

ממשקים

אז מה זה בעצם ממשק?

למעשה זו מחלקה . מחלקה שכל מה שיש בה זה רק הצהרות על פונקציות וזאת מבלי שיהיה בהן מימוש של הפונקציות. אז למה צריך את הממשק?

ממשק בא לתאר יכולת מסוימת הכוונה היא בכך שהממשק מגדיר קשר של can do, בעוד ש abstract-class מגדיר קשר של is a.

A person **can** walk

A person **can** talk

A soldier **is a** person




A soldier **can** shoot

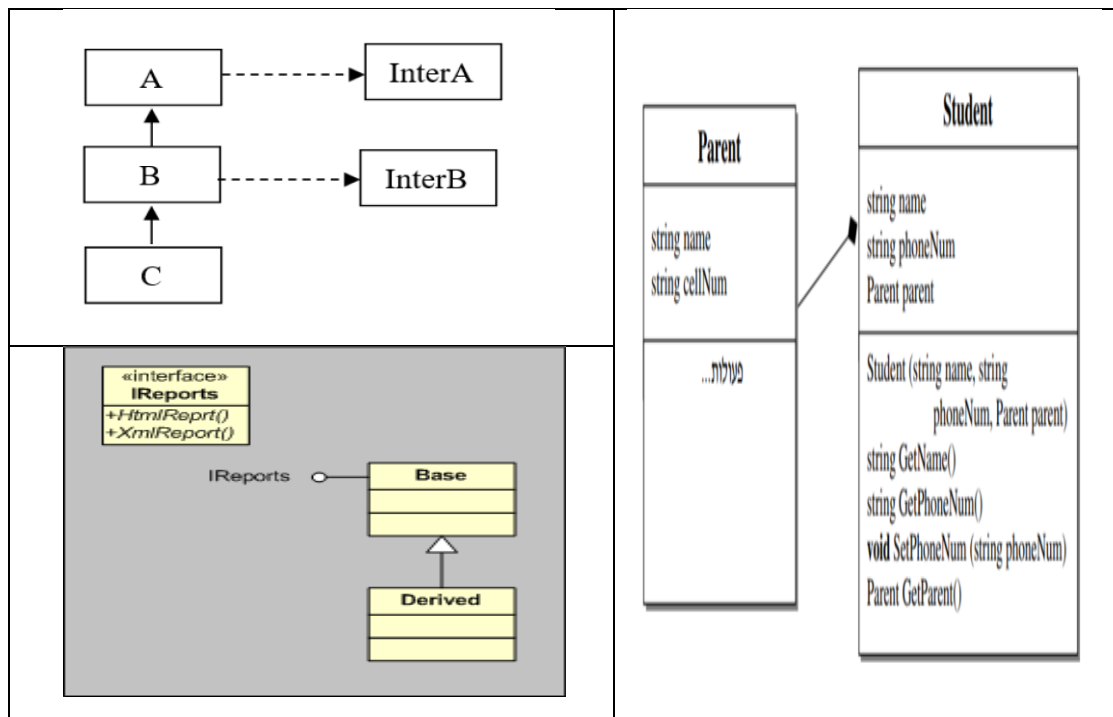
A cat **can** walk

ובעברית

Can do	Is a סוג של	
משמיע קול	חיה	כלב
מצליח בלימודים	בן אדם	תלמיד
מאיצה מהירות	כלי תחבורה	מכונית טויוטה
משמיע שיר/ להריץ קדימה	מוצר חשמלי	נגן DVD
וכו	וכו	וכו

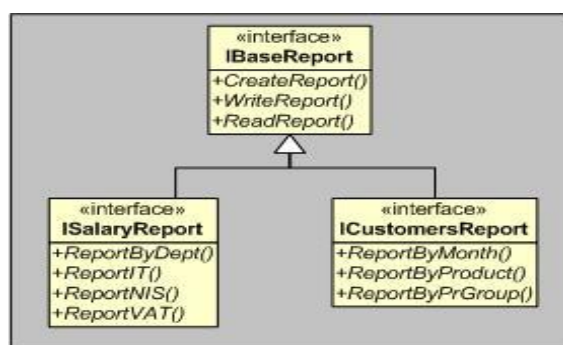
כיצד נסמן מחלקות שיורשות מחלקות מכילות ומחלקות שממשות ממשק

