המרכז האקדמי לב

מחלקה למדעי מחשב, הנדסת תכנה והנדסת מערכות תקשורת

קורס 153007

מיני פרויקט במערכת חלונות

מצגת הקורס

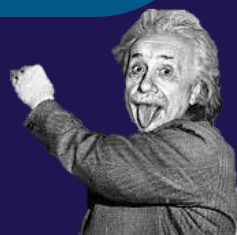
פרק 4 #C נושאים מתקדמים – המשך

תשפ״ג סמסטר א׳

דן זילברשטיין תשפ״ג 2022/23

מה עכשיו?

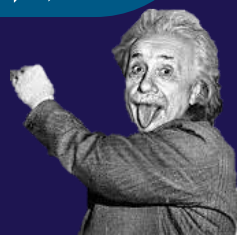
- חריגות – סקירה מהירה ולימוד עצמי
- השלמות...
- דלגטים (delegates)
- אירועים (events)
- תבנית פיתוח "משקיף" (Observer)
- UML



לימוד עצמי

חריגות

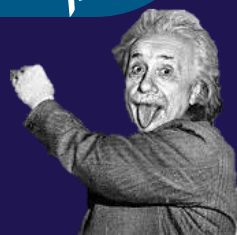
- **קראו מסמך על חריגות בחומרי הקורס**
- תוצאה של פסיקת חומרה או פסיקה פרוגרמטית
- נכתוב תוכנית קצרה שמייצרת חריגה, למשל מ-parsing של מחרוזת למספר
- נחקור את הכתוב בקונסול
- נחקור את מה שקיבלנו בסביבת הפיתוח
- אפשר לראות רשימת חריגות שקריאה לפונקציה יכולה לייצר
- בדרך כלל נרצה למנוע את קריסת התוכנית שלנו



לימוד עצמי

איך תופסים חריגות

- בלוק **try**
- אחד או יותר בלוקים **catch** עם פרמטר של סוג חריגה
- בלוק אחרון (אם נרצה) עם סוג **Exception**
- בלוק **finally** (לא חובה) שמתבצע **תמיד** לפני סיום אחד הבלוקים הנ"ל (גם אם היה לנו return\break\goto)
- אפשר להשתמש בחריגות קיימות (נלחץ Ctrl-Alt-E) או ליצור חריגות משלנו
- ביצירת חריגה משלנו ניצור מחלקה שיורשת חריגה קיימת, ניתן שם שמסתיים במילה **Exception**...

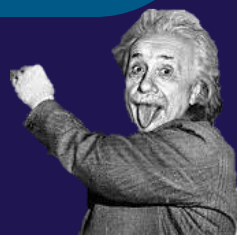


לימוד עצמי

אם לא תפסנו חריגה?

- יש לנו בלוקים `try\catch\finally`
- קרתה חריגה מסוג שלא תפסנו
- הפעלה או אי הפעלה של `finally` תלויה ביישום וקינפוג של ה-CLR
- ראו באתר של מיקרוסופט

https://docs.microsoft.com/en-us/dotnet/framework/performance/reliability-best-practices



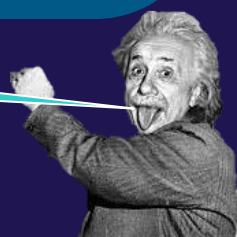
לימוד עצמי

עוד על חריגה

- אם לא הייתה חריגה או אם תפסנו את החריגה – ממשיכים הלאה בתוכנית
  - אם לא עשינו return או throw
- זריקת חריגה – `throw new MyException(…);`
- בתוך `catch` אפשר לזרוק אותה חריגה הלאה: `throw;`
- או לזרוק חריגה חלופית כנ"ל
  - אפשר לצרף חריגה מקורית בפרמטר
- מה יש בתוך החריגה?
  - נתבונן במחלקה Exception

זריקת **Exception**

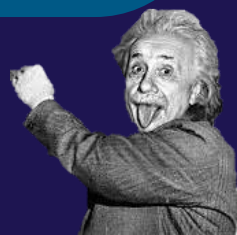
פטור אבל אסור!!!



לימוד עצמי

יצירת סוג חריגה

- בדרך כלל ירושה ישירות מ-**Exception** או מאחת החריגות הכי כלליות
- בשורה שלפני מחלקה נרשום אטריבוט **[Serializable]**
- נממש ממשק **ISerializable** (אין מה לממש – גם לא צריך לרשום – מחלקת **Exception** כבר עשתה את העבודה)
- אפשר להוסיף שדות משלנו לשמירת מידע נוסף – אם רוצים אפשר להוסיף אטריבוט **[NonSerialized]**
- אפשר לשכתב (override) את הפונקציה **ToString()**



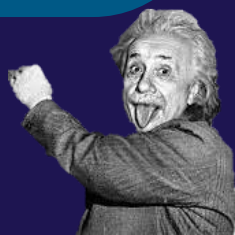
לימוד עצמי

יצירת סוג חריגה - בנאים

- בדרך כלל תממשו לפחות את שלושת הבנאים הבאים:

```
public MyException() :base() {...}
public MyException(string message) : base(message) {...}
public MyException(string message,
                   Exception inner) : base(message, inner) {...}
protected MyException(SerializationInfo info,
                      StreamingContext context)
                     : base(info, context) {...}
```

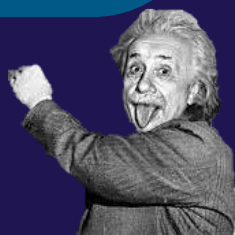
- אם יש שדות שהוספתם – כנראה תרצו להוסיף עוד בנאים



לימוד עצמי

יצירת סוג חריגה - דוגמא

```
[Serializable]
public class OverloadCapacityException : Exception
{
    public int capacity { get; private set; }

    public OverloadCapacityException() : base() { }
    public OverloadCapacityException(string message) : base(message) { }
    public OverloadCapacityException(string message, Exception inner) : base(message, inner) { }
    protected OverloadCapacityException(SerializationInfo info, StreamingContext context)
                            : base(info, context) { }

    // special constructor for our custom exception
    public OverloadCapacityException(int capacity, string message)
                                : base(message) => this.capacity = capacity;

    override public string ToString() =>
                    "OverloadCapacityException: + capacity + "\n" + Message;
}
```

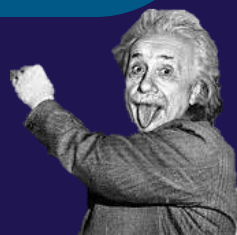
זריקת חריגה בביטוי

- בגרסה מודרנית של C# ניתן לזרוק חריגה מתוך ביטוי
- שימושי בביטוי עם אופרטור טרנארי:

```
MyClass obj = new();
int status = obj != null ? obj.Status
                         : throw new MyBadObjectException();
```

- או ב-coalescing ??:

```
MyClass obj2 = new() ?? throw new MyBadObjectException();
```



אובייקטים מסוג אנונימי

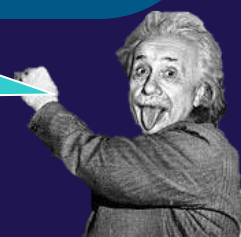
- כמו יצירת מופע עם אתחול מהיר ללא שם מחלקה

```
new { ID = 29, Name = "Dani", Age = 48 }
```

- לומד את השדות מתוך השמות והסוגים באתחול (בדומה ל-var)
- בפועל מייצר שדות פרטיים ותכונות עם get בלבד
- ניתן להציב למשתנה מסוג מרומז

```
var lecturer = new { ID = 29, Name = "Dani" };
```

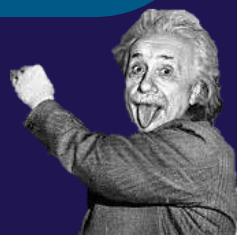
ניתן להשתמש בכל מקום שצריך להעביר **object**



סוג אנונימי מבפנים

- נפעיל את הפונקציה PrintInfo על סוג רגיל וסוג אנונימי עם אותם שדות Id ו-Name. נתבונן בפלט...

