

**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 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 מרוץ המכוניות Formula 1 (F1) השכיר את שירותיכם על מנת שתתחזקו עבורו מערכת לניהול אליפות העולם במרוץ מכוניות.

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 רכב (Car):
   1. תכונות המחלקה: צבע, מהירות מרבית, וסוג.
   2. בנאי המקבל את כל התכונות ומאתחל את האובייקט.
   3. **(5 נק') הוסיפו** למחלקה תמיכה במספר סידורי אוטומטי שיינתן עם יצירת האובייקט. המספר הסידורי יתחיל מ- 1000.
   4. מתודה toString המחזירה מחרוזת עם נתוני המכונית כפי שמופיע בפלט בהמשך.
2. נתונה בהמשך באופן חלקי מחלקה עבור נהג (Driver):
   1. תכונות המחלקה הן שם הנהג, מאיזו מדינה הוא, הדירוג העולמי שלו ומערך הרכבים שיש לו. המספר המקסימלי של רכבים שיש לנהג הוא 2. מותר להוסיף תכונות נוספות במידת הצורך
   2. נתונה שיטה המקבלת מספר סידורי של רכב, במידה ולנהג יש רכב עם מספר סידורי זה השיטה תחזיר הפניה לרכב, אחרת תחזיר null
   3. נתונה שיטה המקבלת סוג של רכב, ומחזירה true אם לנהג יש לפחות רכב אחד מדגם זה, אחרת מחזירה false
   4. נתונה השיטה toString המחזירה מחרוזת עם נתוני הנהג
   5. **(5 נק') כתבו** בנאי המקבל את שם הנהג, המדינה אליה הוא משויך, והדירוג הנוכחי שלו, ותאתחל את כל שדות האובייקט
   6. **(5 נק') כתבו** שיטה המוסיפה לנהג רכב. השיטה תוסיף את הרכב במידה ויש מקום במערך הרכבים ולנהג עדיין אין רכב עם מספר סידורי זה. במקרה של הכנסה השיטה תחזיר true, אחרת תחזיר false
   7. **(5 נק') כתבו** שיטה המחזירה את הרכב המהיר ביותר שיש לנהג. במידה ולנהג עדיין אין רכבים, יש להחזיר null.
3. נתונה באופן חלקי מחלקה עבור Formula1
   1. תכונות המחלקה: מערך הנהגים הרשומים באיגוד ומיקום האליפות האחרונה שהאיגוד קיים
   2. נתון בנאי המקבל את המיקום האחרון שבו התקיימה האליפות
   3. נתונה שיטה המעדכנת את המיקום האחרון בו התקיימה האליפות
   4. נתונה שיטה המוסיפה נהג לאיגוד
   5. נתונה שיטה toString המחזירה מחרוזת עם נתוני האיגוד
   6. **(5 נק')** כתבו שיטה המחזירה את הנהג שיש לו את הרכב הכי מהיר, אם אין נהגים אז השיטה תחזיר null
   7. **(5 נק')** כתבו שיטה המקבלת סוג מסוים של רכב ומחזירה לכמה נהגים יש רכב מסוג זה
   8. **(10 נק')** כתבו שיטה המחזירה את סוג הרכב שיש להכי הרבה נהגים
   9. **(5 נק')** כתבו שיטה המקבלת מספר סידורי של רכב ומחזירה את הנהג אליו שייך. במידה ואינו שייך לאף נהג השיטה תחזיר null

להלן main לדוגמה:

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

Car c1 = **new** Car("White", Car.Type.***AlfaRomeo***, 450);

Car c2 = **new** Car("Red", Car.Type.***Ferrari***, 500);

Car c3 = **new** Car("Blue", Car.Type.***McLaren***, 480);

Car c4 = **new** Car("Yellow", Car.Type.***Renault***, 470);

Car c5 = **new** Car("Red", Car.Type.***RedBullRacing***, 440);

Car c6 = **new** Car("Red", Car.Type.***RedBullRacing***, 430);