מימוש ממשק	הורשה	הכלה
		
<p>מחלקה A מממשת את הממשק InterA. מחלקה B מממשת את InterA. וגם את InterB. C חייבת לממש את כל הממשקים</p>	<p>מחלקה C יורשת ממחלקה B שיורשת ממחלקה A</p>	<p>דוגמה המחלקה S מכלילה הפניה למחלקה Parent</p>

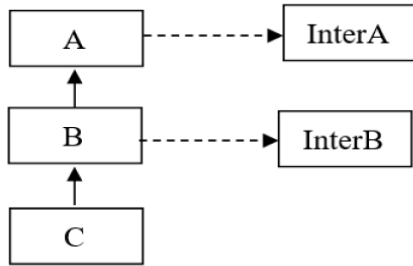


מספר כללים בנוגע לממשקים

- ממשק (Interface) הנו מבנה לוגי המצהיר על מתודות מופשטות (Abstract Methods), מאפיינים מופשטים או אירועים (Events).
- מחלקה יורשת קודם מחלקה ואחר כך מממשת ממשק.
- מחלקה יכולה לרשת מחלקה אחת בלבד, אולם בנוסף היא יכולה לרשת מספר רב של ממשקים.
- ממשק אינו יכול להכיל מתודות עם מימושים אלה רק חתימה של המתודות.
- ממשק אינו יכול להכיל איברי מחלקה (Data Members).
- ממשק דומה למחלקה אבסטרקטית (Abstract class) בזה שלא ניתן לייצר ממנו מופעים.
- בממשקים לא ניתן להגדיר הרשאות גישה הם תמיד public,
- המימושים במחלקות הנגזרות לא חייבים להיות וירטואליים.
- מחלקה B היורשת ממחלקה A וידוע שמחלקה A מממשת ממשק, חובה גם על מחלקה B לממש את הממשקים שמחלקה A מימשה. מחלקה המממשת ממשק נגזר, חייבת לממש את כל מה שמוגדר הן בממשק הבסיס והן בממשק הנגזר



- ממשק יכול לרשת ממשק אחר.



נבחן לרגע את התרשים הבא

A היא מחלקת בסיס. המחלקה B יורשת מ A והמחלקה C יורשת B.
 A מממשת את הממשק InterA.
 B מממשת את הממשק InterB.
 B חייבת לממש גם את הממשק InterA.
 C חייבת לממש את כל הממשקים.

לעיתים קורה שמחלקה יורשת שני ממשקים שונים, אולם שניהם מכילים הגדרה זהה של מתודה. במקרה שכזה נהיה חייבים לממש את המתודות של הממשקים באופן מפורש (Explicit). המגבלות הקיימות במקרה זה הנם:

- לא ניתן להגדיר הרשאת גישה באופן משתמע, הרשאת הגישה היא תמיד public ובאופן מרומז.
- מימוש מפורש אינו יכול להיות וירטואלי.

שימו לב לדוגמה הבאה

```

public interface IPrint1 { void Print(); }
public interface IPrint2 { void Print(); }
class Sample:IPrint1,IPrint2
{
    public Sample() { ... }
    public Sample(int num) { ... }
    void IPrint1.Print() { ... }
    void IPrint2.Print() { ... }
}
  
```

בשני הממשקים IPrint1 IPrint2 – קיימת פעולה המכילה את אותו שם של פעולה Print.
 קריאה לאחת מהפעולות יחייב אותנו לבצע casting

```

static void Main(string[] args)
{
    Sample s = new Sample(33);
    (IPrint1)s.Print();
    (IPrint2)s.Print();
}
  
```

```

IPrint1 i1 = new Sample(12);
i1.Print();
IPrint2 i2 = new Sample(45);
i1.Print();
}

```

תרגיל 1

עבור גן החיות שלנו נגדיר את המשימות הבאות

- חיה צריכה לאכול – כל חיה מקבלת סוג שונה של אוכל במספר פעמים ביום.
- חיה עוברת טיפולים רפואיים. הטיפולים הינם חיסונים שוטפים/ טיפולי שיניים/ טיפולי חירום
- כלבים ופילים עוברים שטיפה פעם בחודש.
- לסוסים צריך לקחת דם פעם בחודש

תרגיל 2 .

