بنام کسی که آفرید از عدم به انسان عطا کرد فکر و قلم

Array

آرایه ها در (c#.teN) تعدادی خانه حافظه هم شکل هستند که برای موارد کاملا مشخص بکار می روند . بطور مثال فرض کنید یک تعداد مشخص عدد دارید که اعداد همه از یک نوع(data type) هستند , یک راه آن است که به تعداد مورد نظر متغیر ایجاد شود و مقدار هر متغیر در آن ذخیره شود که این کار وقتی تعداد متغیرهای از یک شکل زیاد می شود , مدیریت را سخت می کند , برای حل این مشکل از آرایه استفاده می شود , آرایه ها تعدادی خانه حافظه هستند که همه آنها از یک نوع هستند مثلا همه از نوع رشته و یا همه از نوع عدد . به مثال زیر دقت کنید :

```
int a1,a2,a3,a4,a5,a6,a7,a8,a9;
```

متغیر هایی از یک نوع با نام های متفاوت

```
int[] a = new int[9];
```

یک آرایه از نوع عددی با ۹ عنصر

```
int[] b = new int[5]{12,4,6,7,90};
```

یک آرایه عددی با طول ۵ عنصر و مقداردهی اولیه

آرایه ها نسبت به متغیرهای عادی تفاوتهای دارند، برای مثال فرض کنید که از وجود یک مقدار خاص در آرایه مطلع شوید ، برای این کار کافیست یکی از متدهای از پیش تعریف شده در آرایه را صدا کنید . این به این معنی است که (framework) برای آرایه ها یک سری متد نوشته که کار با آنها را سریع تر کند . البته کارآمدتر شود . همچنین آرایه ها از نظر مصرف حافظه بهینه ترین حالت ممکن را دارند . آرایه ها می توانند یک – دو – سه و یا چند بعدی باشند که این قابلیت در محاسبات بسیار پرکاربرد است .

آرایه ها به دو دسته تقسیم می شوند . یکی آرایه های معمولی که تعریف آنها در بالا آمد و دیگری Jagged Array , **جگد اری ها** , آرایه هایی (حداقل) دوبعدی هستند که طول بعدهای آن برابر نیست برای مثال :

```
int[,] simple = new int[2,3]{{1,2,4},{4,45,2})
```

یک آرایه دو بعدی معمولی با مقدار

```
int[][] jagged = new int[3][];

jagged = new int{12,3,6,17,3,79,23,};

jagged = new int{6,89,34};

jagged = new int{54,25};
```

در مثال بالا یک آرایه دو بعدی ساده را دیدید که دارای دو ردیف و سه ستون بود ،یعنی لزوما باید برای هر ردیف سطر عنصر تعریف شود . در مثال جگد اری , یک آرایه با سه ردیف و تعداد نامشخصی ستون تعریف شده که در ردیف اول ۷ عنصر و در ردیف دوم سه عنصر و در ردیف سوم دو عنصر قرار دارد . مزیت جگد اری نسبت به آرایه معمولی این است که تعداد یکی از بعدها می تواند نامشخص باشد .(تعریف دقیق تر)

Jagged arrays

Jagged arraysآرایه ای از آرایه ها است و همانطور که ذکر شد لزومی ندارد که هر ردیف آن با ردیف بعدی هم طول باشد . هنگام تعریف این نوع آرایه شما تعداد ردیف ها را مشخص می نمایید .هر ردیف یک آرایه را نگهداری می کند. در اینجا هر آرایه باید تعریف شود .روش تعریف Jagged array به صورت زیر است:

```
Data type [] []...
```

در اینجا تعداد براکت ها بیانگر ابعاد آرایه می باشد. برای مثال آرایه ی زیر دو بعدی است :

```
int [] [] myJaggedArray;
```

و برای مثال برای دسترسی به پنجمین عنصر از آرایه ی سوم به صورت زیر عمل می شود :

```
myJaggedArray[2][4];
```

مثال : برنامه ای که یک آرایه ی jagged Array را مقدار دهی می کند

```
static void Main(string[] args)
        {
            const int rows = 4;
            // declare the jagged array as 4 rows high
            int[][] jaggedArray = new int[rows][];
            // the first row has 5 elements
            jaggedArray[0] = new int[5];
            // a row with 2 elements
            jaggedArray[1] = new int[2];
            // a row with 3 elements
            jaggedArray[2] = new int[3];
            // the last row has 5 elements
            jaggedArray[3] = new int[5];
            // Fill some (but not all) elements of the rows
            jaggedArray[0][3] = 15;
            jaggedArray[1][1] = 12;
            jaggedArray[2][1] = 9;
            jaggedArray[2][2] = 99;
            jaggedArray[3][0] = 10;
            jaggedArray[3][1] = 11;
            jaggedArray[3][2] = 12;
            jaggedArray[3][3] = 13;
            jaggedArray[3][4] = 14;
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("jaggedArray[0][{0}] = {1}", i,
jaggedArray[0][i]);
            }
            for (int i = 0; i < 2; i++)
            {
                Console.WriteLine("jaggedArray[1][{0}] = {1}", i,
jaggedArray[1][i]);
            }
            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine("jaggedArray[2][{0}] = {1}", i,
jaggedArray[2][i]);
            }
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("jaggedArray[3][{0}] = {1}", i,
jaggedArray[3][i]);
            }
            Console.ReadLine();
        }
```