Car c7 = **new** Car("Pink", Car.Type.***RacingPoint***, 430);

Driver d1 = **new** Driver("Charles Lecrec", 4, "Monaco");

Driver d2 = **new** Driver("Antonio Giovinazzi", 17, "Italy");

Driver d3 = **new** Driver("Carlos Sainz", 6, "Spain");

Driver d4 = **new** Driver("Daniel Ricciardo", 9, "Australia");

Driver d5 = **new** Driver("Max Verstappen", 3, "Netherlands");

Driver d6 = **new** Driver("Sergio Perez", 10, "Mexico");

d1.addCar(c2);

d2.addCar(c1);

d3.addCar(c3);

d4.addCar(c4);

d4.addCar(c6);

d5.addCar(c5);

d6.addCar(c7);

Formula1 f1 = **new** Formula1("Formula 1 Gran Premio Heineken Dג€™italia");

f1.addDriver(d1);

f1.addDriver(d2);

f1.addDriver(d4);

f1.addDriver(d5);

f1.addDriver(d6);

f1.addDriver(d3);

System.***out***.println(f1.toString());

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

Driver DriverBySerialNumber = f1.findDriver(1005);

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

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

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

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

DriverBySerialNumber = f1.findDriver(1200);

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

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

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

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

Driver Driver = f1.findDriverWithTheFastestCar();

System.***out***.println("The Driver who has the fastest Car is:");

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

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

Car.Type carType = f1.findTheMostUsedCarType();

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

}

להלן פלט עבור main זה:

Location of the last competition: Formula 1 Gran Premio Heineken Dג€™italia

Name: Charles Lecrec

Rank: 4

Country: Monaco

Cars:

1) Color: Red, Type: Ferrari

Name: Antonio Giovinazzi

Rank: 17

Country: Italy

Cars:

1) Color: White, Type: AlfaRomeo

Name: Daniel Ricciardo

Rank: 9

Country: Australia

Cars:

1) Color: Yellow, Type: Renault

2) Color: Red, Type: RedBullRacing

Name: Max Verstappen

Rank: 3

Country: Netherlands

Cars:

1) Color: Red, Type: RedBullRacing

Name: Sergio Perez

Rank: 10

Country: Mexico

Cars:

1) Color: Pink, Type: RacingPoint

Name: Carlos Sainz

Rank: 6

Country: Spain

Cars:

1) Color: Blue, Type: McLaren

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

The Driver with Car with the serial number 1005:

Daniel Ricciardo

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

The Driver with Car with the serial number 1200:

-

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

The Driver who has the fastest Car is:

Charles Lecrec

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

The most used car type is RedBullRacing

**השלימו את המחלקה Car עפ"י הנדרש בהגדרות למעלה:**

**public** **class** Car {

**public** **enum** Type{***AlfaRomeo***, ***Ferrari***, ***WilliamsRacing***, ***McLaren***, ***RacingPoint***,

***MercedesAMG***, ***RedBullRacing***, ***Renault***, ***ToroRosso***, ***Haas***}

**private** String color;

**private** Type type;

**private** **int** maxSpeed;

**private** **int** serialNumber; // SOLUTION

**private** **static** **int** *globalSerialNumber* = 1000; // SOLUTION

// SOLUTION

**public** Car(String color, Type type, **int** maxSpeed) {

**this**.color = color;

**this**.type = type;

**this**.maxSpeed = maxSpeed;

serialNumber = *globalSerialNumber*++;

}

**public** **int** getMaxSpeed() {

**return** maxSpeed;

}

**public** String getColor() {

**return** color;

}

**public** Type getType() {

**return** type;

}

@Override

**public** String toString() {

StringBuffer stringBuffer = **new** StringBuffer();

stringBuffer.append("Color: " + color + ", ");

stringBuffer.append("Type: " + type.toString());