קרדיט לצוות תיכון רוטברג סוזי שמוליק ועינת

במחסן מלאי תרופות מעוניינים למחשב את מאגר התרופות במחסן. לביצוע המשימה הוגדרו שתי מחלקות בעבור שני סוגים של תרופות: המחלקה Narcotic בעבור תרופות הדורשות מרשם מיוחד ומחלקה OverTheCounter עבור תרופות רגילות. חולה הנזקק לתרופה ממחלקה זו מקבל 20% הנחה ממחירה אם מספר הכדורים בחבילה שקנה קטן מ 30 .

במחלקה Narcotic מוגדרות התכונות הבאות: שם התרופה, מחיר התרופה, התמחות הרופא הממליץ ומספר כדורים בחבילה והפעולות הבאות: בנאי, פעולות מאחזרות get, פעולות מעדכנות set, פעולות toString .

במחלקה OverTheCounter מוגדרות התכונות הבאות: שם התרופה, מחיר התרופה, מספר כדורים בחבילה, מספר חבילות המותר למכירה אחת. חולה הנזקק לתרופה ממחלקה זו מקבל 10% הנחה ממחירה אם מספר הכדורים המותר למכירה אחת עולה על 5 .

שתי המחלקות מרחיבות את המחלקה Medicine

בנוסף הוגדר מימשק Pay המגדיר את הפעולות הבאות:

public void upPrice(int percent)	פעולה המעלה מחיר בסיס של התרופה באחוזים.
public double myPrice()	פעולה המחשבת ומחזירה את המחיר (בהתאם לאחוזי ההנחה המגיעים לחולה) .

המחלקות Narcotic ו OverTheCounter ממשות את הממשק Pay .

- שרטטו תרשים היררכי של הקשר בין ארבעת הגורמים המתוארים בתרגיל זה.
- כתוב את כותרות, תכונות ופעולות הנדרשות למחלקה Narcotic .
- כתוב את המימשק Pay – הסבר היכן תמומשנה פעולות המימשק והצע מימוש מתאים לפעולה חישוב הנחה לחולה.

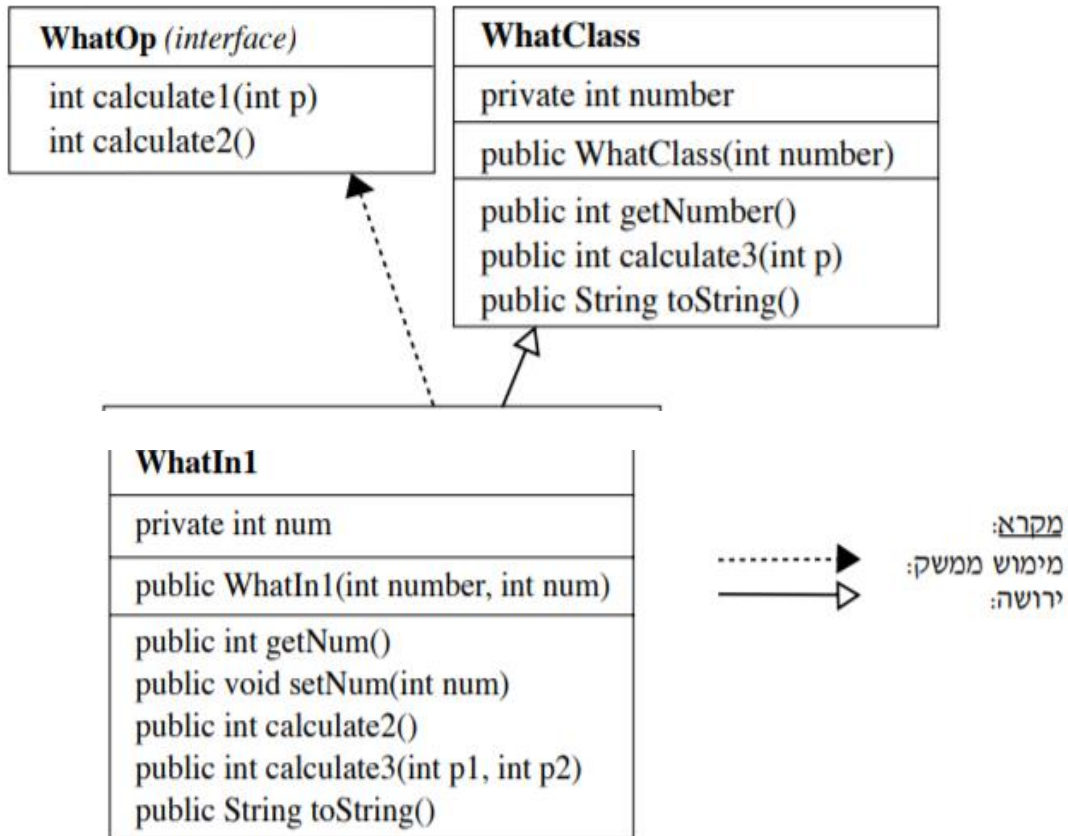
ד. נתונה התכנית ראשית הבאה :

```
public static void main(String[] args)
{
    Medicine [] med=new Medicine[4];
    med[0]=new Narcotic("retalin",180," neurolog", 15);
    med[1]=new OverTheCounter("optalgin",20,90,7);
    med[3]=new Narcotic("prozak",220," psychiatrist",30);
    med[4]=new OverTheCounter("edvil",40,60,4);
    for(int i=0; i<med.length; i++)
    {
        Console. ReadLine (med[i].toString()+" : "+med[i].myPrice()); }
    }
```

