

شیء و کلاس

در کتاب برنامه‌سازی ۲ با مفاهیم شیء‌گرایی^۱ به‌طور مختصر آشنا شدید. مفهوم شیء را شناختید و با کلاس‌های آماده آشنا شدید. در زبان‌های برنامه‌نویسی شیء‌گرا امکان تعریف ویژگی‌ها و رفتارهای اشیاء فراهم شده است. برنامه‌ای که به این زبان‌ها نوشته و اجرا می‌شود، در واقع از تعدادی شیء تشکیل شده است که با یکدیگر در ارتباط و تعامل هستند. در این فصل با کلاس و روش استفاده از آن در برنامه آشنا می‌شویم.

پس از پایان این فصل انتظار می‌رود که فراگیر بتواند:

- ۱- کلاس و شیء را تعریف کند و برای آنها مثال بیاورد.
- ۲- متدها را بشناسد و تعریف نماید.
- ۳- تفاوت متد، فیلد و ویژگی را توضیح دهد.
- ۴- توصیف‌کننده‌های لازم را به‌کار بگیرد.

۴-۱- مفهوم شیء و کلاس

آیا تاکنون به نحوه ساخت یک روبات فکر کرده‌اید؟ قطعاً با ایجاد یک طرح اولیه پیش از ساخت موافق هستید. در این طرح مشخص می‌کنید که روبات چه ویژگی‌هایی دارد، چه فعالیت‌هایی می‌تواند انجام دهد و چگونه این فعالیت‌ها را باید به سرانجام برساند. پس از ساختن این طرح، یک یا چندین نمونه از روبات را می‌توانید بسازید. در زبان برنامه‌نویسی طرح همان کلاس و شیء، نمونه‌ای از کلاس است.

سؤال: در مثال ساخت روبات، کلاس و شیء کدام است؟

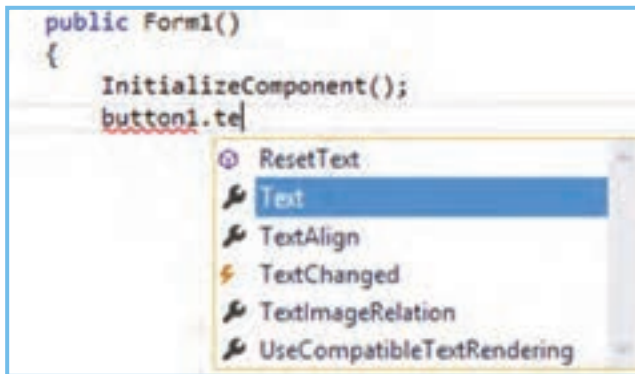
۴-۲- اشیاء آشنا

یک پروژه ایجاد کنید و روی فرم ایجاد شده، یک دکمه با نام button1 ایجاد کنید. تا همین لحظه شما دو شیء کاملاً آشنا دارید: فرم و دکمه در پروژه فعلی. البته ناگفته نماند که در پس پرده، اشیاء بیشتری به وجود آمده‌اند! به نظر شما چه اشیایی؟

به قسمت کد برنامه در فایل (Form1.cs) بروید و در متمد مقداردهی اولیه فرم (InitializeComponent()) عبارت button1 را بنویسید تا لیست هوشمند IntelliSense پدیدار شود. اعضای لیست را بررسی کنید.

سؤال: این موارد همگی متعلق به چه کنترل و شیء‌ای هستند؟

ویژگی Text را در لیست هوشمند پیدا و انتخاب کنید شکل (۴-۱).

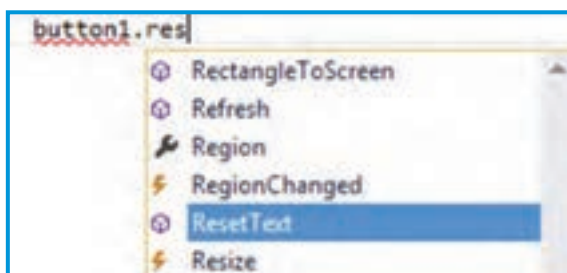


شکل ۴-۱

دستور زیر را بنویسید تا متن دکمه تغییر کند

```
button1.Text="دکمه";
```

اکنون متد EH رویداد کلیک دکمه را با دو بار کلیک بر روی آن ایجاد کنید و در آن **button1** را بنویسید تا لیست هوشمند باز شود. این بار متد ResetText را پیدا کنید (شکل ۴-۲).



