currentNumberOfSurfers; index++) {

**if**(surfers[index].findSurfBoardById(serialNumber) != **null**) {

**return** surfers[index];

}

}

**return** **null**;

}

**שאלה 2 (25 נקודות):**

**חלק א' (10 נקודות)**

להלן שתי פונקציות רקורסיביות:

**public** **static** **void** what(**int** a, **int** b) {

**if** (b == 0) {

System.***out***.print(1 + " ");

**return**;

}

*what*(a, b-1);

System.***out***.print(*what2*(a, b) + " ");

}

**public** **static** **int** what2(**int** x, **int** y) {

**if** (y == 0)

**return** 1;

**return** x\**what2*(x, y-1);

}

1. **(7 נקודות)** ציירו עץ מעקב עבור הפונקציה what בלבד כאשר a=2, b=4 וכתבו מה יהיה הפלט עבור זימון הפונקציה עם ערכים אלו.
2. **(3 נקודות)** כתבו במשפט מה הפונקציה עושה. (שימו לב, אין להסביר מהן הפקודות המבוצעות או לתת דוגמת הרצה - כלומר אין להסביר את ה"איך").

**פתרון:**

A picture containing text, whiteboard

Description automatically generated

**חלק ב' (15 נקודות)**

"פאזל N-מלכות" היא בעיה שבה אנו ממקמים N מלכות שחמט על לוח בגודל NxN כך ששום מלכה לא מאיימת על האחרת.

"מלכה מאיימת" היא מלכה הנמצאת באותה שורה או באותה עמודה.

לדוגמא, עבור לוח שחמט סטנדרטי בגודל 8X8 להלן פתרון תקין:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q |  |  |  |  |  |  |  |
|  |  |  |  | Q |  |  |  |
|  |  |  |  |  |  |  | Q |
|  |  |  |  |  | Q |  |  |
|  |  | Q |  |  |  |  |  |
|  |  |  |  |  |  | Q |  |
|  | Q |  |  |  |  |  |  |
|  |  |  | Q |  |  |  |  |

כתבו את הפעולה הבאה:

**public** **static** **boolean** isNQueen(**char**[][] chessBoard, **int** vertexRow, **int** vertexColumn)

הפעולה מקבלת מטריצה ריבועית של תווים, אינדקס של שורה, ואינדקס של עמודה. כל תו בו יש מלכה ישנו התו Q.

עליכם להחזיר true במידה והלוח מקיים את חוקי פאזל N-מלכות, אחרת החזירו false.

**הערה חשובה! מותר להשתמש בלולאות אך בשום פנים ואופן אסור להשתמש בלולאות מקוננות (לולאה בתוך לולאה).**

**פתרון עם לולאות מקוננות יפסל.**

**פתרון:**

**public** **static** **boolean** isNQueen(**char**[][] chessBoard, **int** vertexRow, **int** vertexColumn)

**שאלה 3 (25 נקודות):**

כתבו את הפונקציה הבאה:

**public** **static** **int** replaceOddDigitsWith0(**int** number)

הפונקציה מקבלת מספר, ומחזירה מספר חדש שהוא המספר שהתקבל אבל כל ספרה אי-זוגית מוחלפת בספרה 0. אם הספרה השמאלית היא אי-זוגית, היא כמובן אינה חלק מהמספר החדש.

**דוגמאות:**

קריאה לפונקציה עם המספר 23456 תחזיר את המספר 20406.

קריאה לפונקציה עם המספר 12345 תחזיר את המספר 2040.

קריאה לפונקציה עם המספר 111454 תחזיר את המספר 404.

**פתרון:**

**public** **static** **int** replaceOddDigitsWith0(**int** number) {

**int** newNumber = 0;

**int** location = 1;

**while** (number > 0) {

**int** digit = number%10;

**if** (digit%2 == 0) {

digit \*= location;

newNumber += digit;

}

location \*= 10;

number /= 10;

}

**return** newNumber;

}

**שאלה 4 (25 נקודות):**

הגדרה: במטריצה "ריבוע של איברים" הם כל ארבעה איברים העונים על האינדקסים הבאים:

([i,j], [i, j+1], [i+1, j], [i+1, j+1]).

כתבו את הפונקציה הבאה:

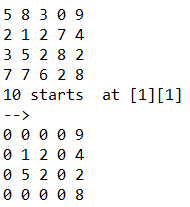
**public** **static** **boolean** hasSquareWithSum(**int**[][] mat, **int** sum)

הפונקציה מקבלת מטריצה של מספרים ומספר. הפונקציה תחזיר true אם קיים ריבוע של איברים שסכומם הוא כמספר, אחרת הפונקציה תחזיר false.

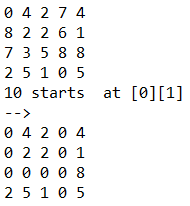
במידה והפונקציה מצאה כזה ריבוע, היא תחליף את כל האיברים שסביב הריבוע להיות עם הערך 0.

להלן דוגמאות למטריצות שחיפשנו בהן ריבוע של איברים עם סכום 10. הדוגמאות מדפיסות את האינדקסים של הפינה השמאלית-עליונה של ריבוע שסכומו 10, ולאחר מכן את המטריצה לאחר שעטפנו את איברי הריבוע עם הספרה 0:

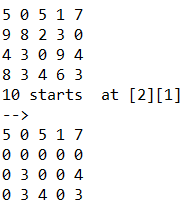
דוגמה 1:



דוגמה 2:



דוגמה 3:



**שימו לב**: בפתרון לא לצאת מגבולות המטריצה.

**מותר וכדאי** לכתוב פונקציות עזר.

**פתרון:**

**public** **static** **void** replaceBorderWith0(**int**[][] mat, **int** row, **int** col) {

**if** (row > 0) { // top row

mat[row-1][col] = mat[row-1][col+1] = 0;

}

**if** (col > 0) { // left col

mat[row][col-1] = mat[row+1][col-1] = 0;

}

**if** (row < mat.length-2) { // buttom row

mat[row+2][col] = mat[row+2][col+1] = 0;

}

**if** (col < mat[0].length-2) { // right col

mat[row][col+2] = mat[row+1][col+2] = 0;

}

// the corners

**if** (row > 0 && col > 0) // top-left

mat[row-1][col-1] = 0;

**if** (row > 0 && col < mat[0].length-2) // top-right

mat[row-1][col+2] = 0;

**if** (row < mat.length-2 && col > 0) // buttom-left

mat[row+2][col-1] = 0;

**if** (row < mat.length-2 && col < mat[0].length-2) // buttom-right

mat[row+2][col+2] = 0;

}

**public** **static** **boolean** hasSquareWithSum(**int**[][] mat, **int** sum) {

**for** (**int** i=0 ; i < mat.length-1 ; i++ ) {

**for** (**int** j=0 ; j < mat[i].length-1 ; j++) {

**if** (mat[i][j] + mat[i][j+1] + mat[i+1][j] + mat[i+1][j+1] == sum) {

System.***out***.println(sum + " starts at [" + i + "][" + j + "]");

*replaceBorderWith0*(mat, i, j);

**return** **true**;

}

}

}

**return** **false**;

}