**return** stringBuffer.toString();

}

// SOLUTION

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

**return** serialNumber;

}

}

**השלימו את המחלקה Driver עפ"י הנדרש בהגדרות למעלה:**

**public** **class** Driver {

**private** String name;

**private** **int** rank;

**private** String country;

**private** Car cars[];

**private** **final** **int** MAX\_NUMBER\_OF\_CARS = 2;

**private** **int** numberOfCars;

// SOLUTION

**public** Driver(String name, **int** rank, String country) {

**this**.name = name;

**this**.rank = rank;

**this**.country = country;

**this**.cars = **new** Car[MAX\_NUMBER\_OF\_CARS];

numberOfCars = 0;

}

**public** Car getCarBySerialNumber(**int** serialNumber) {

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

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

**return** cars[index];

}

}

**return** **null**;

}

**public** **boolean** isDrivingThisType(Car.Type type){

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

**if**(cars[index].getType() == type) {

**return** **true**;

}

}

**return** **false**;

}

// SOLUTION

**public** **boolean** addCar(Car car) {

**if**(numberOfCars >= MAX\_NUMBER\_OF\_CARS) {

**return** **false**;

}

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

**if**(cars[index].getSerialNumber() == car.getSerialNumber()) {

**return** **false**;

}

}

cars[numberOfCars++] = car;

**return** **true**;

}

// SOLUTION

**public** Car getFastestCar() {

Car fastestCar = **null**;

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

**if**(fastestCar == **null** || fastestCar.getMaxSpeed() < cars[index].getMaxSpeed()) {

fastestCar = cars[index];

}

}

**return** fastestCar;

}

@Override

**public** String toString() {

StringBuffer stringBuffer = **new** StringBuffer();

stringBuffer.append("Name: " + getName() + "\n");

stringBuffer.append("Rank: " + rank + "\n");

stringBuffer.append("Country: " + country + "\n");

stringBuffer.append("Cars: " + "\n");

**for** (**int** i = 0; i < numberOfCars; i++) {

stringBuffer.append(" " + (i+1) + ") " + cars[i].toString() + "\n");

}

**return** stringBuffer.toString();

}

**public** **boolean** isDrivingThisCar(**int** serialNumber) {

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

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

**return** **true**;

}

}

**return** **false**;

}

**public** String getName() {

**return** name;

}

}

**השלימו את המחלקה Formula1 עפ"י הנדרש בהגדרות למעלה:**

**public** **class** Formula1 {

**private** Driver drivers[];

**private** String locationOfLastCompetition;

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

**private** **int** currentNumberOfDrivers;

**public** Formula1(String locationOfCompetition) {

setLocationOfCompetition(locationOfCompetition);

drivers = **new** Driver[MAX\_NUMBER\_OF\_DRIVERS];

}

**public** String getLocationOfCompetition() {

**return** locationOfLastCompetition;

}

**public** **void** setLocationOfCompetition(String locationOfCompetition) {

**this**.locationOfLastCompetition = locationOfCompetition;

}

**public** **boolean** addDriver(Driver driver) {

**if**(currentNumberOfDrivers >= MAX\_NUMBER\_OF\_DRIVERS) {

**return** **false**;

}

drivers[currentNumberOfDrivers++] = driver;

**return** **true**;

}

@Override

**public** String toString() {

StringBuffer stringBuffer = **new** StringBuffer();

stringBuffer.append("Location of the last competition: " + locationOfLastCompetition + "\n");

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

stringBuffer.append(drivers[index].toString() + "\n");

}

**return** stringBuffer.toString();

}

// SOLUTION

**public** Driver findDriverWithTheFastestCar() {

Driver selectedDriver = **null**;

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

**if**(selectedDriver == **null** || drivers[index].getFastestCar().getMaxSpeed() > selectedDriver.getFastestCar().getMaxSpeed()) {

selectedDriver = drivers[index];

}

}

**return** selectedDriver;

}

// SOLUTION

**public** **int** returnTheNumberOfDriversWithTheGivenCarType(Car.Type carType) {

**int** counter = 0;

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

**if**(drivers[index].isDrivingThisType(carType)) {

counter++;

