**פתרון x**

**שאלה 1 (50 נקודות)**

בשאלה זו הנכם מאמני קבוצת פוטבול. כחלק מליגת ה NFL ישנו יום בשנה (Draft Day) שבו הקבוצות בוחרות שחקנים מהמכללות השונות ברחבי המדינה. הקבוצות בוחרות השחקנים השונים לפי הסדר באופן מחזורי. הקוד נמצא בחבילה question\_1.

הקבוצה (FootballTeam) מחליטה לבחור שחקן (FootballPlayer) לפי היכולות האתלטיות שלו (AthleticAbilities), התנהגות (Behavior), וכשרון המשחק שלו (GameTalentLevel).

1. (1 נק׳) במחלקה AthleticAbilities קיימות תכונות ב private. לכל תכונה במחלקה, יש לאפשר גישת קריאה מבחוץ אך **לא גישת כתיבה**.
2. (1 נק׳) יש לממש במחלקת AthleticAbilities את הפונק׳ המופיעה למטה. הפונק׳ מחשבת את ממוצע ערכי כל תכונות המחלקה: speed, verticalJump, horizontalJump, numberOfBenchPresses ומחזירה הערך המחושב.

public float calculateAverageScore(){

}

1. (3 נק׳) יש ליצור enum מטיפוס Behavior במחלקת FootballPlayer עם הערכים הבאים: TeamPlayer, LoneWolf, CoOperative, TotalMess. כאשר ערכו של TeamPlayer = 0 וכו׳ (אין צורך לציין במפורש את ערכי האיברים ב enum, ציינתי זאת רק על מנת להגדיר הסדר באופן מובהק).
2. (5 נק׳) יש לכתוב בנאי במחלקת FootballPlayer עם החתימה הבאה. יש לאתחל כל תכונה עם הפרמטר המתאים. יש ליצור תכונות במידת הצורך (אם אין לפרמטר תכונה במחלקה). החתימה כרגע בהערה מכיוון שהיא משתמשת בטיפוס Behavior, יש להוציא הקוד מהערה.

public FootballPlayer(String name, String id, String College, Behavior behavior, Position position, GameTalentLevel gameTalentLevel, AthleticAbilities athleticAbilities){

}

1. (5 נק׳) יש לממש במחלקת FootballPlayer את הפונק׳ public float calculateScore(). הפונק׳ מחשבת את ציון השחקן על ידי סכימת התכונות behavior, gameTalentLevel, הפעלת הפונק׳ calculateAverageScore של המחלקה AthleticAbilities, ומחזירה הערך המחושב.
2. (5 נק׳) יש לממש במחלקת FootballTeam את הבנאי המצוין למטה ולהשלים התכונות החסרות במחלקה בהתאם. שימו לב שאתם מקשרים בין הפרמטרים לתכונות הנכונות.

public FootballTeam(String name, int maxCapacityForDraftedFootballPlayers, FootballPlayer.Position[] missingPositions){

}

1. (5 נק׳) יש לממש במחלקת FootballTeam את הפונק׳ מטה. יש מתאם בין קבוצה לשחקן כאשר מתקיימים התנאים הבאים:
   1. יש מקום לשחקן בקבוצה
   2. העמדה של השחקן מופיעה במערך missingPositions של הקבוצה (עבור סעיף זה יש לאפשר גישת קריאה בלבד לתכונה position של השחקן)
   3. ציון השחקן עובר את סף הקבלה של הקבוצה (גדול שווה)

public boolean isFit(FootballPlayer footballPlayer){

}

1. (5 נק׳) יש לממש במחלקת FootballTeam הפונק׳ המופיעה למטה. הפונק׳ מבצעת הפעולות הבאות:
   1. מצרפת את השחקן החדש לקבוצה
   2. מעדכנת את המערך missingPositions, מסירה את העמדה שהשחקן משחק בא מהמערך. שימו לב כי יכול להיות מצב שלמערך יש כמה עמדות זהות, יש להסיר את המופע הראשון שאתם נתקלים בו. מוזמנים לעדכן המערך בכל דרךשתרצו.

