متدها

تاکنون تمامی اعمالی که ما در برنامه‌هایمان انجام می‌دادیم در متد Main() اتفاق می‌افتادند. این روش برای برنامه‌های ساده و ابتدایی که استفاده کردیم مناسب بود، اما اگر برنامه‌ها پیچیده‌تر شوند و تعداد کارهای مورد نظر ما گسترش یابد، استفاده از متدها جایگزین روش قبل می‌گردد. متدها فوق‌العاده مفید هستند، چراکه کارها را به بخشهای کوچکتر و مجزا تقیسم می‌کنند و در نتیجه استفاده از آنها آسان‌تر خواهد بود .ساختار کلی یك متد به صورت زیر است:

[attributes][modifiers] return-type method-name ([parameters]) { statements }

دو قسمت attributes و modifiers فعلا مورد بحث نیستند. return-type .نوعی است یك متد باز می‌گرداند و می‌تواند هر یك از انواع تعریف شده زبان C# و یا از انواع تعریف شده توسط کاربر باشد. هر متد با نام آن شناخته می‌شود method-name . نام انتخابی برنامه‌نویس برای یك متد است و از طریق همین نام فراخوانی متد انجام می‌شود. پارامترها (parameters) مولفه‌ها یا متغیرهایی هستند که برای انجام یکسری پردازش به متد ارسال می‌شوند و از طریق آنها می‌توان اطلاعاتی را به متد ارسال و یا از آن دریافت نمود، و در نهایت دستورالعمهای متد، دستورهایی از زبان C#هستند که بوسیله آنها عمل مورد نظر برنامه‌نویس انجام می‌شود و عملی است که یك متد آنرا انجام می‌دهد. مثال ۱-۵ پیاده‌سازی یك متد ساده را نمایش می‌دهد .

```
usingSystem;
classOneMethod
{
public static voidMain()
{
stringmyChoice;
OneMethod om = newOneMethod();
do
{
myChoice = om.getChoice();
//تصمیمی بر اساس انتخاب کاربر گرفته می‌شود
switch(myChoice)
{
case"A":
case"a":
Console.WriteLine("You wish to add an address.");
```

```
break;
case"D":
case"d":
Console.WriteLine("You wish to delete an address.");
break;
case"M":
case"m":
Console.WriteLine("You wish to modify an address.");
break;
case"V":
case"v":
Console.WriteLine("You wish to view the address list.");
break;
case"Q":
case"q":
Console.WriteLine("Bye.");
break;
default:
Console.WriteLine("{0} is not a valid choice", myChoice);
break;
}
//اجرای برنامه برای دیدن نتایج موقف می‌شود
Console.WriteLine();
Console.Write("Press Enter key to continue...");
Console.ReadLine();
Console.WriteLine();
} while(myChoice != "Q" && myChoice != "q");
//اجرای برنامه تا زمانیکه کاربر بخواهد ادامه می‌یابد
}
stringgetChoice()
{
stringmyChoice;
//منویی را نمایش می‌دهد
Console.WriteLine("My Address Book\n");
Console.WriteLine("A - Add New Address");
Console.WriteLine("D - Delete Address");
Console.WriteLine("M - Modify Address");
Console.WriteLine("V - View Addresses");
```

```
Console.WriteLine("Q - Quit\n");
Console.Write("Choice (A,D,M,V,or Q): ");
//ورودی دریافتی از کاربر را بررسی می‌کند
myChoice = Console.ReadLine();
Console.WriteLine();
return myChoice;
}
}
```

در این مثال، چاپ منو و دریافت ورودی از کاربر در یك متد مجزا بنام getChoice() صورت می‌گیرد. نوع بازگشتی این متد از نوع رشته‌ای است. از این رشته در دستور switch در متد Main() استفاده می‌شود. همانطور که ملاحظه می‌نمایید، پرانتزهای متد getChoice() خالی هستند، یعنی این متد دارای پارامتر نیست، از اینرو هیچ اطلاعاتی به/ از این متد منتقل نمی‌شود.

