מדינת ישראל
משרד החינוך

סוג הבחינה: בגרות
מועד הבחינה: קיץ תשפ"ג, 2023
מספר השאלון: **899381**
תרגום לערבית (2)

دولة إسرائيل
وزارة التّربية والتّعليم

نوع الامتحان: بچروت
موعد الامتحان: صيف 2023
رقم النّموذج: **899381**
ترجمة إلى العربيّة (2)

מדעי המחשב

علم الحاسوب

הוראות

א. משך הבחינה: שלוש שעות.

ב. מבנה השאלון ומפתח ההערכה:
בשאלון זה שלושה פרקים.

פרק ראשון — (10×1) + (15×1) — 25 נק'
פרק שני — (25×2) — 50 נק'
פרק שלישי — (25×1) — 25 נק'
סך הכול — 100 נק'

ג. חומר עזר מותר בשימוש: כל חומר עזר, חוץ ממחשבון שיש בו אפשרות תכנות.

ד. הוראות מיוחדות:

1. **רשמו על הכריכה החיצונית** של המחברת את **שם המסלול שלמדתם**.
המסלול הוא אחד משלושת המסלולים האלה: אלגוריתמים, מודלים חישוביים, תכנות מונחה עצמים.

2. את כל התוכניות שיש לכתוב בשפת מחשב בפרקים הראשון והשני, כתבו בשפה אחת בלבד — Java או C#.

הערה: לא יורדו נקודות אם תכתבו בתוכניות אות גדולה במקום אות קטנה או להפך.

تعليمات

أ. مدّة الامتحان: ثلاث ساعات.

ب. مبنى النّموذج وتوزيع الدّرجات:
في هذا النّموذج ثلاثة فصول.

الفصل الأوّل – (10×1) + (15×1) – 25 درجة
الفصل الثّاني – (25×2) – 50 درجة
الفصل الثّالث – (25×1) – 25 درجة
المجموع – 100 درجة

جـ. موادّ مساعدة يُسمح استعمالها: كلّ مادّة مساعدة، عدا الحاسبة التي توجد فيها إمكانيّة برمجة.

د. تعليمات خاصّة:

1. **سجِّلوا على الغلاف الخارجيّ** للدّفتر **اسم المسار الذي تعلّمتموه**.
المسار هو أحد المسارات الثلاثة التالية: ألغوريثمات، موديلات حسابيّة، برمجة موجّهة كائنات.

2. اكتبوا جميع البرامج التي يجب كتابتها بلغة حاسوب في الفصلين الأوّل والثاني، بلغة واحدة فقط – Java أو C#.

ملاحظة: لن تُخصم درجات إذا كتبتم في البرامج حرفًا كبيرًا بدلًا من حرف صغير أو بالعكس.

يجب الكتابة في دفتر الامتحان فقط. يجب كتابة "مسوّدة" في بداية كلّ صفحة تُستعمَل مسوّدة.
كتابة أيّة مسوّدة على أوراق خارج دفتر الامتحان قد تسبّب إلغاء الامتحان.

الأسئلة في هذا النّموذج ترد بصيغة الجمع، ورغم ذلك يجب على كلّ طالبة وطالب الإجابة عنها بشكل فرديّ.

ב ה צ ל ח ה !

نتمنّى لكم النّجاح!

الأسئلة

في هذا النموذج ثلاثة فصول.

يجب الإجابة عن أسئلة من ثلاثة الفصول، حسب التعليمات في كلّ فصل.

ملاحظة: في كلّ سؤال يُطلب فيه استقبال، لا حاجة لفحص سلامة المدخَلات.

للّذين يحلّون بلغة Java : في كلّ سؤال يُطلب فيه استقبال، افترِضوا أنّ الأمر التالي مكتوب في البرنامج:

```
Scanner input = new Scanner (System.in);
```

الفصل الأوّل (25 درجة)

أجيبوا عن السؤال 1 – إلزاميّ (10 درجات).

1. "مصفوفة مرتّبة حسب الأعداد الموجبة" هي مصفوفة من نمط صحيح، تظهر فيها جميع الأعداد التي أكبر من 0 بترتيب تصاعديّ (ترتيب تصاعديّ – عدد أكبر من العدد الموجب الذي قبله أو مساوٍ له).

مثال:

المصفوفة arr التي أمامكم هي "مصفوفة مرتّبة حسب الأعداد الموجبة".

	0	1	2	3	4	5	6	7	8
arr	**5**	**9**	− 3	**17**	0	**29**	− 20	− 40	**29**

شرح: الأعداد الموجبة في المصفوفة (5 , 9 , 17 , 29 , 29) تظهر بترتيب تصاعديّ.

اكتبوا عمليّة خارجيّة باسم posOrder بلغة Java أو PosOrder بلغة #C ، تتلقّى مصفوفة من نمط صحيح – arr .
تُعيد العمليّة true إذا كانت arr "مصفوفة مرتّبة حسب الأعداد الموجبة"، خلاف ذلك تُعيد العمليّة false.
افترِضوا أنّه يوجد في المصفوفة عددان موجبان على الأقلّ.

/ يتبع في صفحة 3 /

أجيبوا عن أحد السؤالين 2-3 (15 درجة) .

2. "مصفوفة فروق" هي مصفوفة من نمط صحيح يكون فيها الفرق ثابتًا بين العددين في كلّ خليّتين متجاورتين.
مثال لمصفوفة فروق:

0	1	2	3	4	5
5	9	13	17	21	25

شرح: الفرق بين العددين في كلّ خليّتين متجاورتين في هذه المصفوفة هو 4 .