public void draft(FootballPlayer footballPlayer){

}

1. (6 נק׳) יש לממש במחלקת FootballTeam את הפונק׳ המופיעה למטה. הפונק׳ תמיין את השחקנים של הקבוצה לפי הציון שלהם. אתם יכולים להשתמש באיזו שיטת מיון שבא לכם כל עוד היא עומדת בסיבוכיות זמן ריצה . חריגה מסיבוכיות זו תפסול הפתרון. אסור להשתמש בפתרונות מובנים בשפת Java, למשל, Arrays.sort().

public void sortDraftedFootballplayers(){

}

1. (5 נק׳) יש לממש במחלקה FootballTeam פונק׳ המקבלת ציון של שחקן ומחזירה מופע של FootballPlayer (הפונק׳ למטה), במידה ולא נמצא שחקן הפונק׳ תחזיר null. במידה ויש כמה שחקנים עם אותו הציון ניתן להחזיר מופע של אחד מהם. יש לממש את החיפוש בסיבוכיות זמן ריצה , חריגה מסיבוכיות זו תפסול הפתרון. ניתן לממש באופן איטרטיבי או רקורסיבי. ניתן ומומלץ להשתמש בפונק׳ עזר.

public FootballPlayer find(int footballPlayerScore){

}

1. (5 נק׳) במחלקה DraftContest קיימת פונק׳ load המקבלת נתיב לקובץ המכיל נתונים על 200 השחקנים הפוטנציאלים להיבחר ע״י קבוצה. הפונק׳ קוראת את נתוני השחקנים ומאכלסת את המערך footballPlayers ע״י שימוש בפונק׳ עזר. הקוד חלקי ולא מבצע את העבודה כנדרש. עליכם להבין מה חסר ולהשלים את הקוד על מנת שאחרי הקריאה לפונק׳, התכונה footballPlayers תכיל 200 שחקנים עם ערכים נכונים. להלן הסבר על סדר השדות עבור שחקן אחד. שדה של כל שחקן רשום בשורה חדשה והשחקנים מופרדים ע״י התו ״-״. (ממליץ להסתכל על הקובץ players.txt על מנת להתרשם ממבנה של שחקן, לא להיבהל מכמות השחקנים). כאן אני מסביר לאיזו תכונה שייכת כל שורה ולאיזו מחלקה, בקובץ אין חצים ותיאור השדות.

FootballPlayer:

Perry Silberman 🡪 name

nIe3O7fBr8 🡪 id

Community College of the Air Force 🡪 college

TeamPlayer 🡪 behavior

RunningBack 🡪 position

Low 🡪 gameTalentLevel

AthleticAbilities:

5.842129 🡪 speed

1.9684259 🡪 verticalJump

2.7754474 🡪 horizontalJump

6 🡪 numberOfBenchPresses

-

1. (3 נק׳) יש לממש במחלקה DraftContest את הפונק׳ nextPlayer(). בכל קריאה הפונק׳ תחזיר את השחקן הבא מהתכונה footballPlayers. ניתן להוסיף תכונות ופונק׳ עזר.

public FootballPlayer nextPlayer()

1. (2 נק׳) במחלקה Question1 קיים main, בסוף ה main יש לולאה שקוראת לפונק׳ המדפיסה את נתוני כל השחקנים שהיא בחרה printDetailsOfDradtedPlayers(). יש לממש את סדרת הפונק׳ הרלוונטיות בכל מחלקה (FootballTeam, FootballPlayer, AthleticAbilities) על מנת שהנתונים יוצגו על המסך באופן קריא וברור. נותן לכם יד חופשית לעשות ההדפסות איך שאתם מרגישים. שימו לב, אוריד נק׳ על הדפסה לא ברורה. למען הסר ספק, הדפסה ברורה הינה כזאת שאפשר לקשר בין מה שמודפס על המסך והערכים המקוריים בתכונות.
2. מוזמנים להריץ

**פתרון**

**package** question\_1;

**import** java.io.PrintWriter;

**public** **class** AthleticAbilities {

**private** **float** speed;

**private** **float** verticalJump;

**private** **float** horizontalJump;

**private** **int** numberOfBenchPresses;

**public** AthleticAbilities(**float** speed, **float** verticalJump, **float** horizontalJump, **int** numberOfBenchPresses) {