Collection

کالکشن ها که در دات نت یک معرفی شدند یکسری آرایه بودند که نوع نداشتند و درواقع نوع آنها object بود , برای مثال شما می خواهید یک لیست داشته باشید که بعدا یکسری عدد در آن بریزید , اما هنوز از نوع عدد (int , Double , Decimal ...) اطلاعی ندارید , از آنجا که کلکشن ها آبجکت می گیرند، شما می توانید بدون دردسر کد مربوطه را بنویسید . این قابلیت باعث همه گیر شدن کالکشن ها شد و کالکشن های پیش فرض و پرکاربردی را مایکروسافت ارائه کرد برای مثال Stack و Queue و یا ArrayList و غیره و یا حتی یکسری کالکشن ویژه که برای کاربردهای خاص به کار می آمدند مثل HashTable .

این کالکشن ها بخاطر کاربردی بودن و قابلیت گرفتن آبجکت (آبجکت به این دلیل که معلوم نبود کاربر چه چیزی می خواهد مثلا در Queue بگذارد) سریعا در بین برنامه نویس ها پذیرفته شد و رشد کرد . اما یک مشکل بزرگ وجود داشت .

```
System.Collections.Queue person = new Queue(); //
```

یک کالکشن از نوع صف با نام مشخص

همانطور که گفته شد کالکشن ها آبجکت می گرفتند و این باعث بروز مشکلاتی شد , برنامه نویس ها نمی دانستند که سایر اعضا چه نوع عددی

در کالکشن ریخته اند , و هنگام Cast کردن مشکلات ادیده ای پیش آمد . مثلا برنامه نویس یک , یک کالکشن ایجاد کرده و یکسری double داخل آن ریخته , حالا برنامه نویس دو , صرفا از عددی بودن این کالکشن خبر داشت و هنگام Cast کردن آن به int به مشکل می خورد . این مشکلات باعث بوجود آمدن Generic شد.

Generic

جنریک که در دات نت دو , معرفی شد یک کالکشن بود که بجای آبجکت، یک نوع(epyt etad) می گرفت , برای مثال

```
Ssystem.Collections.Generic.Queue<int>
myTypedcollection = new Queue<int>();
```

در مثال بالا یک جنریک با نوع int ایجاد شده است , مزیت استفاده از جنریک , مشخص بودن نوع آن برای همه است , مهم نیست که برنامه نویس دو , کنار برنامه نویس یک باشد تا از نوع عددی کالکشن باخبر باشد , جنریک مشخصا اعلام می کند که نوع آن چیست . به همه برنامه نویس ها توصیه می شود که از جنریک ها بجای کالکشن های قدیمی استفاده کنند . تقریبا بجای همه کالکشن های دات نت یک , معادلی در جنریک ها موجود است . برای مثال

کالکشن ArrayList به جنریک <> List تبدیل شده و یا Queue , Stack مشخصا با همین نام Stack<> و Queue<> در دات نت دو

معرفی شده اند . بعضی از جنریک ها کاملا جدید هستند و تقریبا برای هر کار پرکاربردی یک جنریک وجود دارد .

حافظه Stack: حافظه در داخل برنامه ها به دو بخش تقسیم می شود: حافظه Stackو حافظه Heap .حافظه Stack، برای نگهداری متغیر های معمولی و نگهداری اطلاعات توابع در هنگام فراخوانی استفاده می شود. به طور کلی می توان گفت مهمترین کاربرد حافظه Stack در فراخوانی توابع و کنترل آنها می باشد. برای این که با طرز کار حافظه Stackبیشترآشنا شوید، می توان این بخش از حافظه را به ستونی از بشقاب های رو هم قرار گرفته تشبیه کرد. آخرین بشقابی که وارد شده، اول از همه از ستون بشقاب ها خارج می شود.

به این حالت به LIFO-Last In First Outمعروف است. این مسئله در مورد توابع نیز سازگار است. زمانی که تابعی فراخوانی می شود این تابع به همراه تمامی متغیرهای محلی خودش در داخل حافظه Stackقرار می گیرد. با فراخوانی یک تابع جدید این تابع بر روی تابع قبلی قرار میگیرد و کار به همین صورت ادامه پیدا می کند. در حقیقت می توان گفت که بالاترین تابع در حافظه Stackتابعی است که هم اکنون در حال اجرا می باشد. زمانی که کار فراخوانی یک تابع تمام شد آن تابع به همراه تمام متغیرهای مربوطه از داخل حافظه Stackخارج می شود.

حافظه Heap : حافظه Heap بخشی جداگانه از حافظه است که هیچ وابستگی به تابع های فراخوانی شده برنامه ندارد. متغیرها به صورت معمولی در این بخش قرار نمی گیرند و باید برنامه نویس به صورت دستی متغیرها رو داخل این بخش تعریف کند. به همین دلیل مدیریت حافظه Heap داخل برنامه باید به صورت دستی توسط برنامه نویس انجام شود. متغیرهای تعریف شده داخل حافظه Heap با اتمام فراخوانی یک تابع از بین نمی روند و تا زمانی که برنامه نویس خود این متغیر را از داخل حافظه Heap پاک نکند باقی خواهند ماند.