طبِّقوا العمليّة الخارجيّة التي أمامكم:

Java – public static int missingNum (int [] arr)

C# – public static int MissingNum (int [] arr)

تتلقّى العمليّة مصفوفة فروق – arr ينقصها خليّة واحدة، وعلى أثر ذلك توجد في المصفوفة خليّتان متجاورتان الفرق بين العددين اللذين فيهما يختلف عن الفرق الثابت. تُعيد العمليّة العدد الذي من المفترَض أن يكون في الخليّة الناقصة.
افترِضوا أنّ كِبَر المصفوفة arr المتلقّاة هو 4 على الأقلّ.
مثال:
بالنسبة للمصفوفة arr التي أمامكم، تُعيد العمليّة العدد 10.

	0	1	2	3	4	5
arr	6	**8**	**12**	14	16	18

شرح: الفرق بين العددين في كلّ خليّتين متجاورتين هو 2 ، باستثناء الخليّتين في المؤشّرين 1 وَ 2 ، حيث أنّ الفرق بين العددين فيهما هو 4. وذلك لأنّه تنقص بينهما خليّة فيها العدد 10 .

/ يتبع في صفحة 4 /

3. وُضعت كاميرات مرور في مسالك للمواصلات العامّة، تصوّر جميع المركبات التي تمرّ أمامها.

معطاة الفئة **CarInfo** – معلومات عن كلّ مركبة صُوِّرت، ولها ثلاث صفات:

- id – رقم لوحة ترخيص المركبة التي صُوِّرت، من نمط نصّ.
- privateCar – إذا كانت المركبة التي صُوِّرت هي مركبة خاصّة، فإنّ الصفة هي true ، وإذا كانت مركبة عامّة، فالصفة هي – false.
- speed – سرعة سفر المركبة التي صُوِّرت، من نمط صحيح.

افترِضوا أنّه توجد عمليّتا get/Get وَ set/Set لكلّ واحدة من الصفات في الفئة.

تُعطى مخالفة سير للمركبة التي تسافر في مسلك المواصلات العامّة، إذا تحقّقت على الأقلّ واحدة من الحالتين التاليتين:

- المركبة هي مركبة خاصّة.
- المركبة تسافر بسرعة أعلى من السرعة المسموحة.

أ. اكتبوا عمليّة داخليّة في الفئة **CarInfo** باسم illegal بلغة Java أو Illegal بلغة #C ، تتلقّى سرعة السفر المسموحة – maxSpeed من نمط صحيح.

تُعيد العمليّة true إذا ارتكبت المركبة مخالفة سير (أي، إذا كانت المركبة مركبة خاصّة وَ/أو سافرت بسرعة أعلى من السرعة المسموحة)، خلاف ذلك تُعيد العمليّة false.

معطاة الفئة **CameraInfo** – معلومات عن كاميرا المرور، ولها ثلاث صفات:

- city – كود المدينة التي وُضِعت فيها الكاميرا، من نمط صحيح. الكود هو عدد بين 0 وَ 99 (بما في ذلك 0 وَ 99). توجد 100 مدينة، ولكلّ مدينة هناك كود خاصّ.
- maxSpeed – السرعة المسموحة في منطقة الكاميرا، من نمط صحيح.
- cars – مصفوفة من نمط **CarInfo** للمركبات التي صُوِّرت بالكاميرا (المصفوفة بدون قِيَم null).

افترِضوا أنّه توجد عمليّتا get/Get وَ set/Set لكلّ واحدة من الصفات في الفئة.

ب. (1) اكتبوا عمليّة داخليّة في الفئة **CameraInfo** باسم allGood بلغة Java أو AllGood بلغة #C ، تُعيد true إذا كانت جميع المركبات التي صُوِّرت بالكاميرا **لم** ترتكب مخالفة سير، خلاف ذلك تُعيد العمليّة false.

يمكن استعمال العمليّة التي كتبتموها في البند "أ".

(2) اكتبوا عمليّة خارجيّة باسم legalCities بلغة Java أو LegalCities بلغة #C .

تتلقّى العمليّة مصفوفة – cameras من نمط **CameraInfo** للكاميرات التي وُضِعت في المدن الـ 100 المختلفة.

تُعيد العمليّة عدد المدن التي **لم** تشهد أيّة مخالفة سير.

ملاحظة: من الممكن أن يكون في نفس المدينة أكثر من كاميرا واحدة.

يمكن استعمال العمليّة التي كتبتموها في البند "ب (1)".

/ يتبع في صفحة 5 /

الفصل الثاني (50 درجة)

انتبهوا: في كلّ سؤال يُطلب فيه تطبيق، بإمكانكم استعمال عمليّات الفئات: دور وراصّة وشجرة بيناريّة وحلقة، بدون تطبيقها. إذا استعملتم عمليّات إضافيّة، يجب تطبيقها.

أجيبوا عن اثنين من الأسئلة 4-6 (لكلّ سؤال – 25 درجة).

4. في هذا السؤال، أُضيفت إلى الفئة **Queue** العمليّة size/Size التي أمامكم. يمكن استعمال العمليّة بدون تطبيقها.

وصف العمليّة	عنوان العمليّة
تُعيد العمليّة عدد الحدود في الدور.	**Java** – public int size() **C#** – public int Size()

طبِّقوا العمليّة الخارجيّة التي أمامكم:

Java – public static boolean twoSum (Queue<Integer> q, int x)

C# – public static bool TwoSum (Queue<int> q, int x)

تُعيد العمليّة true إذا وُجِد في الدور q الذي نتج عددان مجموعهما مساوٍ لقيمة البارامتر x . خلاف ذلك، تُعيد العمليّة false .

مثال: بالنسبة للدور q الذي أمامكم وَ x = 10 ، تُعيد العمليّة true ، لأنّه يوجد في الدور عددان (1 , 9) مجموعهما مساوٍ لِـ 10 .