**this**.speed = speed;

**this**.verticalJump = verticalJump;

**this**.horizontalJump = horizontalJump;

**this**.numberOfBenchPresses = numberOfBenchPresses;

}

**public** **float** calculateAverageScore() {

**float** sum = speed + verticalJump + horizontalJump + numberOfBenchPresses;

**float** average = sum / 4;

**return** average;

}

**public** **float** getSpeed() {

**return** speed;

}

**public** **float** getVerticalJump() {

**return** verticalJump;

}

**public** **float** getHorizontalJump() {

**return** horizontalJump;

}

**public** **int** getNumberOfBenchPresses() {

**return** numberOfBenchPresses;

}

**public** **void** printDetails() {

System.***out***.println(speed);

System.***out***.println(verticalJump);

System.***out***.println(horizontalJump);

System.***out***.println(numberOfBenchPresses);

}

**public** **void** save(PrintWriter printWritter) {

printWritter.println(speed);

printWritter.println(verticalJump);

printWritter.println(horizontalJump);

printWritter.println(numberOfBenchPresses);

}

}

**package** question\_1;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Scanner;

**import** question\_1.AthleticAbilities;

**import** question\_1.FootballPlayer;

**public** **class** DraftContest {

**public** **final** **int** NUMBER\_OF\_PLAYERS = 200;

**private** FootballPlayer[] footballPlayers;

**private** **int** nextPlayer = 0;

**public** DraftContest() {

footballPlayers = **new** FootballPlayer[NUMBER\_OF\_PLAYERS];

}

**public** **void** load(String filePath) **throws** FileNotFoundException {

File file = **new** File(filePath);

Scanner scanner = **new** Scanner(file);

**for**(**int** index = 0; index < NUMBER\_OF\_PLAYERS; index++) {

FootballPlayer footballPlayer = loadFootballPlayer(scanner);

footballPlayers[index] = footballPlayer;

scanner.nextLine(); // reading the delimiter line

scanner.nextLine(); // escaping scanner

}

scanner.close();

}

**public** FootballPlayer nextPlayer() {

**return** footballPlayers[nextPlayer++];

}

**private** FootballPlayer loadFootballPlayer(Scanner scanner) {

String name = scanner.nextLine();

String id = scanner.nextLine();

String college = scanner.nextLine();

FootballPlayer.Behavior behavior = FootballPlayer.Behavior.*valueOf*(scanner.nextLine());

FootballPlayer.Position position = FootballPlayer.Position.*valueOf*(scanner.nextLine());

FootballPlayer.GameTalentLevel gameTalentLevel = FootballPlayer.GameTalentLevel.*valueOf*(scanner.nextLine());

AthleticAbilities athleticAbilities = loadAthleticAbilities(scanner);

FootballPlayer footballPlayer = **new** FootballPlayer(name, id, college, behavior, position, gameTalentLevel, athleticAbilities);

**return** footballPlayer;

}

**private** AthleticAbilities loadAthleticAbilities(Scanner scanner) {

**float** speed = scanner.nextFloat();

**float** verticalJump = scanner.nextFloat();

**float** horizontalJump = scanner.nextFloat();

**int** numberOfBenchPresses = scanner.nextInt();

AthleticAbilities athleticAbilities = **new** AthleticAbilities(speed, verticalJump, horizontalJump, numberOfBenchPresses);

**return** athleticAbilities;

}

}

**package** question\_1;

**import** java.io.PrintWriter;

**public** **class** FootballPlayer {

**public** **enum** Behavior {***TeamPlayer***, ***LoneWolf***, ***CoOperative***, ***TotalMess***}

**public** **enum** GameTalentLevel {***Low***, ***Average***, ***Good***, ***Talent***, ***Star***}

**public** **enum** Position {***Quarterback***, ***RunningBack***, ***WideReceiver***, ***TightEnd***, ***OffensiveLine***, ***Center***, ***Guard***, ***OffensiveTackle***,

***DefensiveEnd***, ***DefensiveTackle***, ***Linebacker***, ***Cornerback***, ***Safety***}

**private** String name;

**private** String id;

**private** String college;

**private** Behavior behavior;

**private** Position position;

**private** AthleticAbilities athleticAbilities;

**private** GameTalentLevel gameTalentLevel;

**public** FootballPlayer(String name, String id, String college, Behavior behavior, Position position, GameTalentLevel gameTalentLevel, AthleticAbilities athleticAbilities) {

