מעבר מ- java ל- C#:

1 רקע:

שפת c# - שפת תכנות שפותחה ע"י מיקרוסופט ונחשבת לאחת משפות התכנות הפופולריות בעולם. היא מיועדת לפיתוח כללי של מגוון אפליקציות בכל התחומים – מאתר Web , דרך משחקים, מאפליקציות למכשירי מובייל וטאבלטים ועד לשירותי ענן.
השפה נחשבת לשפה מונחת עצמים שלימה. הכוונה: בניגוד לשפות כמו++C או JAVA כל האובייקטים בשפה (כולל האובייקטים 'הפרימיטיביים' כמו int , char...) הם מחלקות (עם מתודות).
C זהה מאוד ל java בין מבחינת סינטקס ובין מבחינת מבנה השפה, ולכן היא קלה מאוד למי שלמד בעבר(או לפחות מכיר) את השפות ++c, c ו- java שהן המקור לשפה.
בעמודים הבאים נעסוק בהבדלים העיקרים שבין השפות, נכיר כלים ייחודים שיש ב#c, ונציג את הבסיס לפיתוח משחקי מחשב במערכות ווינדוס.

1.2 סביבת עבודה – סביבת העבודה שנשתמש בה במהלך הקורס היא Visual Studio, הורידו אותה למחשב, היא חינמית. פרויקטים גדולים לעולם לא נעשים בצורה ידנית, ואם תנסו לעשות כך, צפויים לכם חיים קשים. באופן כללי, סביבת עבודה זו היא המומלצת ביותר על אף היותה תוכנה גדולה.

1.3 מעט על ' NET.' - לפני שנעסוק בהבדלים המינוריים בין # C לבין java, יעזור לנו לדעת מעט מה זה 'NET.' , ומה הקשר לסי שארפ. בניגוד לשם המטעה, ל-'.NET' אין שום קשר לאינטרנט, זהו סה"כ שם שיווקי שהתאים לרוח התקופה.
קשה להגדיר בדיוק מה אומר 'NET.' , היות וזה ביטוי כללי לאוסף הטכנולוגיות של מייקרוסופט הקשורות לכלי הפיתוח שלהם.
אבל היקף הביטוי בימים אלו עוסק בעיקר בניהול מערכת קוד המבוססת 'NET.'

מה שיפה ב"NET." שכאשר קוד נכתב בשפה המבוססת 'NET.' (במו # C, או vb.net) הוא מקומפל לכדי רמה של פורמט אחיד בלתי תלוי בשפה ממנה הוא בא, או אז הוא יכול להתקמפל ל- CPU ולרוץ.
זה אומר גם שניתן להשתמש ברכיבים משפות שונות לאותה תוכנית.

יש עוד מלא מידע טכני אודות 'NET.' ,ואם תבקשו מ 20 מתכנתים שונים להגדיר מה זה 'NET.' צפוי שתקבלו 20 תשובות שונות לשאלה ,אך חשוב מכל כעת זה ללמוד #c.

2 מבנה של פרויקט ב- c#:

ב java ניתן להשתמש בערימה של קבצי 'java.' ואז באמצעות הפקודה javac מאגדים את הקבצים לכדי קובץ, 'class.' אחד.
ב- c# המערכת די זהה, ניתן להריץ csc ידנית דרך ה CMD, אבל אף אחד לא באמת עושה את זה.
במקום יש *קבצי solution (.sln)*- זה למעשה רק רשימה של קבצי הפרויקט שמאוגדים יחד.
קבצי project (.cspros)- די זהה ל- java package. קבצים אלו יתקמפלו לקובצי exe או dll . אוגד בתוכו מספר קבצי 'cs.' .
msbuild –מגיע עם רוב המכונות המודרניות המבוססות ווינדוס. יכול להצביע על קובץ solution או project וידע איך לקמפל.
קבצי קוד (.cs)- קבצי הקוד של התוכנית, זהה לקבצי 'java.' , בניגוד ל java השם לא משפיע על המחלקה בקובץ, למרות שמומלץ כן להתאים בין שם הקובץ למחלקה שהוא מייצג.