		رأس الدور							
q		5	4	**1**	4	3	15	**9**	

ملاحظات:

- افترِضوا أنّه يوجد حدّان على الأقلّ في الدور q.
- لا حاجة للحفاظ على الدور q .
- **يُمنع** استعمال مصفوفة وقائمة مرتبطة في هذا السؤال.

/ يتبع في صفحة 6 /

5. معطاة الفئة **NumCount** – عدد القيَم، ولها صفتان:

- num – قيمة عدديّة، من نمط صحيح.
- count – عدد مرّات ظهور القيمة (num) ، من نمط صحيح. العدد أكبر أو مساوٍ لـ 0 .

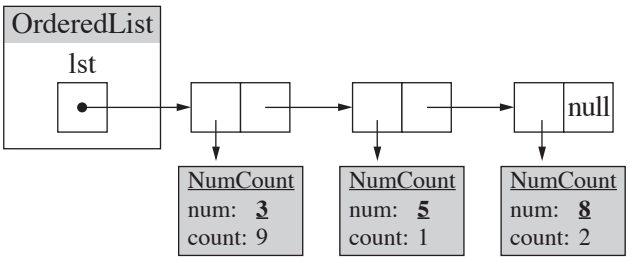
افترِضوا أنّه توجد عمليّتا get/Get وَ set/Set لكلّ واحدة من الصفتين في الفئة، وعمليّة بنائيّة تتلقّى قيَمًا بالنسبة لصفتَي الفئة.

معطاة الفئة **OrderedList** – سلسلة مرتّبة، ولها صفة واحدة:

- lst – يشير إلى رأس سلسلة حلقات من نمط **NumCount**.

سلسلة الحلقات مرتّبة بترتيب تصاعديّ لقيمة الصفة – num . قيمة الصفة num مختلفة في كلّ حلقة.

مثال: السلسلة التي أمامكم تحقّق شروط الفئة (السلسلة مرتّبة بترتيب تصاعديّ حسب قيمة الصفة num ، وقيمة الصفة num مختلفة في كلّ حلقة).



أ. **(1)** طبِّقوا في الفئة OrderedList العمليّة الداخليّة التي أمامكم:

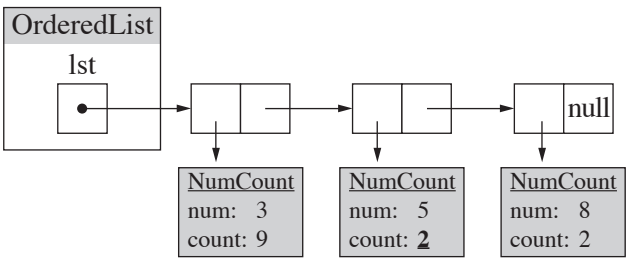
Java – public void insertNum (int x)

C# – public void InsertNum (int x)

تضيف العمليّة قيمة x إلى السلسلة بالطريقة التي أمامكم:

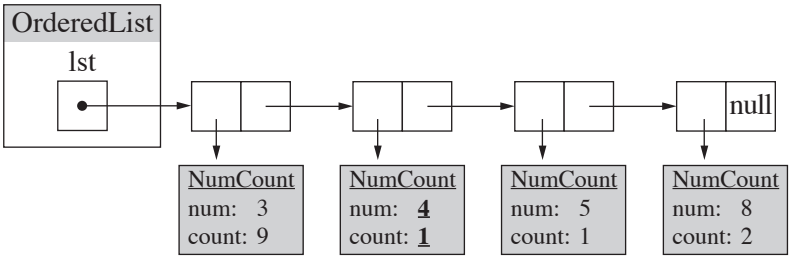
- إذا كانت في السلسلة حلقة صفتها num مساوية لِـ x ، فإنّ العمليّة تُكَبِّر بِـ 1 الصفة count (عدد مرّات الظهور) في نفس الحلقة.
- إذا كانت السلسلة فارغة أو أنّه لا توجد في السلسلة حلقة صفتها num مساوية لِـ x ، فإنّ العمليّة تُدخِل حلقة جديدة، تكون الصفة num فيها مساوية لِـ x والصفة count تكون مساوية لِـ 1 ، في المكان الذي يحافظ على الترتيب التصاعديّ للسلسلة.

مثال: بالنسبة للسلسلة المعروضة أعلاه وَ x = 5 ، بعد انتهاء العمليّة، تبدو السلسلة هكذا:



(انتبهوا: تكملة السؤال في الصفحة التالية.)

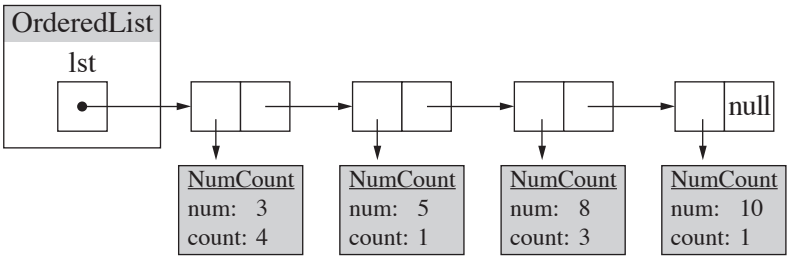
مثال إضافيّ: بالنسبة لنفس السلسلة المعروضة في الصفحة السابقة (في المثال الأوّل) وَ x = 4 ، بعد انتهاء العمليّة، تبدو السلسلة هكذا:



(2) ما هي تعقيدات زمن تشغيل العمليّة التي كتبتموها في البند "أ **(1)**"؟ علِّلوا إجابتكم.

ب. "قيمة الظهور الـn" – هي القيمة التي تظهر في المكان الـ n حسب الترتيب من بداية السلسلة (بحساب عدد مرّات الظهور – count لكلّ قيمة).