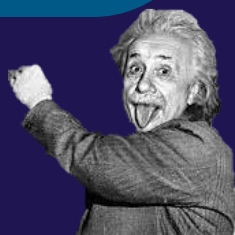
```
class MyClass
{
   public int Id;
   public string Name;
}
static void Main()
{
   PrintInfo("", typeof(MyClass));
   Console.WriteLine("-----------------------------------");
   var anonymousObject = new { Id = 2222, Name = "Yossi" };
   PrintInfo("", anonymousObject.GetType());
   Console.WriteLine("-----------------------------------");
}
```



Reflection – שימוש ב-metadata

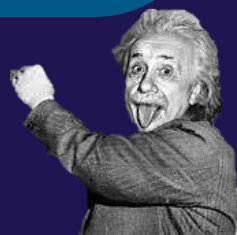
- נלמד סוג אנונימי בעזרת reflection

```
using System.Reflection;
...
static void PrintInfo(string suffix, Type type)
{  Console.WriteLine(suffix + "Type Name: " + type.Name);
   Console.WriteLine(suffix + "Base Type: ");
   if (type.BaseType == null)
      Console.WriteLine(suffix + suffix + "None");
   else
      PrintInfo(suffix + "  ", type.BaseType);
   Console.WriteLine(suffix + "Member Info:");
   MemberInfo[] members = type.GetMembers();
   foreach (var item in members)
      Console.WriteLine(suffix + "name: {0,-15} type: {2,-11} in: {1}",
                        item.Name, item.DeclaringType.Name, item.MemberType);
}
```



סוג אנונימי – כמה יש?

- נגדיר שני מופעים אנונימיים עם אותם שדות ונדפיס את שם הסוג
  - אותו סוג
- אם לשני המופעים הנ"ל גם הערכים שווים?
  - אותו קוד גיבוב (hashvalue)
  - שוויון ב-Equals
  - אי שוויון ב-"=="

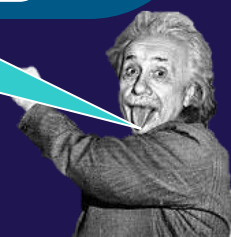


Extension methods – פונקציות הרחבה

- מיוחד ל-#C
- מאפשר להוסיף פונקציות למחלקה קיימת בלי לגעת בה
- בעזרת מחלקה סטטית

```
static class MyTools
{
   public static int ToInt(this string str)
   {
      return int.Parse(str);
   }
}
...   string str = "123";
...   int i = str.ToInt();
```

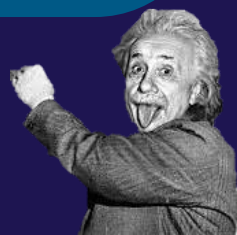
מה נראה אם נקליד "123".ToInt()?



פונקציות הרחבה גנרית – דוגמא עם reflection

- הדפסת שדות וערכיהם של אובייקט כלשהו

```
static class Tools
{
   public static void ToStringProperty<T>(this T t)
   {
      string str = "";
      foreach (PropertyInfo item in t.GetType().GetProperties())
         str += "\n" + item.Name + ": " + item.GetValue(t, null);
      Console.WriteLine(str);
   }
}

... DateTime.Now.ToStringProperty();
```



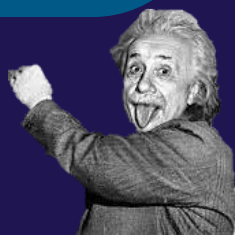
סוגי ערך מתאפסים – nullable value types

- ניתן להשתמש בערך **null** רק עם משתנים מסוגי הפניה
- בסוגי ערך אין לנו מצב של "אין ערך"
- אבל יש מצבים שהיינו רוצים את זה בלי לתת ערך מיוחד
- בעזרת סופיקס "**?**": **int?**, **char?**

```
int? a = null;
```

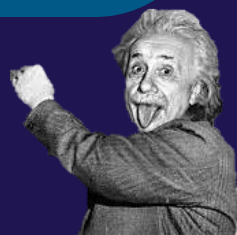
- המרה מסוג רגיל לסוג מתאפס – מרומזת ומפורשת
- המרה מסוג מתאפס לסוג רגיל – רק מפורשת
- אופרטור "**??**"

```
int b = a ?? 10;
```



סוגי ערך מתאפסים – שינויים ב-C# 10

- שימוש בטיפוסי הפנייה "מתאפסים"
- היערות רבות של קומפיילר
  - יש להרבות בשימוש בסוגים מתאפסים – בפרמטרים, ארגומנטים גנריים, משתנים מקומיים, וכו'
  - יש להרבות בשימוש באופרטורים "?" ו- "??"

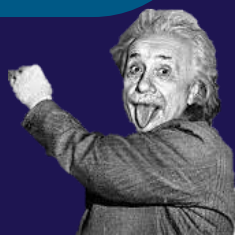


לא חובה

סוגי ערך מתאפסים מאחורי הקלעים

- הצורה של T? היא חלופה ל-Nullable<T>

```
public struct Nullable<T> where T : struct
{
   public Nullable(T value);
   public static explicit operator T(Nullable<T> value);
   public static implicit operator Nullable<T>(T value);
   public bool HasValue { get; }
   internal T value;
   public T Value { get; }
   public override bool Equals(object other);
   public override int GetHashCode();
   public T GetValueOrDefault();
   public T GetValueOrDefault(T defaultValue);
   public override string ToString();
}
```



יצירת תיעוד ב-C# (בעתיד יעבור למצגת 2)

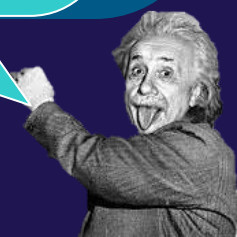
- הערות ב-C# בדומה ל-C++ בעזרת:

```
… code … // comments …
/* …
   … */
```

- יצירת תבנית הערות אוטומטית בעזרת "///" לפני מחלקות ופונקציות

```
/// <summary>
/// ...
/// </summary>
/// <param name="...">...</param>
/// <param name="...">...</param>
/// <returns>...</returns>
```