מה יהיה פלט התכנית?

ה. נודע כי יש להעלות את מחירי התרופות ב 20% הוסיפו לתכנית הראשית שורות קוד שיעדכנו את כל התרופות להיות במחיר החדש.

וטרינר גן החיות החליט שהוא רוצה להגדיר לכל חיה – טיפול רפואי תקופתי. כלומר כל חיה תקבל טיפול בכל פרק זמן המתאים לה. הטיפול יכלול את המידע הבא: במידה וצריך-מתן תרופה, חיסון שנתי, מתן מזון רפואי. הציעו ייצוג מתאים בעזרת ממשקים והגדירו איזו מחלקה תממש את הייצוג שהצעתם



- א. האם מימוש הפעולה `public int calculate1(int p)` במחלקה **WhatClass**, יענה על דרישת המימוש שלה במחלקה **WhatIn1**? נמק את תשובתך.
- ב. מימוש הפעולה הבונה במחלקה **WhatIn1** הוא:

```

public WhatIn1(int number, int num)
{
    this.number = number;
    this.num = num;
}
    
```

האם הפעולה תקינה? אם כן – תאר את ביצוע הפעולה; אם לא – תקן את הפעולה

ג. במחלקה **WhatIn1** מומשה הפעולה `calculate2()`, המחזירה את הערך השלם של ממוצע תכונות העצם:

```
public int calculate2()
{ return (int)((this.number + this.num)/2); }
```

האם הפעולה תקינה? אם כן – תאר את ביצוע הפעולה; אם לא – תקן את הפעולה (אין לשנות דבר פרט לפעולה עצמה). רשום במחברתך את הפעולה המתוקנת.

ד. לפניך תיאור חלקי של המחלקה **WhatIn2**:

<code>class WhatIn2 extends WhatIn1</code>
<code>private int sum</code>
<code>public WhatIn2(int number, int num, int sum)</code>
<code>public int calculate3(int p1, int p2, int p3)</code>

(1) האם אפשר להסתמך על הפעולה הבונה בררת מחדל במקום להגדיר פעולה בונה במחלקה **WhatIn2**? נמק את תשובתך.

(2) הפעולה `calculate3` מומשה בשלוש המחלקות **WhatIn1**, **WhatClass** ו-**WhatIn2** באופן הזה:

<code>class WhatClass</code>	<code>public int calculate3(int p)</code> <code>{ return this.number * p; }</code>
<code>class WhatIn1</code>	<code>public int calculate3(int p1, int p2)</code> <code>{ return this.calculate3(p1) + this.num * p2 * p2; }</code>
<code>class WhatIn2</code>	<code>public int calculate3(int p1, int p2, int p3)</code> <code>{ return this.calculate3(p1, p2) + this.sum * p3 * p3 * p3; }</code>

הנח כי העצם `obj` הוא מטיפוס **WhatIn2**, וערכי התכונות שלו הם:

`number = 1`

`num = 2`

`sum = 3`

לפניך הוראה הכתובה בפעולה הראשית:

```
System.out.println(obj.calculate3(1000, 100, 10));
```

הראה את המעקב אחר ביצוע ההוראה. במעקב התייחס לזימון פעולות ולערכי תכונות העצם. רשום את הפלט המתקבל.