مثال: بالنسبة للسلسلة التي أمامكم وَ n = 7 ، تُعيد العمليّة القيمة 8 .



شرح: ترتيب قِيَم السلسلة بتسلسل، بحسب عدد مرّات ظهورها، هو 10 , 8 , **8** , 8 , 5 , 3 , 3 , 3 , 3 .
n = 7 وفي المكان السابع في التسلسل تظهر القيمة 8 . لهذا تُعيد العمليّة القيمة 8 .

طبِّقوا في الفئة OrderedList العمليّة الداخليّة التي أمامكم:

Java – public int valueN (int n)

C# – public int ValueN (int n)

تتلقّى العمليّة العدد n ، وتُعيد "قيمة الظهور الـn" .

افترِضوا أن "قيمة الظهور الـ n" موجودة في السلسلة.

/ يتبع في صفحة 8 /

6. "العدد الأوّليّ" هو عدد يَقسِم على نفسه وعلى 1 فقط (العددان 1 وَ 2 هما أوّليّان أيضًا).
أمامكم العمليّة الخارجيّة isPrime / IsPrime . يمكن استعمال العمليّة بدون تطبيقها.

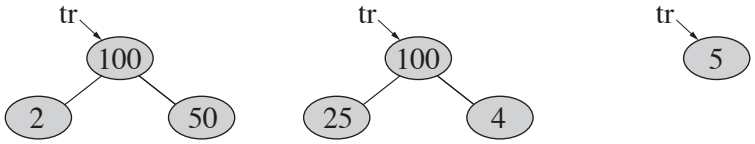
عنوان العمليّة	وصف العمليّة
Java – public static boolean isPrime (int num) **C#** – public static bool IsPrime (int num)	تُعيد العمليّة true إذا كانت القيمة num التي تمّ تلقّيها هي عدد أوّليّ، خلاف ذلك تُعيد العمليّة false .

أ. طبِّقوا العمليّة الخارجيّة التي أمامكم:

Java – public static boolean addNodes (BinNode<Integer> tr)

C# – public static bool AddNodes (BinNode<int> tr)

تتلقّى العمليّة عقدة بدون أبناء (ورقة) قيمتها أكبر من 0 . إذا كانت قيمة العقدة عددًا أوّليًّا، تُعيد العمليّة false .
خلاف ذلك، تضيف العمليّة إلى العقدة ابنَيْن تكون قيمة حاصل ضربهما مساوية لقيمة العقدة، وقيمة كلّ واحد منهما أكبر من 1 . بعد الإضافة تُعيد العمليّة true .
أمثلة: بعد انتهاء العمليّة، يمكن أن تبدو العُقَد هكذا (بالنسبة للقِيَم 5 , 100 , 100):



ب. معطاة العمليّة what/What التي أمامكم، والتي تتلقّى عقدة بدون أبناء قيمتها أكبر من 0 .

بلغة **Java**	بلغة **C#**

public static void what (BinNode<Integer> tr)
{
if (addNodes (tr))
{
what (tr.getLeft());
what (tr.getRight());
}
}
	public static void What (BinNode<int> tr)
	{
	if (AddNodes (tr))
	{
	What (tr.GetLeft());
	What (tr.GetRight());
	}
	}

(1) ارسموا الشجرة كما ستبدو بعد انتهاء العمليّة what/What بالنسبة لعقدة بدون أبناء – tr قيمتها 150 .
يجب تبيين متابعة.

(2) اشرحوا ماذا تنفّذ العمليّة what/What .

/ يتبع في صفحة 9 /

الفصل الثالث (25 درجة)

في هذا الفصل أسئلة في ثلاثة مسارات:
ألغوريثمات، في الصفحتين 9-10.
موديلات حسابيّة، في الصفحتين 11-12.
برمجة موجّهة كائنات بلغة Java ، في الصفحات 14-17؛ برمجة موجّهة كائنات بلغة C#، في الصفحات 18-21.

يجب الإجابة عن سؤال واحد في المسار الذي تعلّمتموه.

ألغوريثمات

إذا تعلّمتم هذا المسار، أجيبوا عن أحد السؤالين 7-8 (25 درجة).

7. في هذا السؤال بندان "أ – ب"، لا علاقة بينهما. أجيبوا عن البندين.

أ. أمامكم خمسة ادّعاءات. اختاروا أربعة منها.

بالنسبة لكلّ واحد من الادّعاءات التي اخترتموها، اكتبوا رقم الادّعاء في دفتركم، واذكروا إذا كان صحيحًا أم غير صحيح.
إذا ذكرتم أنّ الادّعاء صحيح – علّلوا لماذا، وإذا ذكرتم أنّ الادّعاء غير صحيح، أعطوا مثالًا مناقضًا أو علّلوا لماذا.

1. لكلّ شجرة امتداديّة DFS لنفس الرسم البيانيّ G غير الموجّه يوجد نفس عدد الأوراق دائمًا.

2. ارتفاع الشجرة الامتداديّة BFS لرسم بيانيّ G غير موجّه، دائمًا هو أصغر أو مساوٍ لارتفاع الشجرة الامتداديّة DFS لنفس الرسم البيانيّ.

3. رسم بيانيّ G غير موجّه فيه n عُقَد وَ n–2 أقواس هو غير قابل للارتباط دائمًا.

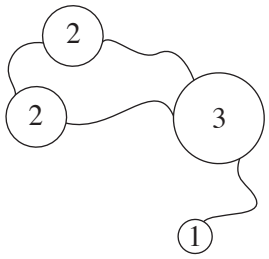
4. في كلّ رسم بيانيّ G موجّه فيه n عُقَد وَ m أقواس، وَ $m >= n$ ، توجد دائرة.

5. رسم بيانيّ G غير موجّه فيه درجات جميع الرؤوس هي أكبر من 0 يكون قابلًا للارتباط دائمًا.