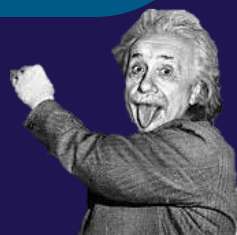
נראה מה קורה בויזיואל סטודיו לאחר שהוספנו הערות /// ואנחנו רוצים להשתמש בפונקציה



הגדרת משתנים מסוג דינמי – dynamic

לימוד עצמי

- שימוש לא זהיר מסוכן ביותר
- לקריאה וחקירה עצמאית
  - במסמכי OSF
  - ב-msdn
  - נסו – **Try it**
  - חפשו – **Google it**



דלגטים – delegate

- מגדיר **סוג** של אוסף של מעין "מצביעים לפונקציה":

```
public delegate int SomeDelegate(int parm1, string parm2);
```

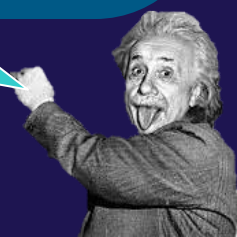
- מכיוון שזה סוג – ניתן להגדיר מחוץ למחלקות
- הסוג החדש יורש ממחלקה MulticastDelegate שיורש בתורו ממחלקה Delegate
- אפשר להגדיר משתנים ושדות עם הסוג החדש

```
SomeDelegate myDelegate = null;
```

- מבצעים קריאה לדלגט כמו קריאה לפונקציה

```
int i = myDelegate(3, "abc");
```

עוד מעט נראה איך מוסיפים פונקציות אמיתיות בדלגט



דלגטים – הצבת ערכים

- מאתחלים עם מופע ע"י בנאי עם שם פונקציה עם <u>אותה חתימה בלבד</u>

```
public delegate void MyDelegate();
void MyFunc1() { ... }
MyDelegate myDelegate = new MyDelegate(MyFunc1);
```

- אפשר גם להשמיט את יצירת המופע (המהדר יבין)

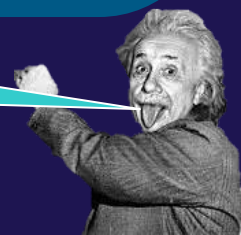
```
myDelegate = MyFunc1;
```

- אמרנו שזה אוסף... אפשר להוסיף או למחוק פונקציות

```
myDelegate += MyFunc2;
myDelegate -= MyFunc1;
```

- בהפעלה של דלגט יופעלו כל הפונקציות שנרשמו!!!

נעשה דוגמה עם **reflection** מה-**OSF**



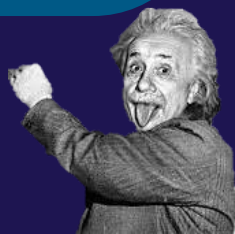
דוגמא עם Reflection עבור דלגט

```
public delegate int SomeDelegate(int x, int y);
class Program
{
  static public int sum(int num1, int num2) => num1 + num2;
  static public int mult(int num1, int num2) => num1 * num2;
  static public int sub(int num1, int num2) => num1 - num2;
  static void Main(string[] args)
  {
    SomeDelegate myDlgt = new SomeDelegate(sum);
    myDlgt += mult; myDlgt += sub; myDlgt -= sum;

    foreach (var d in myDlgt.GetInvocationList()) Console.WriteLine(d.Method);
    if(myDlgt is Delegate) Console.WriteLine("myDlgt is Delegate == true");
    foreach (var item in myDlgt.GetInvocationList()) // (Delegate item …)
      Console.WriteLine( item.DynamicInvoke(3, 2) );
  }
}
```

ב-Delegate גולמי אין מתודה Invoke()

דלגטים – למה?

- כמו מצביע לפונקציה ב-++C
- אפשר לעשות רשימה של "מצביעים לפונקציה"
- מאפשר callback כמו עם מצביע לפונקציה
- נותן כלי חזק ליישום של OCP (נראה דוגמא)
- נותן כלי ליישום של DIP ומאפשר מימוש פשוט של תבנית תיכון Observer שנדבר עליהם כשנלמד אירועים event -



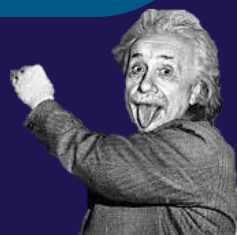
כמה סוגי דלגטים שכבר מוגדרים ב-.NET

- פרדיקט

```
public delegate bool Predicate<T>(T obj);
```

- שימוש בפונקציות אוספים כמו **Find**, **FindAll**
- עוד מעט נראה שימוש בפונקציה אנונימית וביטוי למבדה
- ממיר

```
public delegate TOutput Converter<TInput,TOutput>(TInput input);
```



עוד כמה סוגי דלגטים שכבר מוגדרים ב-.NET

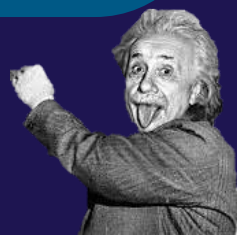
- פעולה

```
public delegate void Action();
public delegate void Action<T1,T2,…>(T1 arg1, T2 arg2, …);
```

- "פונקציות"

```
public delegate TResult Func<Tresult>();
public delegate TResult Func<T1,T2,…, Tresult>(T1 arg1, T2 arg2, …);
```

- שתיהן מוגדרות עם 0 עד 16 פרמטרים מסוגים שונים



פונקציה אנונימית בעזרת דלגט

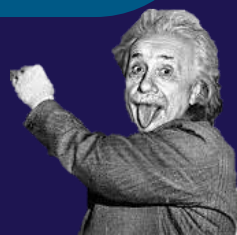
- במקום להגדיר פונקציה ממשית ולהשתמש בה כפרמטר ניתן להגדיר פונקציה אנונימית ישירות:

```
List<int> list1 = ...;
List<int> list2;
list2 = list1.FindAll(delegate(int x) { return x % 2 == 0; });
```

- אפשר גם באתחול והוספה למשתנים\שדות דלגטים

```
SomeDelegate myDlgt = delegate(int x, int y) { return x + y; };
myDlgt += delegate(int x, int y) { return x * y; };
```

- נותן חיסכון בקוד ושיפור נהירות הקוד



ביטוי למבדה

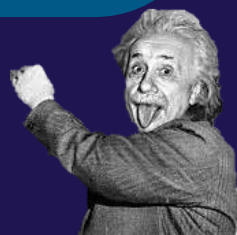
```
list2 = list1.FindAll(delegate(int x) { return x % 2 == 0; });
```

- עוד חיסכון בקוד ושיפור נהירות הקוד
- ע"י שימוש בביטוי למבדה עם סימן "=>"

```
list2 = list1.FindAll( x => x % 2 == 0 );
```

- אם יש כמה פרמטרים?
  - נשתמש בסוגריים ()
- אפשר להשתמש גם באתחול דלגט

```
public delegate int SomeDelegate(int x1, int x2);
... SomeDelegate MyDelegate = (x,y) => x + y;
```