לפניך כמה עקרונות של תכנות מונחה עצמים:

- הכמסה – encapsulation
- העמסה – overloading
- ירושה – inheritance
- דריסה – overriding
- רב-צורתיות – polymorphism

בשאלה זו תהיה התייחסות לחלק מהם.

לפניך מימוש חלקי של שתי מחלקות: **Stam**, **Davar**.

```
public class Stam
{
    private char x;

    public Stam() { this.x = '*'; }
    public Stam(char c) { this.x = c; }
    public Stam GetStam() { return this; }
    public override string ToString() { return "x=" + this.x; }
    public bool IsSame1(Stam other)
    { // התייחסות למימוש בהמשך השאלה }
    public bool IsSame2(Stam other)
    { // התייחסות למימוש בהמשך השאלה }
    public void Same(Stam other)
    {
        if (this.IsSame1(other))
            Console.WriteLine(this + " same1 as " + other);
        else
            Console.WriteLine(this + " not same1 as " + other);
        if (this.IsSame2(other))
            Console.WriteLine(this + " same2 as " + other);
        else
            Console.WriteLine(this + " not same2 as " + other);
    }
    public void Print() { Console.WriteLine(this.ToString()); }
    public void Print(Stam other) { this.Same(other); }
}
```

```
public class Davar: Stam
{
    private int y;
    public Davar(): base() { this.y = 0; }
    public Davar(char c): base(c) { this.y = 0; }
    public Davar(char c, int num): base(c) { this.y = num; }
    public override string ToString() { return "Davar: " + base.ToString(); }
}
```

א. במחלקה **Stam** הוגדרו שתי פעולות בונות.

(1) מהו העיקרון של תכנות מונחה עצמים (מבין העקרונות שצוינו בתחילת השאלה)

המאפשר זאת?

(2) כיצד בוחר המהדר (קומפיילר) איזו פעולה בונה להפעיל?

ב. איזה עיקרון של תכנות מונחה עצמים (מבין העקרונות שצוינו בתחילת השאלה)

מומש בפעולה `ToString()` במחלקה **Davar**? נמק.


```

public class Program
{
    public static void Main(string[] args)
    {
        Stam[] s = new Stam[6];
        s[0] = new Stam();
        s[1] = new Davar();
        s[2] = new Stam('b');
        s[3] = new Davar('b');
        s[4] = new Davar('a', 0);
        s[5] = s[2].GetStam();

        for (int i = 0; i < s.Length; i++)
        {
            s[i].Print();
        }

        s[1].Print(s[0]);
        s[2].Print(s[5]);
        s[3].Print(s[4]);
    }
}

```

קטע 1
 קטע 2
 קטע 3

- (1) הצג את המערך s הנבנה בקטע 1 (במחלקה הראשית הנתונה).
 בעבור כל אחד מהעצמים רשום את ערכי התכונות שלו.
- (2) רשום את פלט הלולאה בקטע 2.
- (3) איזה עיקרון של תכנות מונחה עצמים (מבין העקרונות שצוינו בתחילת השאלה) בא לידי ביטוי בקטע 2?
- (4) ממש את הפעולות IsSame1(), IsSame2() שבמחלקה Stam כך שהפלט המתקבל מקטע 3, יהיה הפלט שלהלן:

```

Davar: x = * same1 as x = *
Davar: x = * not same2 as x = *
x = b same1 as x = b
x = b same2 as x = b
Davar: x = b not same1 as Davar: x = a
Davar: x = b not same2 as Davar: x = a