**this**.name = name;

**this**.id = id;

**this**.college = college;

**this**.behavior = behavior;

**this**.position = position;

**this**.athleticAbilities = athleticAbilities;

**this**.gameTalentLevel = gameTalentLevel;

}

**public** **float** calculateScore() {

**float** athleticAbilitiesAverageScore = athleticAbilities.calculateAverageScore();

**float** score = behavior.ordinal() + athleticAbilitiesAverageScore + gameTalentLevel.ordinal();

**return** score;

}

**public** Position getPosition() {

**return** position;

}

**public** String getId() {

**return** id;

}

**public** **void** printDetails() {

System.***out***.println(name);

System.***out***.println(id);

System.***out***.println(college);

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

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

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

athleticAbilities.printDetails();

}

**public** **void** save(PrintWriter printWritter) {

printWritter.println(name);

printWritter.println(id);

printWritter.println(college);

printWritter.println(behavior.toString());

printWritter.println(position.toString());

printWritter.println(gameTalentLevel.toString());

athleticAbilities.save(printWritter);

}

}

**package** question\_1;

**import** question\_1.FootballPlayer;

**import** question\_1.FootballPlayer.Position;

**public** **class** FootballTeam {

**private** String name;

**private** FootballPlayer[] draftedFootballplayers;

**private** FootballPlayer.Position[] missingPositions;

**private** **int** numberOfDraftedPlayers;

**private** **int** playerScoreThreshold;

**public** FootballTeam(String name, **int** maxCapacityForDraftedFootballPlayers, FootballPlayer.Position[] missingPositions, **int** playerScoreThreshold) {

**this**.name = name;

draftedFootballplayers = **new** FootballPlayer[maxCapacityForDraftedFootballPlayers];

**this**.missingPositions = missingPositions;

**this**.playerScoreThreshold = playerScoreThreshold;

numberOfDraftedPlayers = 0;

}

**public** **boolean** isFit(FootballPlayer footballPlayer) {

**float** footballPlayerScore = footballPlayer.calculateScore();

FootballPlayer.Position position = footballPlayer.getPosition();

**if**(numberOfDraftedPlayers >= draftedFootballplayers.length) {

**return** **false**;

}

**if**(footballPlayerScore < playerScoreThreshold) {

**return** **false**;

}

**for**(FootballPlayer.Position missingPosition : missingPositions) {

**if**(missingPosition == position) {

**return** **true**;

}

}

**return** **false**;

}

**public** **void** draft(FootballPlayer footballPlayer) {

draftedFootballplayers[numberOfDraftedPlayers] = footballPlayer;

numberOfDraftedPlayers++;

removeMissingPostion(footballPlayer.getPosition());

}

**public** **void** sortDraftedFootballplayers() {

**int** length = numberOfDraftedPlayers;

**for** (**int** index = 0; index < length - 1; index++){

**for** (**int** indexB = 0; indexB < length - index - 1; indexB++){

**if** (draftedFootballplayers[indexB].calculateScore() > draftedFootballplayers[indexB + 1].calculateScore()){

swap(indexB, indexB + 1);

}

}

}

}

**public** **void** printDetailsOfDradtedPlayers() {

System.***out***.println(name);

System.***out***.println(numberOfDraftedPlayers);

System.***out***.println(playerScoreThreshold);

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

draftedFootballplayers[index].printDetails();

}

}

**public** FootballPlayer find(**int** footballPlayerScore) {

FootballPlayer footballPlayer = internalFind(footballPlayerScore, 0, draftedFootballplayers.length);

**return** footballPlayer;

}

**private** FootballPlayer internalFind(**int** footballPlayerScore, **int** lowBoundry, **int** highBoundry) {

**if**(lowBoundry > highBoundry) {

**return** **null**;

}

**int** middle = lowBoundry + (highBoundry - lowBoundry) / 2;

**if**(draftedFootballplayers[middle].calculateScore() == footballPlayerScore) {

**return** draftedFootballplayers[middle];

}

**if**(draftedFootballplayers[middle].calculateScore() == footballPlayerScore) {

**return** internalFind(footballPlayerScore, middle + 1, highBoundry);

}

**return** internalFind(footballPlayerScore, lowBoundry, middle - 1);

}

**private** **void** swap(**int** indexA, **int** indexB) {

FootballPlayer temp = draftedFootballplayers[indexA];

draftedFootballplayers[indexA] = draftedFootballplayers[indexB];

draftedFootballplayers[indexB] = temp;