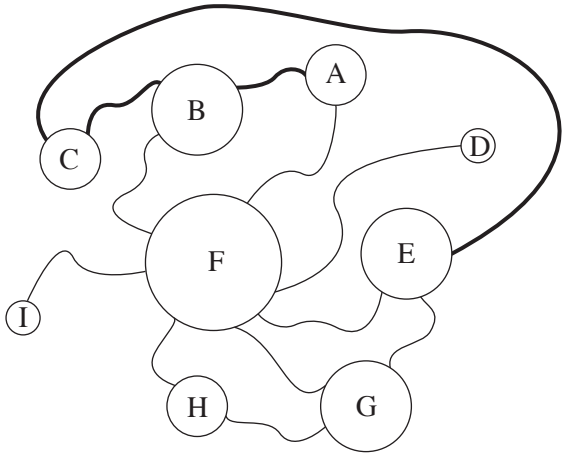
ب. "شجرة امتداديّة عظمى" هي شجرة امتداديّة لرسم بيانيّ G غير موجّه فيه مجموع قِيَم الأقواس هو أكبر ما يمكن.
اكتبوا ألغوريثمًا يجد "شجرة امتداديّة عظمى" في الرسم البيانيّ G .

/ يتبع في صفحة 10 /

8. في "شبكة شوارع" الانتقال من شارع إلى شارع هو دائمًا من خلال دوّار سير.
كِبَر نصف قطر الدوّار (بالأمتار) هو كعدد الشوارع المرتبطة بالدوّار. مثلًا: نصف قطر الدوّار الذي يربط 3 شوارع، هو 3 أمتار، ونصف قطر الدوّار الذي يربط 6 شوارع هو 6 أمتار.
مثال: أمامكم رسم لشبكة شوارع. داخل كلّ دوّار مذكور نصف قطر نفس الدوّار.



أمامكم شبكة الشوارع "NET" في مدينة معيّنة.



يرغب عابر سبيل في السير من دوّار معيّن إلى دوّار آخر عبر المسار الأقصر. المسار الأقصر هو المسار الذي فيه مجموع أنصاف أقطار جميع الدوّارات التي يمرّ فيها هو الأصغر. مجموع أنصاف الأقطار لا يشمل نصف قطر الدوّار الذي يبدأ منه مساره، لكنّه يشمل نصف قطر الدوّار الذي يُنهي فيه مساره.
مثال: في الرسم أعلاه لشبكة الشوارع "NET"، المسار الأقصر من الدوّار A إلى الدوّار E مرسوم بخطّ غامق.
كِبَر أنصاف الأقطار في هذا المسار هو 3+2+3 ومجموعها هو 8 . هذا المجموع هو الأصغر من بين جميع الإمكانيّات.

أ. في شبكة الشوارع "NET" ، ما هو المسار الأقصر من الدوّار H إلى الدوّار C ؟ اكتبوا أسماء الدوّارات في هذا المسار، حسب الترتيب، من اليسار إلى اليمين (لا حاجة لتنفيذ متابعة).

ب. (1) اكتبوا ألغوريثمًا يجد المسار الأقصر من الدوّار K_1 إلى الدوّار K_2 بالنسبة لشبكة شوارع ما.
ملاحظة: يجب كتابة ألغوريثم ناجع لا يمرّ على جميع المسارات الممكنة.

(2) ارسموا الرسم البيانيّ الذي يمثّل شبكة الشوارع "NET" المعطاة أعلاه، بطريقة تلائم الألغوريثم الذي كتبتموه.

/ يتبع في صفحة 11 /

موديلات حسابيّة

إذا تعلّمتم هذا المسار، أجيبوا عن أحد السؤالين 9–10 (25 درجة) .

9. معطاة اللغات $L_1 - L_4$ فوق الأبجديّة $\{a,b\}$:

L_1 = لغة جميع الكلمات التي فيها عدد مرّات ظهور الحرف a مساوٍ لعدد مرّات ظهور الحرف b .

L_2 = لغة جميع الكلمات التي فيها عدد مرّات ظهور الحرف a أكبر من عدد مرّات ظهور الحرف b .

L_3 = لغة جميع الكلمات التي فيها أكثر من ثلاثة حروف.

L_4 = لغة جميع الكلمات التي تبدأ بالحرف a وتنتهي بالحرف b أو تبدأ بالحرف b وتنتهي بالحرف a .

أجيبوا عن جميع البنود "أ – ز" التي أمامكم:

أ. ابنوا أوتوماتًا نهائيًّا محدودًا يتلقّى اللغة L_4 .

ب. برهِنوا أنّ اللغة $L_3 \cap L_4$ هي نظاميّة.

جـ. اكتبوا اللغة الناتجة من العمليّة $L_1 \cup L_2$. هل اللغة الناتجة هي نظاميّة؟ علِّلوا إجابتكم.

د. اكتبوا اللغة الناتجة من العمليّات $L_1 \cap L_2 \cup \overline{L_2}$.

هـ. هل اللغة $L_1 \cap L_2$ هي نظاميّة؟ علِّلوا إجابتكم.

و. ابنوا أوتوماتًا نهائيًّا محدودًا ليس كاملًا، يتلقّى اللغة $L_2 \cap \overline{L_3}$.

ز. اكتبوا اللغة الناتجة من العمليّة $L_4 \cap R(L_4)$.

/ يتبع في صفحة 12 /

10. العمليّة **Generate** تُنفَّذ على عددين طولهما متساوٍ، وهما مركَّبان من الرقمين 0 وَ 1 فقط. تُنتِج العمليّة كلمة جديدة بنفس الطول، بالطريقة التالية:

إذا كان الرقم في المكان n هو 0 في العددين، فإنّ الحرف في المكان الـ n هو F، خلاف ذلك، الحرف في المكان الـ n هو T.