```

```

public class AA
{
    private string st;

    public AA()                { this.st = "excellent"; }
    public AA(string st)      { this.st = st; }
    public string GetSt()     { return this.st; }
    public void SetSt (string st) { this.st = st; }
    public override string ToString() { return "st = " + this.st; }
}

public class BB : AA
{
    private int num;

    public BB() : base()      { this.num = 1; }
    public BB(int num, string st) : base(st) { this.num = Math.Abs(num); }
    public int GetNum()       { return this.num; }
    public void SetNum(int num) { this.num = num;}
    public override string ToString() { return base.ToString() + " num = "
                                        + this.num;}
}

```

- א.** הגדר במחלקה **AA** פעולה בוליאנית הניתנת לדריסה, בשם `IsLike (Object obj)`, המקבלת עצם `obj` מטיפוס `Object`. אם העצם `obj` הינו מטיפוס **AA** וגם תוכן המחרוזת `st` של `obj` זהה לתוכן המחרוזת `st` של העצם הנוכחי – הפעולה תחזיר `true`, אחרת – תחזיר `false`.
- ב.** הגדר במחלקה **BB** פעולה הדורסת את הפעולה שהגדרת בסעיף א. אם העצם `obj` הינו מטיפוס **BB** וגם ערך התכונה `num` שלו זהה לערך התכונה `num` של העצם הנוכחי – הפעולה תחזיר `true`, אחרת – תחזיר `false`.

```
AA a = new AA("excellent");  
BB b = new BB();  
a = b;  
if (a.IsLike(b)) Console.WriteLine(a);
```

האם קטע התכנית תקין?

אם כן – מה יהיה פלט הקטע? רשום איזו גרסה של הפעולה IsLike תופעל – זו של AA או זו של BB.
אם לא – הסבר מהי השגיאה ומתי היא תתגלה: בזמן קומפילציה או בזמן ריצה.

```
AA aa = new AA();  
BB bb = new BB(2, "excellent");  
bb = aa;  
if (bb.IsLike(aa)) Console.WriteLine(bb);
```

האם קטע התכנית תקין?

אם כן – מה יהיה פלט הקטע? רשום איזו גרסה של הפעולה IsLike תופעל – זו של AA או זו של BB.
אם לא – הסבר מהי השגיאה ומתי היא תתגלה: בזמן קומפילציה או בזמן ריצה.

הפעולה מחזירה מחרוזת המורכבת משרשר התכונה st של עצמים מטיפוס AA במערך, באופן הזה:

- אם לעצם יש ב התכונה st, תשורשר המחרוזת שבתכונה st פעם אחת.
- אם לעצם יש ג התכונה num, המחרוזת שבתכונה st תשורשר num פעמים.
- אם אין במערך אף עצם מטיפוס AA, תוחזר מחרוזת ריקה.

לפניך פרויקט ובו המחלקות **A**, **D**, **B** ו-**OopTest**:

```
public class B
{
    private static int numB = 0;
    private int m1;
    private int m2;

    public B(int m1, int m2)
    {
        this.m1 = m1;
        this.m2 = m2;
        numB++;
        Console.WriteLine("B(" + m1 + ", " + m2 + ") , #" + numB);
    }
}

public class D : B
{
    private static int numD = 0;
    private double d;

    public D(double d, int x) : base(x, x)
    {
        this.d = d;
        numD++;
        Console.WriteLine("D(" + d + ", " + x + ") , #" + numD);
    }

    public D(double d, int x, int y) : base(x, y)
    {
        this.d = d;
        numD++;
        Console.WriteLine("D(" + d + ", " + x + ", " + y + ") , #" + numD);
    }
}
```

```

public class A
{
    private static int numA = 0;
    private A a;
    private B b;

    public A(A a , B b)
    {
        this.a = a;
        this.b = b;
        numA++;
        Console.WriteLine("A Constructor , #" + numA);
    }
}

```

```

public class OopTest
{
    public static void Main(string[] args)
    {
        B w1 = new B(2 , 3);
        B w2 = new D(1.5 , 6);
        B w3 = new D(2.3 , 8 , 9);
        A w4 = new A(null , w1);
        A w5 = new A(w4 , w3);
    }
}

```

כתוב מעקב אחר הפעולה Main במחלקה **OopTest** , וכתוב את הפלט.
 במעקב יש לכתוב את ערכי המשתנים, ובעבור כל עצם – את ערכי התכונות שלו.