درون این متد، ابتدا متغیر myChoice را اعلان نموده‌ایم. هرچند نام و نوع این متغیر همانند متغیر myChoice موجود در متد Main() است، اما این دو متغیر دو متغیر کاملاً مجزا از یکدیگر می‌باشند. هر دو این متغیرها، متغیرهای محلی (Local) هستند، از اینرو تنها درون بلوکی که تعریف شده‌اند قابل دسترس می‌باشند. به بیان دیگر این دو متغیر از وجود یکدیگر اطلاعی ندارند.

متد getChoice() منویی را در کنسول نمایش می‌دهد و ورودی انتخابی کاربر را دریافت می‌نماید. دستور return داده‌ها را از طریق متغیر myChoice به متد فراخواننده آن، یعنی Main() ، باز می‌گرداند. توجه داشته باشید که، نوع متغیری که توسط دستور return باز گردانده می‌شود، باید دقیقاً همانند نوع بازگشتی متد باشد. در این مثال نوع بازگشتی، رشته است.

در C# دو گونه متد وجود دارد. یکی متدهای استاتیك (Static) و دیگری متدهای نمونه (Instance). متدهایی که در اعلان خود شامل کلمه کلیدی static هستند، از نوع استاتیك هستند، بدین معنا که هیچ نمونه‌ای از روی این متد قابل ایجاد نیست و این تنها همین نمونه موجود قابل استفاده است. از روی متدهای استاتیك نمی‌توان شیء (Object) ایجاد کرد. در صورتیکه در اعلان متد از کلمه کلیدی static استفاده نشده باشد، متد بعنوان متد نمونه در نظر گرفته می‌شود، بدین معنا که از روی آن می‌توان نمونه ایجاد کرد و شیء تولید نمود. هر یك از اشیاء ایجاد شده از روی این متدها، تمامی عناصر آن متد را دارای می‌باشند.

در این مثال، چون getChoice() بصورت استاتیك اعلان نشده است، پس باید برای استفاده از آن شیء جدیدی تولید شود. تولید شیء جدید بوسیله OneMethod om = new OneMethod() صورت می‌پذیرد. در سمت چپ این اعلان، مرجع این شیء جدید، یعنی om ، قرار دارد که از نوع OneMethod است. در اینجا توجه به یك نکته بسیار مهم است، om *به خودی خود شیء نیست، بلکه می‌تواند مرجعی به*

شیءای از نوع OneMethod() *را در خود نگه‌دارد* .در سمت راست این اعلان، تخصیص شیء جدیدی از نوع OneMethod() به متغیر om صورت گرفته است. کلمه کلیدی newعملگری است که شیء جدیدی را در heap ایجاد می‌نماید. اتفاقی که اینجا روی داده اینست که نمونه جدیدی از OneMethod() در heap تولید شده و سپس به مرجع om تخصیص داده می‌شود. حال که نمونه‌ای از متد OneMethod() را به om تخصیص داده‌ایم، از طریق om می‌توانیم با این متد کار نماییم .

متدها، فیلدها و سایر اعضای یك كلاس از طریق عملگر نقطه "." قابل دسترس هستند. هنگامیکه می‌خواهیم متد getChoice() را فراخوانی کنیم، بوسیله عملگر نقطه از طریق om به آن دسترسی پیدا می‌نماییم . om.getChoice() : برای نگهداری مقداری که getChoice() بر می‌گرداند، از عملگر "=" استفاده نموده‌ایم. رشته بازگشتی از متد getChoice() درون متغیر محلی myChoice متد Main() قرار می‌گیرد. از این قسمت، اجرای برنامه همانند قبل است.