مثال: العمليّة **Generate** على العددين $x = 11001$ وَ $y = 10010$، تُنتِج الكلمة TTFTT كما هو موصوف أمامكم:

	0	1	2	3	4
x	1	1	0	0	1
y	1	0	0	1	0
	Generate				
الكلمة الناتجة	T	T	F	T	T

اكتبوا آلة تيورينج تتلقّى في بداية الشريط مُدخَلًا لعددين طولهما متساوٍ، مركَّبَين من الرقمين 0 وَ 1 فقط.
تُعيد الآلة الكلمة الناتجة من العمليّة **Generate** عليهما.

توجيهات:

- العددان في المُدخَل يفصل بينهما # .
- لا حاجة للحفاظ على المُدخَلات (يمكن تغيير المُدخَلات إلى رموز مختلفة).
- الكلمة الجديدة تظهر في مكان ما في الشريط بين إشارتَي $.
- لا حاجة لفحص سلامة المُدخَلات.

مثال:

الشريط قبل التشغيل:

	0	1	2	3	4	5		0	1	2	3	4	5			
⊢	1	0	1	0	1	0	#	1	0	1	1	0	0	∆	∆	...

الشريط بعد التشغيل:

...	$	T	F	T	T	T	F	$	...

/ يتبع في صفحة 13 /

انتبهوا: تكملة الامتحان في الصفحة التالية.

/ يتبع في صفحة 14 /

برمجة موجّهة كائنات بلغة Java

إذا تعلّمتم هذا المسار وتكتبون بلغة Java، أجيبوا عن أحد السؤالين 11-12 (25 درجة).

11. في مطعم "الأخوة" يمكن حجز مكان لزبون واحد أو أكثر. بعد انتهاء الوجبة، يدفع كلّ واحد من الزبائن بمفرده مقابل الطعام الذي أكله. يستطيع كلّ زبون أن يختار الدفع نقدًا أو بالتطبيق أو ببطاقة اعتماد. لِمَن يدفع ببطاقة اعتماد هناك إمكانيّة لدفع الحساب بعدّة أقساط متساوية.

بهدف إدارة منظومة الدفعات، يطوِّرون للمطعم مشروعًا فيه واجهة التطبيق (Interface) **IPayment**، والفئات **Restaurant, Reservation, Credit, App, Cash** ، كما هو مفصَّل فيما يلي:

- ❖ **IPayment** – الدفع

  توجد في واجهة التطبيق العمليّة double getPrice()
- ❖ **Cash** – الدفع نقدًا

  صفتا الفئة:
  - sumCash – المبلغ للدفع نقدًا، من نمط حقيقيّ.
  - name – اسم الزبون، من نمط نصّ.
- ❖ **App** – الدفع بالتطبيق

  صفتا الفئة:
  - sumApp – المبلغ للدفع بالتطبيق، من نمط حقيقيّ.
  - phoneNumber – رقم هاتف، من نمط نصّ.
- ❖ **Credit** – الدفع ببطاقة اعتماد

  صفات الفئة:
  - num – عدد الأقساط، من نمط صحيح.
  - part – المبلغ في كلّ قسط، من نمط حقيقيّ.
  - creditNumber – رقم بطاقة الاعتماد، من نمط نصّ.
- ❖ **Reservation** – حجز

  صفات الفئة:
  - date – تاريخ الحجز، من نمط نصّ.
  - total – الحساب الكلّيّ لجميع الزبائن معًا في نفس الحجز، من نمط حقيقيّ.
  - payments – مصفوفة تحفظ المبلغ الذي دفعه كلّ واحد من الزبائن في نفس الحجز، من نمط **IPayment** (المصفوفة بدون قِيَم null).
- ❖ **Restaurant** – المطعم

  صفتا الفئة:
  - array – مصفوفة بِكبَر 10,000، من نمط **Reservation** .
  - current – كمّيّة الحجوزات المحفوظة، من نمط صحيح.

  افترِضوا أنّ الحجوزات تُحفَظ بتسلسل في مصفوفة وليس بينها قِيَم null .

ملاحظة: افترِضوا أنّه في كلّ فئة تمّ تعريف عمليّتَي get وَ set وعمليّات بنائيّة.

(انتبهوا: تكملة السؤال في الصفحة التالية.)

أ. **(1)** ارسموا مخطّطًا هرميًّا يصف العلاقات بين فئات وواجهة تطبيق المشروع.
يجب الإشارة إلى تطبيق واجهة التطبيق بواسطة السهم ----◁ وإلى احتواء بواسطة الإشارة ◆—— .

(2) يقدّم المطعم تخفيضات للزبائن حسب طريقة الدفع: الزبون الذي يدفع نقدًا ومبلغ الدفع هو أكثر من 200 شيكل يستحقّ تخفيضًا بقيمة 10% ، والزبون الذي يدفع بالتطبيق يستحقّ دائمًا تخفيضًا بقيمة 5% ، والزبون الذي يدفع ببطاقة اعتماد يدفع سعرًا كاملًا.
عمليّة واجهة التطبيق getPrice تُعيد مبلغ الدفع الذي يدفعه كلّ زبون بعد حساب التخفيض (عندما لا يوجد تخفيض، تُعيد العمليّة سعرًا كاملًا). طبِّقوا عمليّة واجهة التطبيق getPrice في كلّ واحدة من الفئات اللازمة.

ب. اكتبوا في الفئة **Reservation** عمليّة باسم cashTotal . تُعيد العمليّة مبلغ المال النقديّ الذي تمّ تلقّيه في الحجز (بعد حساب التخفيض).