2.2 מרחב שם (namespace)- בסי שארפ, בדומה ל ++c, משתמשים במרחבי שם.
מרחבי השם(שדומים לשמות ה- packages ב- java) יכולים להיות מה שרוצים בלי קשר למיקום הקובץ.
בדומה ל ++c ניתן להגדיר את מרחב השם בראש הדף ובכך לחסוך בכתיבה מיותר:

: C++

```
using std::cout;
using std::endl;
. . .
```

: C#

```
System.Console.WriteLine("Hello World!");
int i = System.Convert.ToInt32("123");
```

ובשימוש ב- namespace:

```
using System;
. . .
Console.WriteLine("Hello World!");
int i = Convert.ToInt32("123");
```

2.3 Program.cs – בכל תוכנית, קובץ ה exe קורא למתודה main של התוכנית הקובץ Program.cs.
הערה: ניתן להגדיר דרך ה Visual Studio איזה קובץ רוצים להגדיר כקובץ הראשי שממנו תפעל התוכנית, במידה והשתמשנו בכמה קבצי project באותה תוכנית.
ב- Visual studio:

To set a single startup project

1. In Solution Explorer, select the desired startup project within your solution.
2. On the Project menu, choose Set as StartUp Project.

2.3 מוסכמות חברתיות בנוגע לכתיבת קוד ב- #c - אלו רק מוסכמות, אין חובה לכבד אותן אם מתעקשים לא לעשות כך.
אבל אם אתם מתכננים לעבוד עם אנשים אחרים שאמורים לקרוא את הקוד שלכם, מומלץ ביותר להקפיד על המוסכמות הבאות:

* מחלקות מתחילות באותיות גדולות- בדיוק כמו ב- java .
* בניגוד לjava מתודות מתחילות גם באותיות גדולות (uppercase).
* ממשקים מתחילים עם האותיות ' IE ' או רק ' I ' (uppercase).
* סוגריים מסולסלות מתחילות מהשורה מתחת.

* משתנים מוגדרים ב- camelCase כלומר כל מילה חדשה בשם של המשתנה, נכתוב בצמוד למילה הקודמת, ומה שיפריד בין המילים יהיה האות הגדולה של תחילת המילה החדשה. שלא כמו ב ++c ששם המשתנים מוגדרים כ- road_kill_case (למשל int flagCounter).
* בניגוד ל- java המשתנים של ה- enums מוגדרים כ- PascalCase (כמו camelCase רק שגם המילה הראשונה מתחילה באות גדולה) , ולא Upper ROAD_KILL_CASE (כמו road_kill_case רק שהכל באותיות גדולות).

3 הבדלי טרמינולוגיה:

ישנם מספר מילים שמורות שזהות בין java ל #c במהותן אך עם שם שונה:
* ב- java למשתנים המוגדרים כ- package friendly (כלומר אלה שלא הוכרזו כ- private, public או protected) יש מילה שמורה ב #c- internal . משתנים, שדות וכדו' המוגדרים internal ניתנים לשימוש בקודים מאותו הפרויקט בלבד.
* המילה השמורה final ב java היא יוצאת דופן- אפשר להגדיר איתה מחלקות שלא ניתן לרשת מהן, או ניתן להגדיר שדות או משתנים שלא ניתן לשנות את ערכם. ב- #cלעומת זאת אם נרצה להגדיר מחלקה שלא ניתן לרשת ממנה נשתמש במילה השמורה sealed, ועבור שדות שלא נרצה שישנו את ערכם יש את המילה השמורה readonly או const.
הערה: ל const ו- readonly יש הבדל קטן בעיקר בהגדרה של המשתנה בזמן קומפילציה: משנה קבוע (const) חייב להיות מוגדר בזמן קומפילציה, לכן לא ניתן להגדיר את המשתנה דרך מתודה , ומשנה readonly הוא יותר דינאמי, הוא כמו reference למקום בזיכרון.
* for each – בשפות ממשפחת c, כאשר השתמשנו בלולאת for each השתמשנו בסינטקס הבא:

```
for (type var : array)
{
    statements using var;
}
```