}

**private** **void** removeMissingPostion(Position positionToRemove) {

FootballPlayer.Position[] newMissingPositions = **new** FootballPlayer.Position[missingPositions.length - 1];

**boolean** isSkipped = **false**;

**for**(**int** index = 0, newMissingPositionsIndex = 0; index < missingPositions.length; index++) {

**if**(missingPositions[index] != positionToRemove && !isSkipped) {

newMissingPositions[newMissingPositionsIndex] = missingPositions[index];

newMissingPositionsIndex++;

isSkipped = **true**;

}

}

missingPositions = newMissingPositions;

}

}

**package** question\_1;

**import** java.io.FileNotFoundException;

**import** question\_1.DraftContest;

**import** question\_1.FootballPlayer;

**import** question\_1.FootballTeam;

**public** **class** Question1 {

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

**final** String playerFilePath = "src/question\_1/players.txt";

DraftContest draftContest = **new** DraftContest();

draftContest.load(playerFilePath);

FootballTeam[] footballTeams = *loadFootballTeams*();

**int** numberOfPlayers = draftContest.NUMBER\_OF\_PLAYERS;

**int** teamIndex = 0;

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

FootballPlayer footballPlayer = draftContest.nextPlayer();

FootballTeam footballTeam = footballTeams[teamIndex];

teamIndex = teamIndex == footballTeams.length - 1 ? 0 : teamIndex + 1;

**boolean** isFit = footballTeam.isFit(footballPlayer);

**if**(!isFit) {

**continue**;

}

footballTeam.draft(footballPlayer);

}

**for** (FootballTeam footballTeam : footballTeams) {

footballTeam.sortDraftedFootballplayers();

}

**for** (FootballTeam footballTeam : footballTeams) {

footballTeam.printDetailsOfDradtedPlayers();

}

}

**private** **static** FootballTeam[] loadFootballTeams() {

FootballPlayer.Position[] RamsMissingPositions = **new** FootballPlayer.Position[] {FootballPlayer.Position.***Quarterback***, FootballPlayer.Position.***OffensiveLine***, FootballPlayer.Position.***DefensiveTackle***, FootballPlayer.Position.***Linebacker***, FootballPlayer.Position.***OffensiveTackle***, FootballPlayer.Position.***TightEnd***};

FootballTeam rams = **new** FootballTeam("Los Angeles Rams", 6, RamsMissingPositions, 7);

FootballPlayer.Position[] eaglesMissingPositions = **new** FootballPlayer.Position[] {FootballPlayer.Position.***DefensiveTackle***, FootballPlayer.Position.***DefensiveTackle***, FootballPlayer.Position.***DefensiveEnd***, FootballPlayer.Position.***OffensiveLine***, FootballPlayer.Position.***TightEnd***, FootballPlayer.Position.***WideReceiver***, FootballPlayer.Position.***Cornerback***};

FootballTeam eagles = **new** FootballTeam("Philadelphia Eagles", 7, eaglesMissingPositions, 5);

FootballPlayer.Position[] raidersMissingPositions = **new** FootballPlayer.Position[] {FootballPlayer.Position.***TightEnd***, FootballPlayer.Position.***Quarterback***, FootballPlayer.Position.***Center***, FootballPlayer.Position.***OffensiveLine***};

FootballTeam raiders = **new** FootballTeam("Oaklan Raiders", 4, raidersMissingPositions, 3);

FootballTeam[] footballTeams = **new** FootballTeam[] {raiders, eagles, rams};

**return** footballTeams;

}

}

**שאלה 2 (30 נק׳)**

מילה נחשבת elfish אם היא מכילה האותיות e, l, f ללא חשיבות לסדר. לדוגמא, המילים הבאות נחשבות elfish: whiteleaf, tasteful, unfriendly, waffles, כיוון שהן מכילות האותיות האלו. אין לשנות את חתימת הפונק׳, להוסיף תכונות, או להשתמש בלולאות. חריגה מהכללים תגרור פסילת הפתרון. יש לכתוב הפתרון בקובץ Question2.java תחת החבילה question\_2.