}

}

**return** counter;

}

// SOLUTION

**public** Car.Type findTheMostUsedCarType(){

Car.Type[] carTypes = Car.Type.*values*();

**int** selectedNumberOfDrivers = returnTheNumberOfDriversWithTheGivenCarType(carTypes[0]);

Car.Type selectedCarType = carTypes[0];

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

**int** numberOfSurfers = returnTheNumberOfDriversWithTheGivenCarType(carTypes[index]);

**if**(numberOfSurfers > selectedNumberOfDrivers) {

selectedCarType = carTypes[index];

selectedNumberOfDrivers = numberOfSurfers;

}

}

**return** selectedCarType;

}

// SOLUTION

**public** Driver findDriver(**int** serialNumber) {

**if**(currentNumberOfDrivers == 0) {

**return** **null**;

}

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

**if**(drivers[index].isDrivingThisCar(serialNumber)) {

**return** drivers[index];

}

}

**return** **null**;

}

}

**שאלה 2 (25 נקודות):**

**חלק א' (10 נקודות)**

להלן פונקציה רקורסיבית:

**public** **static** **void** foo(**int** x) {

**if** (x <= 0)

**return**;

*foo*(--x);

System.***out***.printf("%d ", x);

*foo*(--x);

}

ציירו עץ מעקב עבור הפונקציה what בלבד כאשר n=4 וכתבו מה יהיה הפלט עבור זימון הפונקציה עם ערך זה.

**פתרון:**

**A close up of text on a whiteboard

Description automatically generated**

**חלק ב' (15 נקודות)**

בהינתן מחרוזת, יש להציג את כל תתי המחרוזות כך ששום תו אינו משותף בין שתי תתי מחרוזות.

דוגמאות:

עבור הקלט ABC נקבל את הפלט הבא:

{ABC}

{AB}{C}

{A}{BC}  
{A}{B}{C}

עבור הקלט ABCD נקבל את הפלט הבא:

{ABCD}

{ABC}{D}

{AB}{CD}

{AB}{C}{D}  
{A}{BCD}  
{A}{BC}{D}  
{A}{B}{CD}  
{A}{B}{C}{D}

כתבו את הפונקציה הבאה:

**public** **static** **void** breakIntoSubStringsWithoutOverlaps(

String str, **int** i, String out) {

הפונקציה תדפיס את כל תתי-המחרוזות שהאפשריות מהמחרוזת שהתקבלה.

הפונקציה המקבלת מחרוזת, אינדקס התחלה, ומחרוזת חוזרת שתסייע הדפסת תת המחרוזות.

עזרה: מותר להשתמש בפונקציות של המחלקה String, כמו למשל substing(startIndex, endIndex) שמחזירה את תת-המחרוזת החל מאינדקס startIndex עד האינדקס endIndex (לא כולל).

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

**פתרון עם לולאות מקוננות יפסל.**

להלן main לדוגמה המזמן את הפונקציה שעליכם להשלים:

**class** SubStrings {

**private** **static** **final** String ***OPEN\_BRACKET*** = "{";

**private** **static** **final** String ***CLOSED\_BRACKET*** = "} ";

**private** **static** **final** String ***EMPTY\_STRING*** = "";

// Function to break a string into all possible combinations of

// non-overlapping substrings enclosed within a parenthesis.

**public** **static** **void** breakIntoSubStringsWithoutOverlaps(

String str, **int** i, String out) {

}

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

// input string

**final** String s = "ABCD";

**int** startIndex = 0;

*breakIntoSubStringsWithoutOverlaps*(

s, startIndex, ***EMPTY\_STRING***);

}

}

**פתרון:**

**public** **static** **void** breakIntoSubStringsWithoutOverlaps(

String str, **int** i, String out) {

**if** (i == str.length()) {

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

**return**;

}

// consider each substring S[i, j]

**for** (**int** j = str.length() - 1; j >= i; j--)

{

String subStr = ***OPEN\_BRACKET*** + str.substring(i, j + 1)

+ ***CLOSED\_BRACKET***;

// append the substring to the result and recur with

// index of the next character to be processed and

// result string so far

*breakIntoSubStringsWithoutOverlaps*(str, j+1, out+subStr);

}

}

**שאלה 3 (25 נקודות):**

"מספר מושלם" הוא מספר אשר שווה לסכום גורמיו (גורם הוא מספר שהמספר מחלק ללא שארית).