לעומת זאת בסי שארפ מכריזים על הלולאה בצורה שיותר דומה לסינטקס של פייתון:

```
foreach (type var in array)
{
    statements using var;
}
```

4 הבדלי טיפוסים:

ב- #c הטיפוסים הנחשבים כ-'פרימיטיביים' זהים לטיפוסים הפרימיטיביים של ++c : int,double, bool, long, short,uint ...
למעשה כל הטיפוסים הפרימיטיביים של סי שארפ הם אובייקטים היורשים ממחלקת valuetype שהדבר המייחד אותם שכאשר מייצרים אובייקט מסוג valuetype יוצרים **אינסטנס** של האובייקט ולא **רפרנס** שלו כמו בשאר האובייקטים (שיורשים ממחלקות) , ובכך הם יושבים על המחסנית בזיכרון ולא על הערימה הדינמית. עצם היות המשתנים הפרימיטיביים 'אובייקטים' מאפשרת שימוש במתודות של המשתנה מבלי לקרוא למחלקה מיוחדת המיועדת לכך (למשל מחלקת Integer בjava).
כמו כן, לא ניתן לתת ערך null למשתנים היורשים valuetype .
הערה: ניתן ליצור משתנה מסוג value type (כלומר נשמר במחסנית ולא בערימה הדינמית) ע"י יצירת structs ונדבר על כך בהמשך.

4.2 אוספים- למעשה להרבה אובייקטים בסי שארפ היורשים ממחלקת collection יש סינטקס שונה משל של אלה של java, בין מבחינת השם של האובייקט או המתודות שלו, ובין מבחינת תוכן המתודות.

לרוב כאשר מתעסקים עם ממשקים תופיע האות 'I' שמסמלת אינטרפייס, כך למשל הממשק List ב-java זהה לממשק IList של סי שארפ. מערך רב ממדי הוא דוגמא לאובייקט יוצא דופן מבחינת סינטקס: כאשר רצינו להגדיר מערך דו ממדי חדש ב java הגדרנו אותו כך :

```
dataType[][] arr= new datatype[SIZE][SIZE];
```

בסי שארפ לעומת זאת אין שימוש בשני זוגות סוגריים אלא בזוג אחד עם פסיק באמצע שמפריד בין הממדים (העמודות והשורות למשל).

```
dataType[,] arr= new dataType[SIZE,SIZE];
```

בנוסף בכדי להשיג את אורך העמודות או השורות ב java :

```
int row= arr.length;
int col= arr[0].length;
```

ובסי שארפ:

```
int row = arr.GetLength(0);
int col = arr.GetLength(1);
```

לפעמים השוני הוא רק אם האובייקט מתחיל באות קטנה או באות גדולה, למשל String ב- java נקרא string בסי שארפ.

4.3 המילה שמורה var – המילה השמורה var יכולה להחליף טיפוסים בזמן הכרזה על האובייקט, ובכך מאפשרת לקומפיילר להסיק על סוג המשתנה בזמן קומפילציה. דומה מאוד למשתנה auto ב- c++, רק שלא ניתן להחזיר var מפונקציה.
למשל השורות הבאות:

```
int num = 123;
string str = "asdf";
Dictionary<int, string> dict = new Dictionary<int, string>();
```

זהות ל:

```
var num = 123;
var str = "asdf";
var dict = new Dictionary<int, string>();
```

5 הגדרת מחלקות ומבנים:

כשאנחנו מגדירים מחלקות בjava אנחנו צריכים לחלק בין מחלקות שאנחנו יורשים מהן עם המילה השמורה extends, וממשקים עם המילה implements ופסיק שמפריד בין כל ממשק שאנחנו יורשים ממנו (במידה ויש כמה).
ב- #c, בדומה ל++c, אנחנו פשוט שמים נקודתיים לאחר שם המחלקה שאותה יוצרים, ומפרידים בפסיק בין שמות המחלקה או הממשקים שמהם יורשים, והיות וכל הממשקים בסי שארפ מחילים באו גדולה 'I' (כמוסכמה) קל להבדיל בין ממשק למחלקה, עוד סיבה למה חשוב לשמור על המוסכמות החברתיות.
בדומה ל-java גם כאן ניתן לרשת ממחלקה אחת (בלבד) ומכמה ממשקים.

```
public sealed class MyThing : AbstractMyThing, IThing
{
  private readonly int value;
  public MyThing(int value)
  {
    this.value = value;
  }
}
```

5.2 מבנים (structs)- מבנה הוא טיפוס מסוג value type שדומה במהותו למחלקה, רק שניתן להשתמש בו גם מבלי לאתחל אותו עם אופרטור new, ובכך האובייקט לא ישמר בערימה הדינאמית ויחסוך מקום בזיכרון.
תכונות של מבנה: * מבנה ושדותיו יחשבו כ- internal כברירת מחדל, אלא אם יוגדרו אחרת (public, private או protected).
*ניתן גם ליצור אובייקט מסוג מבנה חדש ע"י אופרטור new, במקרה כזה יקרא הבנאי המתאים לכך.
*במידה ויוצרים מבנה מבלי להשתמש ב-new, לא נקרא שום בנאי ולכן לא מאותחלים השדות של המבנה, אי לכך נצטרך לאתחל בצורה ידנית את המשתנים של המבנה, ואם נשתמש באחד מהשדות שלא אתחלנו אותו תיזרק שגיאת קומפילציה.
*מבנה לא יכול להכיל בנאי ריק- הוא יכול להכיל רק בנאי עם פרמטרים, או בנאי סטאטי. הבנאי צריך להכיל את כל השדות של המבנה כפרמטרים.
*לא ניתן לרשת ממבנה, ומבנה לא יכול לרשת מאחרים.

```
public struct Test
    {
         public int i ;
         public int j;
         public  Test(int _i, int _j)
        {
            i = _i;
            j = _j;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Test t;
            t.i = 0;
            t.j = 0;
            Test t2 = new Test(1, 1);
        }
    }
```

לסיכום: מבנה הוא טיפוס value type ולכן הוא מהיר יותר מאובייקט מסוג מחלקה. בכללי מבנים טובים לתוכניות של משחקים, אך במקרה של העברת מידע בין מחלקות או שימוש דינמי במידע עדיף להשתמש במחלקה או משתנים פרימיטיביים.

5.3 setters and getters- ב #c אין צורך להגדיר מתודות מיוחדות של set ו- get למשתנה מסוים, למעשה יש סינטקס שמקל עלינו להגדיר ולהשתמש במתודות
הנה דוגמת קוד בjava-

```
public class Foo {
  private int value;
  public Foo(int value) {
    this.value = value;
  }

  public int getValue() {
    return value;
  }

  // package-scoped setter
  void setValue(int value) {
    this.value = value;
  }
}
```

וכך אותו קוד יראה ב- #c:

```
class Foo
{
  public int Value { get; internal set; } // project-scoped setter

  public Foo(int value)
  {
    this.Value = value;
  }
}
. . . (main)
Foo foo = new Foo(42);
foo.Value = -1;
int newValue = foo.Value;
```

ניתן אפילו לעקוף את הבנאי של המחלקה ככה: (הגדרנו ישירות את משתנה המחלקה בלי בנאי)

```
class Foo
{
  public int Value { get; internal set; }
}

Foo foo = new Foo() { Value = 42 };
```