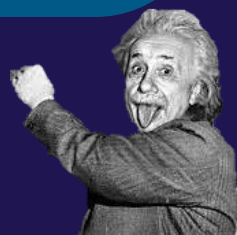
ביטוי למבדה – עוד מקרים

- אם אין פרמטרים?
  - נשתמש בסוגריים () ריקות

```
... SomeDelegate MyDelegate = () => "Best performance!";
```

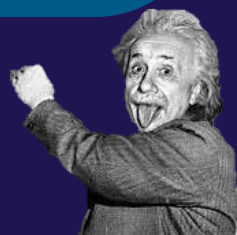
- אם הקוד קצת יותר כבד?
  - נחזיר למקומם את הצומדיים ואת הוראת return

```
... SomeDelegate MyDelegate = x => { x += 13; return x <= 1000; };
```



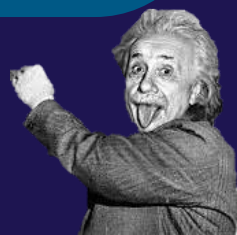
callbacks – DIP

- אירועי חומרה (לחיצת עכבר וכו')
- מקור – מערכת הפעלה
- מטרה – להגיב בתוכנית שלנו
- מערכת הפעלה לא מכירה את התוכנית שלנו...
- לא נשנה את מערכת הפעלה
- צריך דרך לרשום פונקציה שלנו לאירוע מערכת הפעלה
- נוגע לתכנון כל מערכת תוכנה והרחבותיה



ניסיון ראשון - מדפסת

```
public delegate void PrintEventHandler();
public class MyPrinter
{
   public PrintEventHandler PageOver = null;
   private int pageCount = 20;
   private void handlePageOver()
   //                    { if (PageOver != null) PageOver(); }
                                 => PageOver?.Invoke(this);
   public void Print(int pages)
   {
      if (pages <= pageCount) pageCount -= pages;
      else { pageCount = 0; handlePageOver(); }
   }
}
```

ניסיון ראשון - משתמשים

```
class User
{
   MyPrinter myPrinter;
   public User(MyPrinter printer)
   {
      myPrinter = printer;
      printer.PageOver = myPageOver;
   }
   private void myPageOver()
   {
      ...
   }
}
```

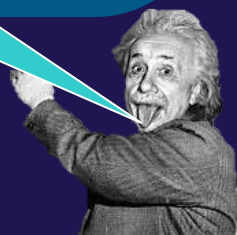
לא נגענו בתוכנת המדפסת!

אם יש כמה משתמשים –
הם דורסים אחד לשני

ניסיון שני - משתמשים

```
class User
{
   MyPrinter myPrinter;
   public User(MyPrinter printer)
   {
      myPrinter = printer;
      printer.PageOver += myPageOver;
   }
   private void myPageOver()
   {
      ...
   }
}
```

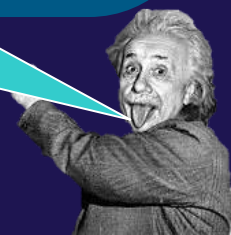
מה קורה לראשון?



ניסיון שני - ראשי

```
class Program
{
    static void Main(string[] args)
    {
        MyPrinter printer = new MyPrinter();
        User u1 = new User(printer);
        User u2 = new User(printer);
        printer.PageOver();
        Console.WriteLine("Please enter pages to print:");
        int x = int.Parse(Console.ReadLine());
        printer.Print(x);
    }
}
```

פריצה: אפשר להפעיל ללא רשות!

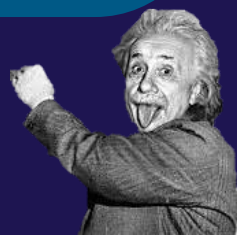


אירועים – events

- ניתן להוסיף להגדרת שדה מסוג דלגט

```
public event PrinterEventHandler PageOver = null;
```

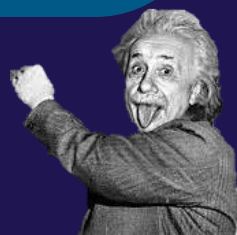
- בתוך המחלקה אפשר לעשות כל דבר כמו עם דלגט רגיל
- מחוץ למחלקה ניתן לבצע רק פעולות **+=** ו-**-=**



אירועים – events – המשך

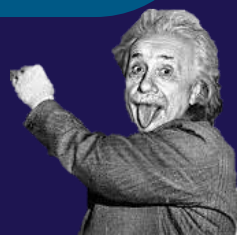
- לאחר הגדרת השדה כאירוע קיבלנו שגיאות קומפילציה:
  - בניסיון ראשון של משתמש – "="
  - בניסיון הפעלה ב-Main עם הפעלת הדלגט
- יתרונות נוספים בסביבת פיתוח:
  - "ברק" מסמן שדות מסוג אירוע
  - תנסו ללחוץ TAB לאחר הוספה לאירוע

```
myPrinter.PageOver +=<TAB>
```



ניסיון שלישי - מדפסת

```
public delegate void PrintEventHandler;
public class MyPrinter
{
   public event PrintEventHandler PageOver = null;
   private pageCount = 20;
   private void handlePageOver() => PageOver?.Invoke(this);
   public void Print(int pages)
   {
      if (pages > pageCount) pageCount -= pages;
      else { pageCount = 0; handlePageOver(); }
   }
}
```



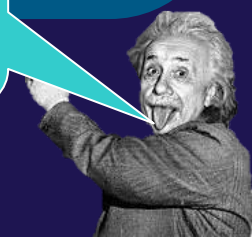
דלגט עבור טיפול באירועים – EventHandler

- מוגדר בספרית NET.

```
public delegate void EventHandler (object sender, EventArgs e);
```

- ניתן להעביר את האובייקט של שולח האירוע
- EventArgs – מחלקה ריקה (כמעט)
  - שדה סטטי Empty המכיל אובייקט "ריק" של ארגומנטים
  - ניתן להוסיף מידע פרטני על האירוע בעזרת ירושה ממנה
  - בדרך כלל נוסיף שדות readonly שנאתחל בבנאי
  - ניתן להוסיף שדה "מצב" המסמן שמישהו כבר טיפל באירוע

בפונקציה שנרשום לשדה הדלגט ("המשקיף" על האירוע) צריך לבדוק את הסוג ולהמיר...

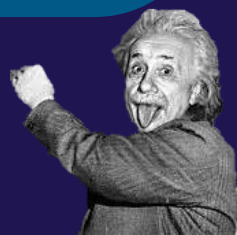


דלגט גנרי – EventHandler<T>

- מוגדר בספרית NET.