דוגמה:

הגורמים של המספר 6 הם 1,2,3 (כי 6 מחלק כל אחד מהם ללא שארית).

סכום גורמיו של המספר 6 הוא 6, ולכן הוא מספר מושלם: 1+2+3=6

הגורמים של המספר 28 הם 1,2,4,7,14 (כי 28 מחלק כל אחד מהם ללא שארית).

סכום גורמיו של המספר 28 הוא 28, ולכן הוא מספר מושלם: 1+2+4+7+14=28

כתבו את הפונקציה הבאה:

**public** **static** **boolean** isPerfect(**int** num)

הפונקציה מקבלת מספר true אם הוא מספר מושלם, אחרת תחזיר false .

מותר וכדאי לכתוב פונקציות עזר.

**פתרון:**

**private** **static** **int** countFactors(**int** num) {

**int** count = 0;

**for** (**int** i=1 ; i <= num/2 ; i++) {

**if** (num%i == 0)

count++;

}

**return** count;

}

**public** **static** **int**[] getFactors(**int** num) {

**int**[] res = **new** **int**[*countFactors*(num)];

**int** index = 0;

**for** (**int** i=1 ; i <= num/2 ; i++) {

**if** (num%i == 0) {

res[index] = i;

index++;

}

}

**return** res;

}

**public** **static** **boolean** isPerfect(**int** num) {

**int**[] factors = *getFactors*(num);

**int** sum = 0;

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

sum += factors[i];

**return** sum == num;

}

**שאלה 4 (25 נקודות):**

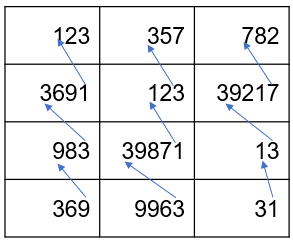
הגדרה: שני **מספרים הם אחים** אם ספרת האחדות של הראשון שווה לספרה השמאלית של השני.

דוגמה למספרי אחים: 13 ו- 398 כי ספרת האחדות של 13 היא 3, וזו הספרה השמאלית ביותר במספר השני.

הגדרה: מטריצה היא **מטריצת אחים** אם כל איבר הוא אח של האיבר שלידו ואח של האיבר שמתחתיו. לצורך ההגדרה:

* בכל שורה האיבר השמאלי יחשב לראשון והימני לשני.
* בכל עמודה האיבר התחתון יחשב לאיבר הראשון, והעליון לשני.

דוגמה למטריצת אחים:



ניתן לראות שבכל שורה ספרת האחדות של האיבר ה- j שווה לספר הימנית של האיבר במיקום ה- j+1,

וכן ניתם לראות שבכל עמודה ספרת האחדות של האיבר בשורה ה- i+1 שווה לספרה הימנית של האיבר בשורה ה- i.

כתבו את הפונקציה הבאה:

**public** **static** **boolean** isBrothersMatrix(**int**[][] mat)

הפונקציה מקבלת מטריצה של מספרים ומחזירה true האם המטריצה היא מטריצת אחים, אחרת תחזיר false.

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

**מותר וכדאי** לכתוב פונקציות עזר.

**פתרון:**

**private** **static** **boolean** areBrothers(**int** num1, **int** num2) {

**while** (num2 > 10) {

num2 /= 10;

}

**return** num1%10 == num2;

}

**public** **static** **boolean** isBrothersMatrix(**int**[][] mat) {

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

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

**if** (!*areBrothers*(mat[i][j], mat[i][j+1]))

**return** **false**;

}

}

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

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

**if** (!*areBrothers*(mat[j][i], mat[j-1][i]))

**return** **false**;

}

}

**return** **true**;

}