בס"ד

Generics מאפשרת לכתוב מחלקות שיש בהם שימוש חוזר בסוגים שונים של משתנים, בלי להשתמש בהמרות, בלי Boxing ו Unboxing ועם Type safety.
ב Generics אנחנו יוצרים מעין תבנית למתודה או מחלקה.
בהגדרת המחלקה הגנרית נעביר עם תחביר מיוחד - בתוך סוגריים משולשות <> את הסוג של הפרמטר, וכך בהמשך נוכל לממש וליצור מופעים של המחלקה או להשתמש במתודה עם סוגים שונים של משתנים.
נראה דוגמה:

```
// Declare the generic class.
public class GenericList<T>
{
    public void Add(T input) { }
}
class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // Declare a list of type int.
        GenericList<int> list1 = new GenericList<int>();
        list1.Add(1);

        // Declare a list of type string.
        GenericList<string> list2 = new GenericList<string>();
        list2.Add("");

        // Declare a list of type ExampleClass.
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
        list3.Add(new ExampleClass());
    }
}
```

כאשר יממשו את המחלקה החדשה שיצרנו, הקומפיילר יבדוק איזה Type העברנו, ויבנה מחלקה זהה, כאשר הוא מחליף כל מקום שכתבנו T לסוג שמימשנו.
בדוגמה הקודמת, הקומפיילר יצור 3 סוגים של מחלקות ויעשה בהם שימוש כל אחד במקומו.
הקומפיילר יצור מחלקה GenericLis~Int, GenericList~String ו GenericList~ExampleClass.

הקומפיילר יצר מחלקה שנראית כך:

```
public class GenericList~Int
{
    public void Add(int input) { }
}
```

בדוגמה הנ"ל: מיד אחר כך הקומפיילר יצור **מופע** של GenericLis~Int ויצביע עליו עם המשתנה שנקרא list1.

אם ננסה לכתוב `list1.Add("");` נקבל שגיאת קומפילציה, כי המתודה Add של GenericList~Int כמובן מצפה לקבל int.

השמות שהצגתי כשמות המחלקות שהקומפיילר יוצר, הם לא השמות המדויקים, ונועדו להמחשת העניין בלבד!

מחלקות ושיטות גנריות משלבות שימוש חוזר, בטיחות סוג (type safety) ויעילות באופן שמקבילותיו הלא גנריות לא יכולות .

לרוב, משתמשים ב Generics עם אוספים ומתודות שפועלות עליהם .

אם כך, Generics אפשרה לנו לעצב מחלקות ומתודות שדוחות את הגדרת ה Type של הפרמטר עד שישתמשו בו וימשו אותו בקוד, כלומר עד שיצרו מופע שלו.

יש המון מחלקות גנריות שמובנות בדטנט ,

ה namespace שנקרא System.Collections.Generic מכיל מספר מחלקות שמטפלות באוספים מבוססות גנריות .

האוספים הלא גנריים, כגון ArrayList אינם מומלצים ומתוחזקים לצורכי תאימות .

כמו בדוגמה למעלה, תוכל גם אתה ליצור סוגים ושיטות כלליות בהתאמה אישית כדי לספק פתרונות כלליים ודפוסי עיצוב משלך, בטוחים ויעילים.

לטיפול באוספים, ברוב המקרים, עליך להשתמש במחלקת List <T> המסופקת על ידי NET . במקום ליצור משלך (למרות שזה מגניב...)

שימו לב שפרמטר T משמש במספר מיקומים שבהם בדרך כלל יש להגדיר במפורש את הסוג. הוא משמש מזהה לסוג של ערך מוחסר, מזהה לסוד של פרמטר של מתודה, כסוג של שדה. נראה את זה בדוגמה הבאה, שממשת רשימה מקושרת פשוטה.

```
// type parameter T in angle brackets
public class GenericList<T>
{
    // The nested class is also generic on T.
    private class Node
    {
        // T used in non-generic constructor.
        public Node(T t)
        {
            next = null;
```

```
            data = t;
        }

        private Node next;
        public Node Next
        {
            get { return next; }
            set { next = value; }
        }

        // T as private member data type.
        private T data;

        // T as return type of property.
        public T Data
        {
            get { return data; }
            set { data = value; }
        }
    }

    private Node head;

    // constructor
    public GenericList()
    {
        head = null;
    }

    // T as method parameter type:
    public void AddHead(T t)
    {
        Node n = new Node(t);
        n.Next = head;
        head = n;
    }

    public IEnumerator<T> GetEnumerator()
    {
        Node current = head;

        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
        }
    }
}
```

T מופיע בדוגמה למעלה

•כסוג של פרמטר שיטה בשיטת.AddHead

•כסוג ההחזרה של מאפיין הנתונים במחלקת.Node

•כסוג ה private member במחלקה המקוננת.

סיכום

• השתמש ב Generics על מנת למקסם את השימוש החוזר בקוד, בטיחות הקוד והביצועים.

• השימוש הנפוץ ביותר בגנריקה הוא יצירת מחלקות לטיפול באוספים.

• ספריית מחלקות .NET מכילה מספר כיתות אוסף גנריות במרחב השמות System.Collections.Generic. יש להשתמש באוספים הגנריים בכל מקום אפשרי במקום בישנות כגון ArrayList שנמצא במרחב השמות System.Collections.

• תוכל ליצור ממשקים, מחלקות, שיטות, אירועים ו members עם גנריקה משלך.

• מחלקות גנריות עשויות להיות מוגבלות כדי לאפשר גישה לשיטות על סוגי נתונים מסוימים.

• ניתן לקבל מידע על הסוגים המשמשים בסוג נתונים גנרי בזמן ריצה באמצעות Reflection.

Constraints – הגבלות

אילוצים מציינים את הציפיות שלנו מה Type parameter. הצהרה על אילוץ מוודאת שישתמשו במתודות ובמחלקות הגנריות שכתבת רק באופן שתכנת, ומאפשרת להשתמש במתודות ומאפיינים ששיכים לסוג שקיבלנו כפרמטר בתוך המתודה או בתוך המחלקה.
כמו תמיד, בכדי להבין נעבור מיד לדוגמה.

מקטע זה יושלם בגרסה הבאה של המסמך (שיוגש בעז"ה לפני השיעור הבא)

תרגילי כיתה ובית

1. כתבו פונקציה Combine שמחברת מערכים שהתקבלו כפרמטר מכל סוג.
2. ממשו אוסף Read only גנרי שמבוסס על מערך, למחלקה יהיה מתג (סוויטץ) – משתנה, שיגדיר אם ניתן כרגע להוסיף איברים למערך. זרוק InvalidOperationException אם ניסו להוסיף ערך בזמן ש IsReadOny = true.
3. צרו אוסף שבו האיברים תמיד ממוינים, האוסף יעבוד על char, string, int, double לא על Classes.
4. צרו מחלקה גנרית שיודעת ליצור מחלקות חדשות שיש להם Id ולאתחל בהם את ה ID. באיזה Constraints נשתמש?

5. כתבו Stack גנרי (מתודות: Push, Pop, Peek)
6. כתבו Queue גנרי (מתודות: Enqueue, Dequeue)
7. ממשו מחלקה שמדפיסה את הסוג של המחלקה ואת המאפיין Name שבמחלקה. איך נוודא שיש מאפיין בשם Name במחלקה?