پارامترهای متد

به مثال ۲-۵ توجه کنید.

```
usingSystem;
classAddress
{
public stringname;
public stringaddress;
پایان کلاس}//Address
classMethodParams
{
public static voidMain()
{
stringmyChoice;
MethodParams mp = newMethodParams();
do
{
//منویی نمایش داده شده و ورودی از کاربر دریافت می‌گردد
myChoice = mp.getChoice();
//تصمیمی بر اساس ورودی کاربر گرفته می‌شود
mp.makeDecision(myChoice);
//جهت دیدن نتایج توسط کاربر، اجرای برنامه موقتا موقف می‌گردد
Console.Write("Press Enter key to continue...");
Console.ReadLine();
Console.WriteLine();
} while(myChoice != "Q" && myChoice != "q");
//اجرای حلقه تا زمانیکه کاربر بخواهد ادامه پیدا می‌نماید
```

```
پایان متد}//Main
نمایش منو و دریافت ورودی از کاربر//
stringgetChoice()
{
stringmyChoice;
نمایش منو//
Console.WriteLine("My Address Book\n");
Console.WriteLine("A - Add New Address");
Console.WriteLine("D - Delete Address");
Console.WriteLine("M - Modify Address");
Console.WriteLine("V - View Addresses");
Console.WriteLine("Q - Quit\n");
Console.WriteLine("Choice (A,D,M,V,or Q): ");
دریافت ورودی کاربر//
myChoice = Console.ReadLine();
returnmyChoice;
پایان متد}//getChoice()
تصمیم‌گیری//
voidmakeDecision(stringmyChoice)
{
Address addr = newAddress();
switch(myChoice)
{
case"A":
case"a":
addr.name = "Meysam";
addr.address = "C# Persian";
this.addAddress(refaddr);
break;
case"D":
case"d":
addr.name = "Ghazvini";
this.deleteAddress(addr.name);
break;
case"M":
case"m":
addr.name = "CSharp";
this.modifyAddress(outaddr);
```

```
Console.WriteLine("Name is now {0}.", addr.name);
break;
case"V":
case"v":
this.viewAddresses("Meysam", "Ghazvini", "C#", "Persian");
break;
case"Q":
case"q":
Console.WriteLine("Bye.");
break;
default:
Console.WriteLine("{0} is not a valid choice", myChoice);
break;
}
}
//وارد کردن یك آدرس
voidaddAddress(refAddress addr)
{
Console.WriteLine("Name: {0}, Address: {1} added.", addr.name,
addr.address);
}
//حذف یك آدرس
voiddeleteAddress(stringname)
{
Console.WriteLine("You wish to delete {0}'s address.", name);
}
//تغییر یك آدرس
voidmodifyAddress(outAddress addr)
{
//Console.WriteLine("Name: {0}.", addr.name); //خطا رخ می‌دهد
addr = newAddress();
addr.name = "Meysam";
addr.address = "C# Persian";
}
//نمایش آدرس‌ها
voidviewAddresses(params string[] names)
{
foreach(stringname innames)
```

```
{
Console.WriteLine("Name: {0}", name);
}
}
}
```

مثال ۲-۵، نمونه تغییر یافته مثال ۱-۵ است که در آن تمامی برنامه ماژولار شده و به متدهای مختلف تقسیم شده است. در زبان #C چهار گونه پارامتر وجود دارندref : ، out، params و . value بمنظور آشنایی با پارامترها، در مثال ۲-۵ کلاسی با نام Addressبا دو فیلد از نوع رشته تولید کرده‌ایم.

درون متد()Main ، متد ()getChoice را فراخوانی کرده‌ایم تا از کاربر ورودی دریافت کنیم و این ورودی در متغیر رشته‌ای myChoice قرار می‌گیرد. سپس متغیر myChoiceرا بعنوان آرگومان به متد ()makeDecision ارسال نموده‌ایم. در اعلان ()myDecision، همانطور که ملاحظه می‌نمایید، پارامتر این متد از نوع رشته و با نام myChoiceتعریف شده است. توجه نمایید که این متغیر نیز محلی است و تنها درون متد ()makeDecision قابل استفاده است. هرگاه در اعلان متد، برای پارامترهای آن هیچ modifier آورده نشود، این پارامتر بعنوان value در نظر گرفته می‌شود. در مورد پارامترهای مقداری (value parameter) ، اصل مقدار متغیر یا پارامتر به پشته (Stack)کپی می‌شود. متغیرهایی که بصورت مقداری بعنوان پارامتر برای يك متد ارسال می‌شوند، همگی محلی بوده و تغییرات ایجاد شده بر روی آنها به هیچ وجه تغییری بر روی متغیر اصلی ایجاد نمی‌نماید .

دستور switch در متد ()makeDecision برای هر case يك متد را فراخوانی می‌نماید. فراخوانی این متدها با آنچه در متد ()Main دید مقداری متفاوت است. علاوه بر مرجع mp، در این فراخوانی‌ها از کلمه کلیدی this نیز استفاده شده است. کلمه کلیدی thisارجاعی به شیء فعلی دارد .