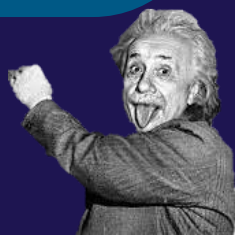
```
public delegate void EventHandler<TEventArgs>
                                    (object sender, TEventArgs e);
```

- עכשיו לא צריך המרות ובדיקות – הקומפיילר יעשה את העבודה



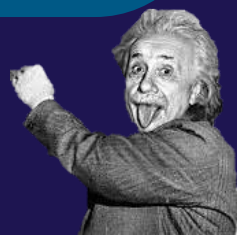
Observable – משקיף תקני

```
public class ValueChangedArgs : EventArgs
{
   public readonly int OldV; public readonly int NewV;
   public ValueChangedArgs(int oldV, int newV)
      { OldV = oldV; NewV = newV; }
}
public class MyValue
{
   private int value = 0;
   public event EventHandler<ValueChangedArgs> OnValueChanged = null;
   private void valueChangedHandler(int oldV, int newV)
   {
      if (onValueChanged != null)
         onValueChanged(this, new ValueChangedArgs(oldV, newV));
   }
```



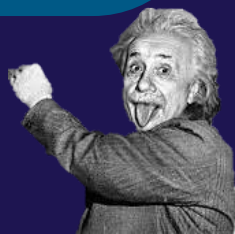
משקיף תקני – Observable – המשך

```
    public int Value
    {
        get { return value; }
        set
        {
            if (value != this.value)
            {
                int temp = this.value;
                this.value = value;
                valueChangedHandler(temp, value);
            }
        }
    }
}
```



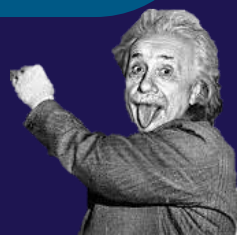
משקיף תקני – Observer

```
public class ValueChangeObserver
{
    public ValueChangeObserver(MyValue v)
    {
        v.OnValueChanged += myOnValueChanged;
    }
    private void myOnValueChanged(object sender, ValueChangedArgs args)
    {
        Console.WriteLine("Value changed by {0}",
                          args.NewV - args.OldV);
    }
}
```



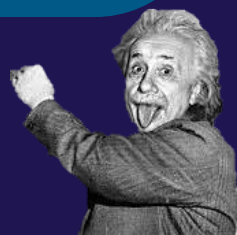
משקיף תקני – Observer נוסף

```
public class ValueAverageChangeObserver
{
   private int sum = 0, count = 0;
   public ValueAverageChangeObserver(MyValue v)
   {
      v.OnValueChanged += myOnValueChanged;
   }
   private void myOnValueChanged(object sender, ValueChangedArgs args)
   {
      count++; sum += args.newV - args.oldV;
      Console.WriteLine("Value average change is {0:F}",
                        (double)sum / (double)count);
   }
}
```



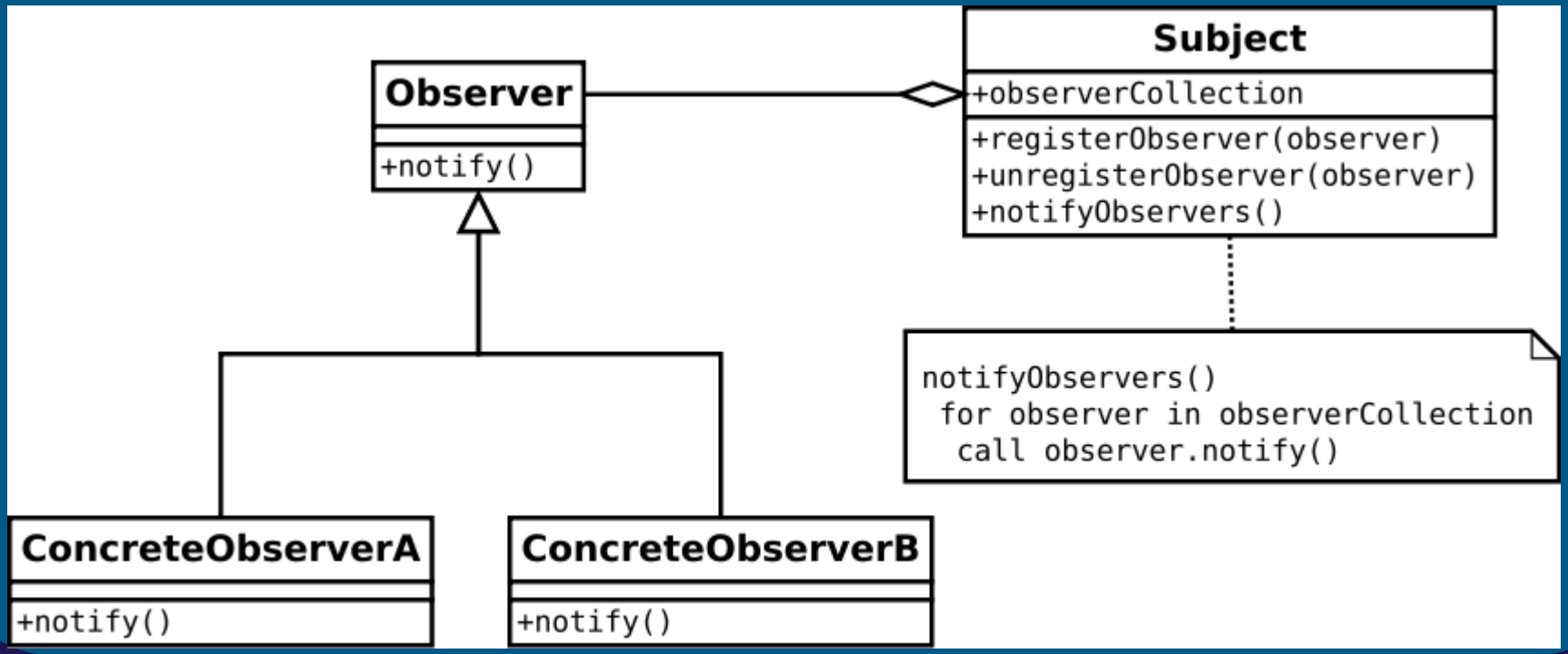
משקיף תקני – בדיקה

```
public class Program
{
    static void Main(string[] args)
    {
        MyValue v = new MyValue();

        new ValueChangeObserver(v);
        new ValueAverageChangeObserver(v);

        v.Value = 100;
        v.Value = 210;
        v.Value = 150;
        v.Value = 180;
    }
}
```



תבנית המשקיף

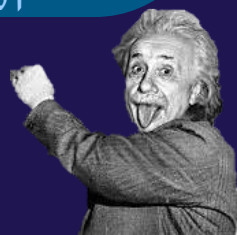
- Subject = Observable
- unregister = Remove ,register = Add



Unified Modeling Language – UML

- להראות ארכיטקטורה של מערכת עייי דיאגרמות
- Static (structural) diagrams
  - Class Diagram
  - Composite Structrure Diagram
  - Component Diagram
  - Object Diagram
  - Package Diagram
  - Profile Diagram
- Behavioral Diagrams
  - Activity
  - State / Statechart
  - Sequence
  - וכו'

לא למבחן

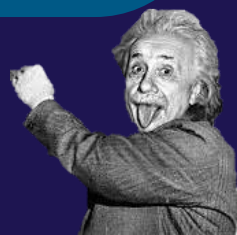


לא למבחן

Class Diagrams – UML

BankAccount
owner : String balance : int = 0
deposit (amount : int) withdrawal (amount : int) : bool