جـ. أمامكم عمليّة في الفئة **Reservation** :

```
public void printDetails () {
    for (int i = 0; i < payments.length; i ++) {
        System.out.println (payments[i].getDetails());
    }
}
```

تطبع العمليّة بالنسبة لكلّ زبون تفاصيل معيّنة حسب المعايير التالية:
- إذا دفع الزبون نقدًا، تطبع العمليّة اسمه.
- إذا دفع الزبون بالتطبيق، تطبع العمليّة رقم هاتفه.
- إذا دفع الزبون ببطاقة اعتماد، تطبع العمليّة رقم بطاقة اعتماده.

أضيفوا العمليّات الضروريّة إلى المشروع كي تنفّذ العمليّة المطلوب. اذكروا بالنسبة لكلّ عمليّة أضفتموها، إلى أيّة فئة أو إلى أيّة واجهة تطبيق تتبع.
ملاحظة: لا تغيّروا العمليّة printDetails .

/ يتبع في صفحة 16 /

12. أمامكم الفئتان First وَ Second :

```
public class First
{
    private static int count = 0;
    protected int x;
    protected int y;
    public First (int num) {
        this.x = num;
        this.y = num;
        count ++;
        System.out.println ("First 1");
    }
    public First (int num1, int num2) {
        this.x = num1;
        this.y = num2;
        count ++;
        System.out.println ("First 2");
    }
    public static int getCount() {
        return count;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
    public int sum() {
        return this.x + this.y;
    }
    public void add(First other) {
        this.x += other.x;
        this.y += other.y;
        System.out.println("x = "+ this.x +
                           "y = "+ this.y);
    }
}
```

```
public class Second extends First
{
    private int z;
    public Second (int num) {
        super (num);
        this.z = num;
        System.out.println ("Second");
    }
    public int sum() {
        return super.sum() + this.z;
    }
    public void add (First other) {
        this.x += other.getX();
        this.y += other.getY();
        if (other instanceof Second)
            this.z += ((Second)other).z;
        System.out.println("x = "+ this.x +
                           " y = "+ this.y+ " z = " +this.z);
    }
}
```

(انتبهوا: تكملة السؤال في الصفحة التالية.)

معطاة الفئة Tester :

```
public class Tester
{
    public static void main(String[] args)
    {
        First f1 = new First (40);
        First f2 = new First (40, 50);
        First f3 = new Second (100);
        Second s1 = new Second (100);
        Second s2 = new Second (100);
        // ***
    }
}
```

أ. اذكروا الكائنات التي نتجت في العمليّة main ، واكتبوا مُخرَج العمليّة.

ب. عوِّضوا كلّ واحد من الأوامر 1–10 التالية في العمليّة main في المكان المُشار إليه أعلاه بِـ ***.
اكتبوا في الدفتر رقم الأمر، واذكروا إذا كان الكود سليمًا أم غير سليم.
إذا كان الكود سليمًا – اكتبوا المُخرَج، وإذا كان غير سليم، اشرحوا لماذا.
ملاحظة: لا توجد علاقة بين الأوامر. أي، يجب التطرّق إلى كلّ أمر كأنّه هو الوحيد في العمليّة main.

1. System.out.println ("Total = " + First.getCount());
2. System.out.println ("Total = " + Second.getCount());
3. System.out.println ("sum = " + s1.sum());
4. System.out.println ("sum = " + f3.sum());
5. s1=new First (100);
6. f1.add (s2);
7. s1.add (s2);
8. s2.add (f3);
9. ((First)s1).add (f1);
10. s1=new Second (100, 100);

/ يتبع في صفحة 18 /

برمجة موجّهة كائنات بلغة #C

إذا تعلّمتم هذا المسار وتكتبون بلغة #C، أجيبوا عن أحد السؤالين 13-14 (25 درجة) .

13. في مطعم "الأخوة" يمكن حجز مكان لزبون واحد أو أكثر. بعد انتهاء الوجبة، يدفع كلّ واحد من الزبائن بمفرده مقابل الطعام الذي أكله. يستطيع كلّ زبون أن يختار الدفع نقدًا أو بالتطبيق أو ببطاقة اعتماد. لِمَن يدفع ببطاقة اعتماد هناك إمكانيّة لدفع الحساب بعدّة أقساط متساوية.

بهدف إدارة منظومة الدفعات، يطوِّرون للمطعم مشروعًا فيه واجهة التطبيق (Interface) **IPayment**، والفئات **Restaurant, Reservation, Credit, App, Cash** ، كما هو مفصَّل فيما يلي :

- **IPayment** – الدفع

  توجد في واجهة التطبيق العمليّة double GetPrice()
- **Cash** – الدفع نقدًا

  صفتا الفئة :
  - sumCash – المبلغ للدفع نقدًا، من نمط حقيقيّ.
  - name – اسم الزبون، من نمط نصّ.
- **App** – الدفع بالتطبيق

  صفتا الفئة :
  - sumApp – المبلغ للدفع بالتطبيق، من نمط حقيقيّ.
  - phoneNumber – رقم هاتف، من نمط نصّ.
- **Credit** – الدفع ببطاقة اعتماد

  صفات الفئة :
  - num – عدد الأقساط، من نمط صحيح.
  - part – المبلغ في كلّ قسط، من نمط حقيقيّ.
  - creditNumber – رقم بطاقة الاعتماد، من نمط نصّ.
- **Reservation** – حجز

  صفات الفئة :
  - date – تاريخ الحجز، من نمط نصّ.
  - total – الحساب الكلّيّ لجميع الزبائن معًا في نفس الحجز، من نمط حقيقيّ.
  - payments – مصفوفة تحفظ المبلغ الذي دفعه كلّ واحد من الزبائن في نفس الحجز، من نمط **IPayment** (المصفوفة بدون قِيَم null) .
- **Restaurant** – المطعم

  صفتا الفئة :
  - array – مصفوفة بِكبَر 10,000، من نمط **Reservation** .
  - current – كمّيّة الحجوزات المحفوظة، من نمط صحيح.