אבל רגע! היופי ב setter ו- getter של java שניתן להוסיף להן קוד שירוץ עם הקריאה למתודה.
ניתן לעשות זאת גם בסי שארפ! set; ו- get; כברירת מחדל יתנהגו כמו שדות אך ניתן לתת להם יישום מיוחד:

```
public class Foo
{
  private int value;
  public int Value
  {
    get
    {
      return this.value;
    }

   set
    {
      // the word "value" here that you're setting to the field
      // "this.value" is actually a C# keyword available in
      // all setters.
      this.value = value;
    }
  }

  public bool IsEven { get { return this.value % 2 == 0; } }
}
```

5.4 אופרטורים - כמו ב c++ גם בסי שארפ ניתן להגדיר אופרטורים למחלקה.
בסי שארפ כל האופרטורים מוגדרים כסטאטיים

```
public struct Test
    {
         public int i ;
         public int j;
         public  Test(int _i, int _j)
        {
            i = _i;
            j = _j;
        }
        public static Test operator +(Test t1,Test t2)
        {
            Test t3;
            t3.i = t1.i + t2.i;
            t3.j = t1.j + t2.j;
            return t3;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Test t;
            t.i = 0;
            t.j = 1;
            Test t2 = new Test(1, 1);
            Test t3 = t + t2;
            Console.WriteLine(t3.i + "," + t3.j);
        }
    }
```

5.5 מתודות וירטואליות- בסי שארפ, כמו ב++c ושלא כמו ב-java, בכדי שמחלקה יורשת תדרוס מתודה של מחלקת האב, על המתודה להיות מוגדרת כמתודה וירטואלית. זאת על מנת לחסוך בזיכרון בזמן יצרת טבלאות וירטואליות עבור כל אובייקט.
במתודה של האב יש לציין virtual ליד ערך החזרה (או ליד void במידה ואין ערך חזרה) ואצל הבן יש לציין override באותה מתודה, אחרת לא תהיה דריסה של המתודה ממחלקת האב.

```
1.  class Shape
2.     {
3.        public double length=0.0;
4.        public double width =0.0;
5.        public double radius =0.0;
6.
7.        public Shape(double length, double width)
8.        {
9.            this.length = length;
10.           this.width = width;
11.       }
12.
13.       public Shape(double radius)
14.       {
15.           this.radius = radius;
16.       }
17.
18.       public  virtual void Area()
19.       {
20.           double area = 0.0;
21.           area = Math.PI * Math.Pow(radius, 2);
22.           Console.WriteLine("Area of Shape is "+ area);
23.       }
24.    }
25.
26.    class Rectangle  : Shape
27.    {
28.
29.        public Rectangle(double length, double width): base(length, width)//super
    כמו
30.        {
31.        }
32.
33.        public override void Area()
34.        {
35.            double area = 0.0;
36.            area = length * width;
37.            Console.WriteLine("Area of Rectangle is "+ area);
38.        }
39.    }
```

5.6 מחלקות פנימיות - המילה 'static' באופן כללי מתייחסת למשנה או מתודה השייכים למשהו כללי של סוג המחלקה(כלומר שדה שמשותף לכל האובייקטים מאותו הסוג). בעוד שמשתנה או מתודה שמוגדרים כלא סטאטיים שייכים לאובייקט ספציפי ולא לכלל האובייקטים מאותו סוג.
הדבר נכון ל- java ול- #c, אולם כאשר מדברים על מחלקות המילה השמורה static עובדת בצורה שונה לחלוטין.
ב- java מחלקה מקוננת (מחלקה בתוך מחלקה) שהיא סטאטית אז המחלקה מוגדר כשייכת למחלקה העוטפת אותה, כלומר צריך להשתמש בשם המחלקה העוטפת בכדי לקרוא למחלקה הפנימית: outerClass.innerClass , ומחלקה שאינה מוגדרת כסטאטית נדירה מאוד ולרוב מיוחסת לטעות של המתכנת.
ב- #c לעומת זאת כל המחלקות המקוננות מוגדרות (כברירת מחדל) כסטאטיות במובן הjava-אי של המילה, מבלי שצריך להכריז עליהן static בשם המחלקה במיוחד.

```
// Java
public class Foo {
  public static class Bar {
    ...
  }
}
// C#
public class Foo
{
  public class Bar
  {
    ...
  }
}
// Code for both C# and Java to instantiate:
Foo foo = new Foo();
Bar bar = new Foo.Bar();
```