- מחלקה – מלבן
- חלק עליון – שם המחלקה
- חלק אמצעי – שדות (attributes) של המחלקה
  - שם
  - סוג
  - אתחול (אפשרי)
- חלק תחתון – מתודות או פעולות של המחלקה
  - עם פרמטרים לסוגיהם וסוג ערך מוחזר (עם יש)

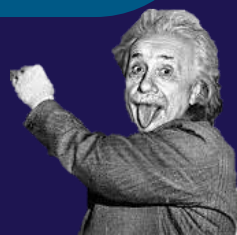


UML הרשאת גישה ותחום (מופע\מחלקה)

- הרשאות גישה
  - public +
  - private -
  - protected #
- classified (static)

BankAccount
+owner : String #balance : int = 0 -counter : int
+deposit (amount : int) +withdrawal (amount : int) : bool +showCounter() : int

לא למבחן

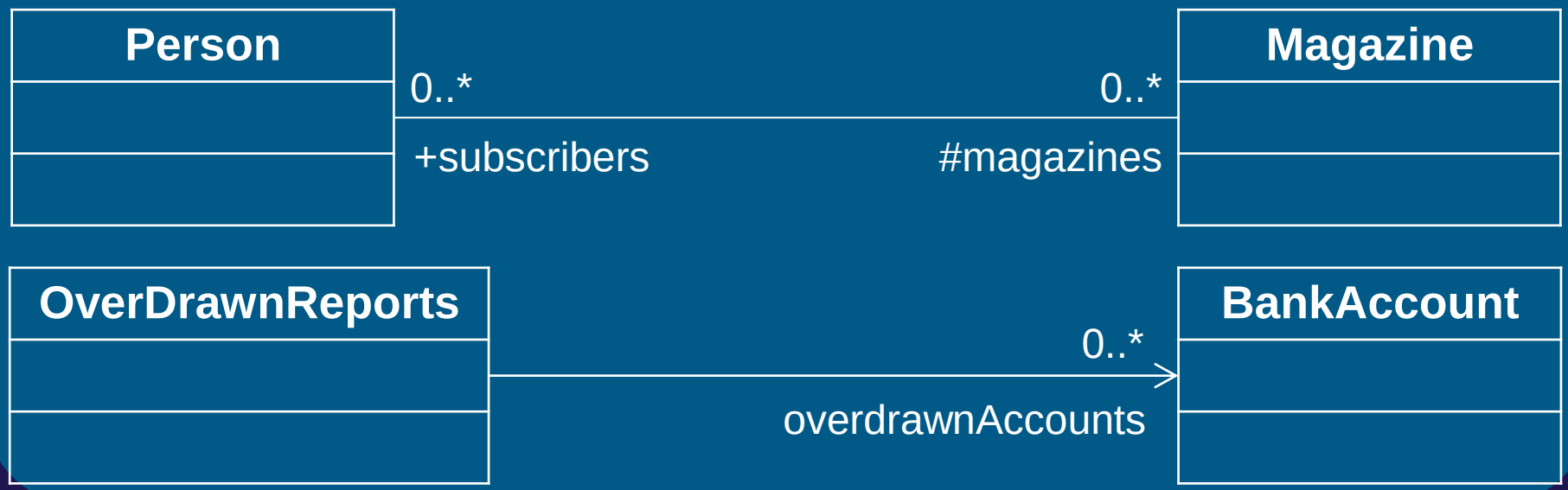


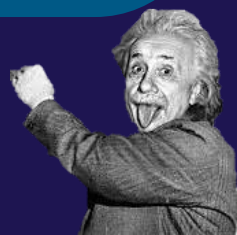
לא למבחן

UML – קשרים אסוציאטיבים

- קשרים
  - association
    - bi-directional – קו בלי חצים
    - uni-directional – קו עם חץ

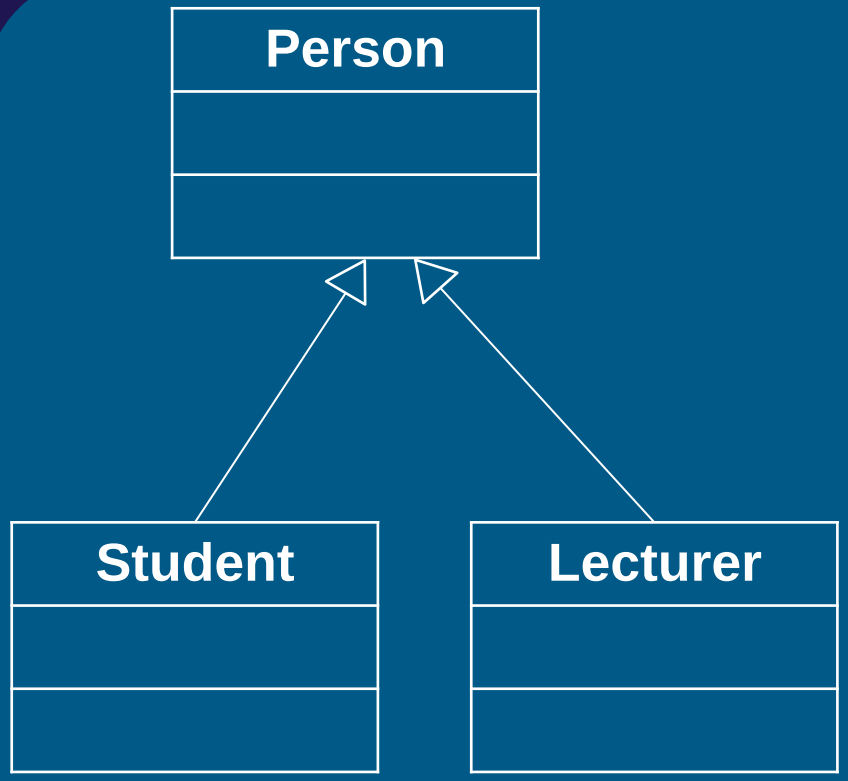
0..1		Zero or One
1		One
0..*	*	Zero or more
1..*		One or more
3		Three
2..6		Two to Six