  افترِضوا أنّ الحجوزات تُحفَظ بتسلسل في مصفوفة وليس بينها قِيَم null .

ملاحظة: افترِضوا أنّه في كلّ فئة تمّ تعريف عمليّتَي Get وَ Set وعمليّات بنائيّة.

(انتبهوا : تكملة السؤال في الصفحة التالية .)

/ يتبع في صفحة 19 /

أ. **(1)** ارسموا مخطّطًا هرميًّا يصف العلاقات بين فئات وواجهة تطبيق المشروع.
يجب الإشارة إلى تطبيق واجهة التطبيق بواسطة السهم ----◁ وإلى احتواء بواسطة الإشارة ◆—— .

(2) يقدّم المطعم تخفيضات للزبائن حسب طريقة الدفع: الزبون الذي يدفع نقدًا ومبلغ الدفع هو أكثر من 200 شيكل يستحقّ تخفيضًا بقيمة 10% ، والزبون الذي يدفع بالتطبيق يستحقّ دائمًا تخفيضًا بقيمة 5% ، والزبون الذي يدفع ببطاقة اعتماد يدفع سعرًا كاملًا.
عمليّة واجهة التطبيق GetPrice تُعيد مبلغ الدفع الذي يدفعه كلّ زبون بعد حساب التخفيض (عندما لا يوجد تخفيض، تُعيد العمليّة سعرًا كاملًا). طبِّقوا عمليّة واجهة التطبيق GetPrice في كلّ واحدة من الفئات اللازمة.

ب. اكتبوا في الفئة **Reservation** عمليّة باسم CashTotal . تُعيد العمليّة مبلغ المال النقديّ الذي تمّ تلقّيه في الحجز (بعد حساب التخفيض).

جـ. أمامكم عمليّة في الفئة **Reservation** :

```
public void PrintDetails () {
    for (int i = 0; i< payments.Length; i ++) {
        Console.WriteLine (payments[i].GetDetails());
    }
}
```

تطبع العمليّة بالنسبة لكلّ زبون تفاصيل معيّنة حسب المعايير التالية:
- إذا دفع الزبون نقدًا، تطبع العمليّة اسمه.
- إذا دفع الزبون بالتطبيق، تطبع العمليّة رقم هاتفه.
- إذا دفع الزبون ببطاقة اعتماد، تطبع العمليّة رقم بطاقة اعتماده.

أضيفوا العمليّات الضروريّة إلى المشروع كي تنفّذ العمليّة المطلوب. اذكروا بالنسبة لكلّ عمليّة أضفتموها، إلى أيّة فئة أو إلى أيّة واجهة تطبيق تتبع.
ملاحظة: لا تغيّروا العمليّة PrintDetails .

/ يتبع في صفحة 20 /

14. أمامكم الفئتان First وَ Second :

```
public class First
{
    private static int count = 0;
    protected int x;
    protected int y;
    public First (int num) {
        this.x = num;
        this.y = num;
        count ++;
        Console.WriteLine ("First 1");
    }
    public First (int num1, int num2) {
        this.x = num1;
        this.y = num2;
        count ++;
        Console.WriteLine ("First 2");
    }
    public static int GetCount() {
        return count;
    }
    public int GetX() {
        return x;
    }
    public int GetY() {
        return y;
    }
    public  virtual int Sum() {
        return this.x + this.y;
    }
    public virtual void Add (First other) {
        this.x += other.x;
        this.y += other.y;
        Console.WriteLine ("x = "+this.x +
                           " y = " +this.y);
    }
}
```

```
public class Second : First
{
    private int z;
    public Second (int num) : base (num) {
        this.z = num;
        Console.WriteLine (“Second”);
    }
    public override int Sum() {
        return base.Sum() + this.z;
    }
    public override void Add (First other) {
        this.x += other.GetX();
        this.y += other.GetY();
        if (other is Second)
            this.z += ((Second)other).z;
        Console.WriteLine("x = "+ this.x +
                    " y =" + this.y + " z =" + this.z);
    }
}
```

(انتبهوا: تكملة السؤال في الصفحة التالية.)

/ يتبع في صفحة 21 /

معطاة الفئة Tester :

```
public class Tester
{
    public static void Main(string[] args)
    {
        First f1 = new First (40);
        First f2 = new First (40, 50);
        First f3 = new Second (100);
        Second s1 = new Second (100);
        Second s2 = new Second (100);
        // ***
    }
}
```

أ. اذكروا الكائنات التي نتجت في العمليّة Main ، واكتبوا مُخرَج العمليّة.

ب. عوِّضوا كلّ واحد من الأوامر 1–10 التالية في العمليّة Main في المكان المُشار إليه أعلاه بِـ ***.
اكتبوا في الدفتر رقم الأمر، واذكروا إذا كان الكود سليمًا أم غير سليم.
إذا كان الكود سليمًا – اكتبوا المُخرَج، وإذا كان غير سليم، اشرحوا لماذا.
ملاحظة: لا توجد علاقة بين الأوامر. أي، يجب التطرّق إلى كلّ أمر كأنّه هو الوحيد في العمليّة Main.

1. Console.WriteLine ("Total = " + First.GetCount());
2. Console.WriteLine ("Total = " + Second.GetCount());
3. Console.WriteLine ("sum = " + s1.Sum());
4. Console.WriteLine ("sum = " + f3.Sum());
5. s1 = new First (100);
6. f1.Add (s2);
7. s1.Add (s2);
8. s2.Add (f3);
9. ((First)s1).Add (f1);
10. s1 = new Second (100, 100);

בהצלחה!
نتمنّى لكم النّجاح!