6 פונקציות קונסול:

כשאנחנו מתכנתים ב-java אנחנו משתמשים בפונקציות של האובייקט System כדי לתקשר עם הקונסול (console) , ובפרט באובייקטים של system כדי לטפל בקלטים ופלטים, למשל כדי להדפיס למסך השתמשנו בפונקציה:

```
System.out.print("…");
System.out.println("…");
```

או כדי לקבל קלט או היו צריכים לבצע את הפרוצדורה הבאה:

```
import java.util.Scanner;  // Import the Scanner class


class MyClass {

  public static void main(String[] args) {

    Scanner myObj = new Scanner(System.in);  // Create a Scanner object

    System.out.println("Enter username");
```

```
    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

כאשר אנחנו רוצים לתקשר עם הקונסול ב-#C נשתמש גם כן במרחב השם System אך נשתמש באובייקט Console של System (System.Console).
מחלקת Console היא למעשה מחלקה שנועדה כדי לטפל בקלטים ופלטים סטנדרטים מהקונסול, ובשגיאות למיניהם. לא ניתן לרשת מהמחלקה.
פונקציות שימושיות שיש למחלקה הן:

```
Console.WriteLine();//משמש להדפסה וירידת שורה
```

```
Console.Write();//משמש להדפס אך בלי ירידת שורה
```

```
Console.WindowHeight;  //שדה שמשמש להגדרת גובה ההמסך של הקונסול
Console.WindowWidth;  //שדה המשמש להגדרת רוחב החלון
Console.WindowWidth = 80;
Console.WindowHeight=55;
```

לפעמים נרצה גם לקבל קלט או לגלות איזה מקש ספציפי לחץ עליו המשתמש. האובייקט key מחזיר לנו אותו:

```
Console.ReadKey(bool).Key;
```

דוגמא:

```
private static void get_key()
        {
            var ch = Console.ReadKey(false).Key;
            switch (ch)
            {
                case ConsoleKey.UpArrow:
                    set_player_movment(-1, 0);
                    return;
                case ConsoleKey.DownArrow:
                    set_player_movment(1, 0);
                    return;
                case ConsoleKey.RightArrow:
                    set_player_movment(0, 1);
                    return;
```

```
                case ConsoleKey.LeftArrow:
                    set_player_movment(0, -1);
                    return;
                case ConsoleKey.Enter:
                    print_solution();
                    return;
            }
        }
```

קבלה של קלט מהקונסול:

```
Console.ReadLine();
Console.Read();//קריאה של תו אחד
```

צביעה של הטקסט:

```
Console.ForegroundColor;

Console.ForegroundColor = ConsoleColor.DarkYellow;// צובע את התו הבא בצבע צהוב
Console.Write("☺ ");
Console.ForegroundColor = ConsoleColor.White;// חוזרים לצבע (לבן) המקורי כדי שרק התו הספציפי יצבע
```

לנקות את המסך(כמו clear במערכת לינוקס או הפקודה cls ווינדוס):

```
Console.Clear();
```

הערה: לכל אחת מהפונקציות שהצגנו למעלה צריך להוסיף את הקידומת system אלא אם הגדרנו אותו במרחב השם (כפי שראינו ב- 2.2)
לרשימה המלאה ועוד על מחלקת קונסול:

https://docs.microsoft.com/en-us/dotnet/api/system.console?view=netframework-4.8