متد ()addAddress پارامتری از نوع ref دارد. وجود چنین پارامتری بدین معناست که مرجعی از این پارامتر به متد ارسال می‌شود و این مرجع همچنان به شیء اصلی درون heap نیز اشاره دارد چراکه آدرس شیء مورد نظر به متد کپی می‌شود. در مورد پارامترهایref ، هرگونه تغییری که بر روی متغیر محلی رخ دهد، همان تغییر بر روی متغیر اصلی نیز اعمال می‌گردد. امکان تغییر مرجع وجود ندارد و تنها شی‌ای که مورد آدرس‌دهی واقع شده، می‌تواند تغییر پیدا نماید. پارامترهای مرجعی (ref) را می‌توان به عنوان عناصر ورودی/خروجی برای متد در نظر گرفت.

پارامترهای out در مواردی استفاده می‌شوند که ارسال اطلاعات به متد از طریق پارامتر مد نظر نباشد، بلکه ارسال اطلاعات از متد مورد نظر باشد. استفاده از این پارامترها از اینرو کارآمد هستند که برنامه مجبور به کپی کردن پارامتر به متد نیست و از حجم سرباره (Overhead) برنامه می‌کاهد. در برنامه‌های عادی این مسئله چندان به چشم نمی‌آید، اما در برنامه‌های تحت شبکه که سرعت ارتباط و انتقال داده‌ها

بسیار مهم است، این پارامترها ضروری می‌شوند.

متد modifyAddress() دارای پارامتری از نوع out است. پارامترهای out فقط به متد فراخواننده آن بازگشت داده می‌شوند. از آنجائیکه این پارامترها از متد فراخواننده هیچ مقداری دریافت نمی‌کنند و فقط درون متدی که به عنوان پارامتر به آن ارسال شده‌اند قابلیت تغییر دارند، از اینرو درون این متدهایی که به آنها ارسال می‌شوند، قبل از اینکه بتوان از آنها استفاده نمود باید مقداری به آنها تخصیص داده شود. اولین خط در متد modifyAddress() بصورت توضیحات نوشته شده است. این خط را از حالت توضیحات خارج کرده و سپس برنامه اجرا کنید تا ببینید چه اتفاقی رخ خواهد داد. هنگامیکه این پارامتر مقدار دهی شود و مقداری را به متد فراخواننده خود بازگرداند، این مقدار بر روی متغیر متد فراخواننده کپی می‌گردد .توجه نمایید که پارامترهای out می‌بایست قبل از دستور return درون متد مقدار دهی شده باشند .

یکی از ویژگیهای مفید زبانC# ، وجود پارامترهای params است که بوسیله آنها می‌توان متدی را اعلان کرد که تعداد متغیری متغیر را به عنوان پارامتر دریافت نماید. پارامترهای params حتماً باید یکی از انواع آرایه تك بعدی و یا آرایه دندانه‌دار (Jagged Array)باشند. در متد makeDecision() چهار متغیر رشته‌ای را به متد viewAddresses()ارسال نموده‌ایم که این متد پارامترهای خود را بصورت params دریافت می‌نماید. همانطور که ملاحظه می‌نمایید، تعداد متغیرهای ارسالی به متد می‌تواند متغیر باشد اما دقت داشته باشید که تمامی این متغیرها در یك آرایه تك بعدی قرار گرفته‌اند. درون متد viewAddresses() نیز با استفاده از دستور foreach تمامی عناصر موجود در این آرایه را نمایش داده‌ایم. پارامترهای params فقط متغیرهای ورودی دریافت می‌نمایند و تغییرات اعمال شده تنها بر روی متغیر محلی تاثیر می‌گذارد .

خلاصه

در مقاله با ساختار کلی یك متد آشنا شدید. فرا گرفتید که در زبان C# چهار نوع پارامتر برای متدها وجود دارند .پارامترهایvalue، ref ، out و . params همانطور که گفته شد حالت پیش فرض برای پارامترها، value است مگر آنکه صریحاً مشخص گردد .