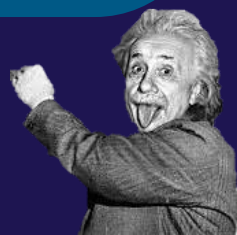


לא למבחן

UML – ירושה



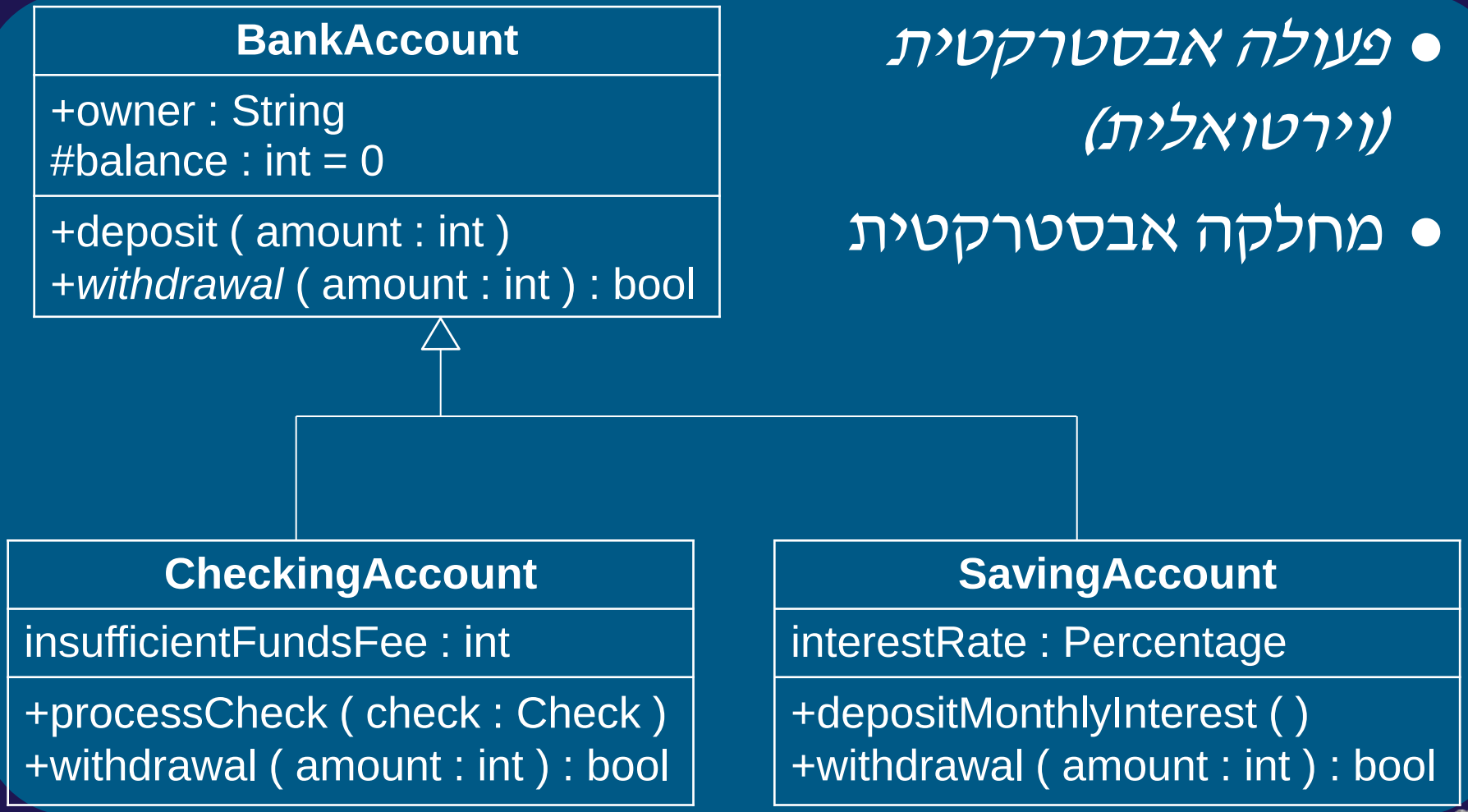
- Generalization
  - IS-A
- על-מחלקה (מחלקת אב) הנה הכללה של תת-מחלקה
- ירושה – inheritance



לא למבחן

UML – מחלקה\פעולה אבסטרקטית

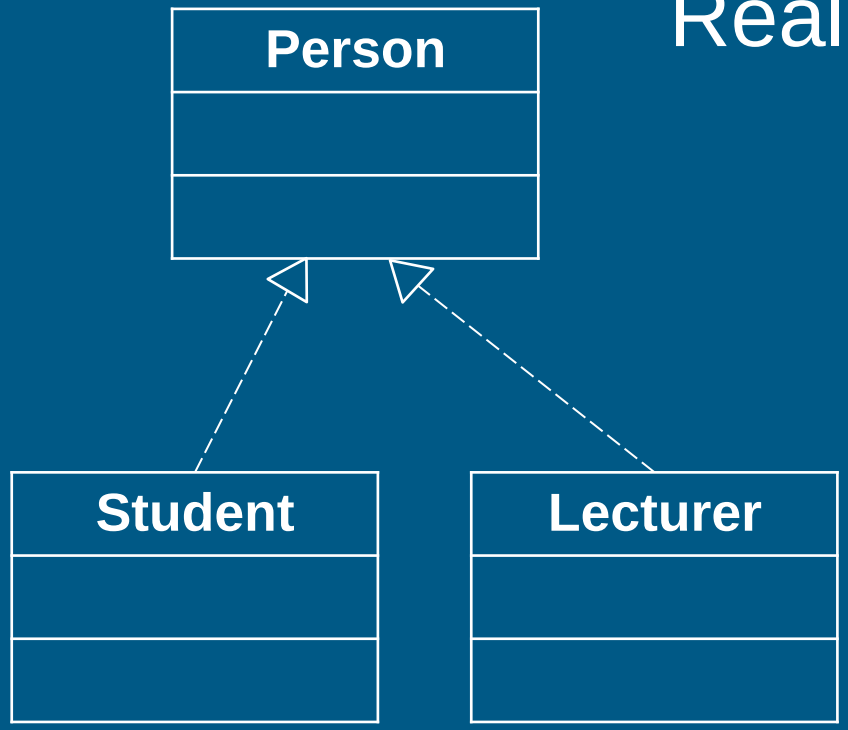
- *פעולה אבסטרקטית (וירטואלית)*
- מחלקה אבסטרקטית

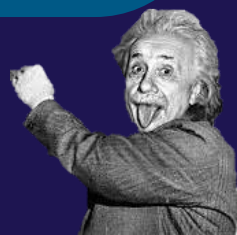


לא למבחן

UML – יישום

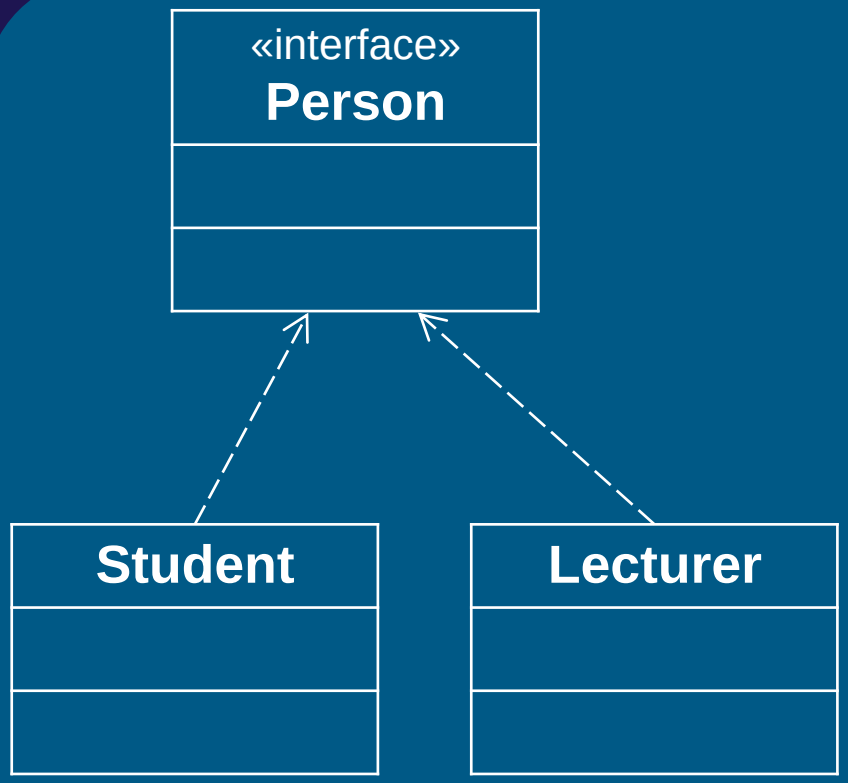
- Realization / Implementation
- יישום \ מימוש של מחלקה אבסטרקטית