1. (15 נק׳) יש לכתוב פונק׳ רקורסיבית עם החתימה מטה. כך שבהינתן מילה הפונק׳ תחזיר true אם המילה מקיימת התנאים, ו false אם לא.

דוגמא: עבור המילה tasteful הפונק׳ תחזיר true. עבור המילה tastful הפונק׳ תחזיר false.

public static boolean isElfish(String word){

return *isContainsChars*(word, "elf");

}

private static boolean isContainsChars(String word, String checker){

}

1. (15 נק׳) יש לכתוב פונק׳ רקורסיבית כך שבהינתן שתי מילים הפונק׳ מחזירה true אם המילה הראשונה (origin) היא elfish וכל האותיות שלה מופיעות במילה השניה (compare).

דוגמא: עבור המילה tasteful והמילה tastefulness הפונק׳ תחזיר true. עבור המילה tasteful והמילה tasteulness הפונק׳ תחזיר false כי חסר את האות f.

Public static boolean isXish(String origin, String compare){

}

**package** question\_2;

**public** **class** Question2 {

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

System.***out***.println(*isElfish*("whiteleaf"));

System.***out***.println(*isXish*("whiteleaf", "whiteleaf"));

}

**public** **static** **boolean** isElfish(String word) {

**return** *isContainsChars*(word, "elf");

}

**private** **static** **boolean** isContainsChars(String word, String checker) {

**if**(checker.length() == 0) {

**return** **true**;

} **else** **if**(word.length() == 0) {

**return** **false**;

}

**if**(checker.contains(word.charAt(0) + "")) {

**return** *isContainsChars*(word.substring(1), checker.replace(word.charAt(0) + "", ""));

}

**return** *isContainsChars*(word.substring(1), checker);

}

**public** **static** **boolean** isXish(String origin, String compare) {

**return** *isElfish*(origin) && *isContainsChars*(compare, origin);

}

}

**שאלה 3 (20 נק׳)**

הפונקציה מקבלת כפרמטר מטריצה בגודל N\*N ומספר שלם size. הפונקציה תחפש במטריצה רצף של מספרים זהים באורך size אנכית ואופקית (לא באלכסון). הפונקציה תדפיס את המיקומים במטריצה שבהם מתחיל רצף כזה, כלומר תדפיס זוג מספרים שהם הקואורדינטות של תחילת הרצף. יש לכתוב הפתרון בקובץ Question3.java תחת החבילה question\_3.

**public** **static** **void** findDigitsInRowOrColumn(**int**[][] mat, **int** size)

דוגמא: עבור מטריצה הבא כאשר N=5 ו size=3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 1 | 0 |  |
| **1** | 4 | 3 | **2** | 1 | 0 |
| **1** | **2** | **2** | **2** | 0 | 1 |
| **1** | 0 | 9 | **2** | 1 | 2 |
| **1** | 4 | 2 | 0 | 2 | 3 |
| **1** | **1** | **1** | 2 | 1 | 4 |

יודפסו הקואורדינטות:

point : 0,1

point : 0,4

point : 1,1

point : 1,4

point : 2,4

point : 4,2

כל נקודה כזו מהווה התחלה של רצף אופקי או אנכי באורך 3

**package** question\_3;

**public** **class** Question3 {

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

**int**[][] matrix = {

{ 1, 2, 3, 4 ,1},

{ 0, 2, 2, 2 ,1},

{ 1, 2, 9, 0 ,1},

{ 2, 0, 2, 4 ,1},

{ 1, 2, 1, 1 ,1}};

*findDigitsInRowOrColumn*(matrix, 3);

}

**public** **static** **void** findDigitsInRowOrColumn(**int**[][] mat, **int** size){

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

**for** (**int** j = 0; j < mat.length; j++) {

//check to right

**for** (**int** k = 1; k <= size && j + k < mat.length; k++) {

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

**if** (k == size-1) {

System.***out***.println("point : " + i + "," + j);

**break**;

}

}**else** {

**break**;

}

}

//check to bottom

**for** (**int** k = 1; k <= size && i + k < mat.length; k++) {

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

**if** (k == size-1) {

System.***out***.println("point : " + i + "," + j);

**break**;

}

}**else** {

**break**;

}

}

}

}

}

}