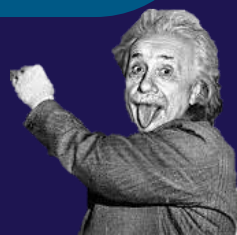


לא למבחן

UML – מימוש

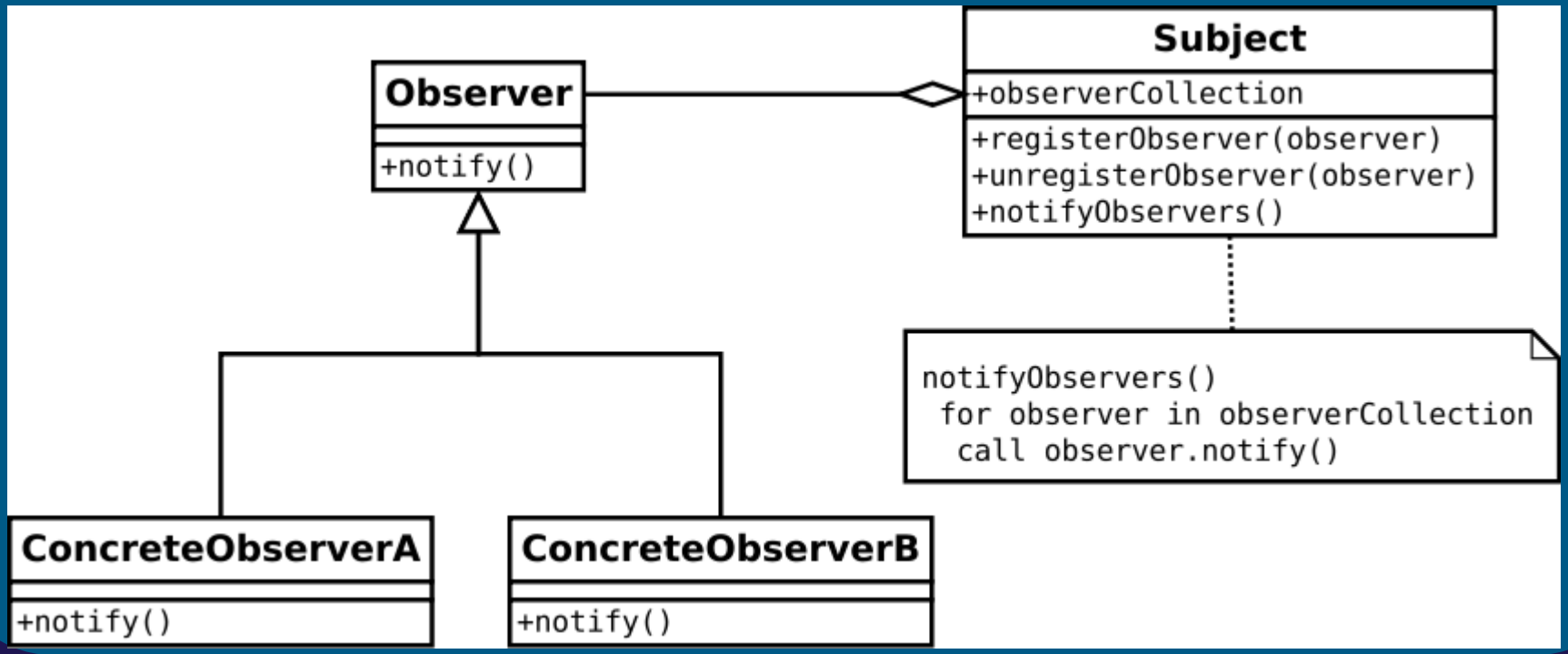


- Interface Realization
- מימוש ממשק



תבנית המשקיף

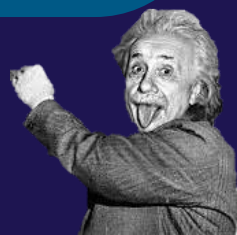
- Subject = Observable
- unregister = Remove ,register = Add



תבנית המשקיף ללא דלגטים ואירועים – ממשקים

```
public interface IObserver<T>
{
   void Notify(T parm);
}
public interface IObservable<T>
{
   void AddObserver(IObserver<T>);
   void RemoveObserver(IObserver<T>);
}
```

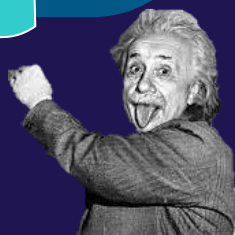
לא חובה



תבנית המשקיף ללא דלגטים ואירועים – מושקף

```
public class MyObservable : IObservable<int>
{
   private List<IObserver<int>> observers
                                    = new List<IObserver<int>>;

   public void AddObserver(IObserver<int> obs)
      { observers.Add(obs); }
   public void RemoveObserver(IObserver<int> obs)
      { observers.Remove(obs); }

   private void notifyAll(int parm)
      { foreach (var item in observers) item.Notify(parm); }

    ... notifyAll(data); ...
}
```

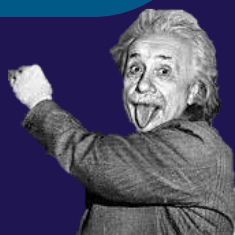
לא חובה



תבנית המשקיף ללא דלגטים ואירועים – משקיף

```
public class MyObserver1 IObserver<int>
{
   private IObservable myObservable;
   public MyObserver1(IObservable obs)
   {
      myObservable = obs;
      myObservable.AddObserver(this);
   }
   ~MyObserver1() { myObservable.RemoveObserver(this); }
   public void Notify(int parm)
   {
      Console.WriteLine("Got update: {0}", parm);
   }
}
```

לא חובה



הגדרת property ל-event

- ניתן להוסיף תכונה (property) לשדה של event
- יש להשתמש ב-add ו-remove

```
private event EventHandler<...> myEvent;

public event EventHandler<...> MyEvent
{
   add { ... myEvent += value; }
   remove { ... myEvent -= value; }
}
```