شکل ۴-۲

به ایکن متفاوت این گزینه با گزینه Text در لیست دقت کنید.
سؤال: چه تفاوتی بین این دو گزینه انتخابی می بینید؟
 با انتخاب متد ResetText و کامل کردن آن داریم:

```
private void button1_Click(object sender, EventArgs e)
{
    button1.ResetText ( );
}
```

برنامه را اجرا کنید. خواهید دید که متن کنترل button1 برابر با «دکمه» است. دکمه را کلیک کنید خواهید دید که متن دکمه خالی می شود. آیا شما از چگونگی انجام این عملیات باخبر بودید؟ آیا برای آنکه با دکمه کار کنید نیاز است از کدهای درونی و شکل دهنده دکمه و نحوه عملکرد آنها با خبر باشید؟ در انجام همین عملیات ساده نکات خوبی مورد توجه قرار می گیرد:

- شیء button1 نمونه‌ای از یک کلاس است. این کلاس همان طرح اولیه برای ایجاد دکمه است و ویژگی‌های دکمه، عملیات آن و نحوه انجام عملیات در آن ذکر شده است. شیء button1 براساس این طرح ایجاد شده است.

- Text یک ویژگی برای دکمه محسوب می شود. دقیقاً همان طور که می توانید بگویید چه متنی روی یک صفحه نوشته شده و یا متن روی صفحه را تغییر دهید، می توانید از ویژگی Text دکمه برای خواندن متن دکمه یا تغییر آن استفاده کنید.

● ResetText یک متد است؛ بنابراین بیانگر عملیات یا رفتاری است. این متد، متن روی دکمه را پاک می‌کند. هنگام استفاده از این متد نیازی نیست که از چگونگی انجام این عمل مطلع باشیم که به وسیله همان کدهای درونی متد صورت می‌گیرد.

● شیء button1 از ویژگی‌ها و متدهای مختلف تشکیل شده است^۱ که همه در طرح اولیه تعریف شده است.

● شیء button1 یک هویت مستقل دارد و برای کار کردن نیاز به کدهای آن نداریم. تنها چیزی که لازم داریم این است که بدانیم این شیء دارای چه ویژگی‌ها و متدهایی است و هر کدام چه کاربردی دارند.

۳-۴- اشیاء چگونه ساخته می‌شوند؟

همان‌طور که بیان شد، شیء بر اساس کلاس ساخته می‌شود. در واقع شیء، نمونه ساخته شده بر اساس نقشه و مدلی است که کلاس مشخص می‌کند. اگر شما طرح یک روبات را داشته باشید به معنای آن نیست که آن روبات را اکنون در دست دارید. ابتدا باید هزینه کنید و قطعات آن را تهیه کرده و بر اساس طرح، سر هم کنید. ساختن یک شیء هم برای سیستم، نیاز به هزینه دارد. هر شیء که ساخته می‌شود به میزان مورد نیاز، فضایی از Ram به آن اختصاص داده می‌شود. دستور ایجاد یک شیء جدید از کلاس در حالت کلی به صورت زیر است:

() نام کلاس new

به پروژه خود باز می‌گردیم. ابتدا می‌خواهیم بینیم کلاس Form1 کجا ایجاد شده است. فایل کد Form1.cs را باز کنید. در کد این فایل همان‌طور که بارها دیده‌ایم و حتی نام برده‌ایم کلاس Form1 تعریف شده است.

```
public partial class Form1 : Form
```

```
{  
    ادامه برنامه  
}
```

ایجاد کلاس Form1
از روی کلاس Form

۱- البته همان‌طور که مشاهده می‌شود ما رویدادهایی نیز داریم که یک نحوه خاص از به‌کارگیری و اجرای متدها است.

تعریف کلاس Form1 از اینجا شروع می‌شود. البته این یک بخش از تعریف کلاس Form1 است. قسمت اصلی تعریف کلاس فرم ما، class Form1 می‌باشد. کلمات public و partial توصیف کننده می‌باشند که در ادامه فصل توضیح بیشتری دربارهٔ توصیف کننده public داده خواهد شد. همچنین در این تعریف، قسمتی که با کادر مشکی مشخص شده است، به این معناست که کلاس Form1 از روی کلاس Form ساخته شده است و به نحوی کلاس Form حق پدیری بر گردن آن دارد!

؟ سؤال: کلاس Form کجا تعریف شده است؟

در این بخش از تعریف Form1، شما تاکنون کدهای زیادی نوشته‌اید و دیده‌اید. بنابراین به خوبی با آن آشنا هستید و می‌دانید معمولاً چه کدهایی در آن نوشته می‌شود. اما بخش دیگری هم هست که آن را حتماً دیده‌اید. فایل Form1.Designer.cs را در ویژوال استودیو باز کنید. این کدها نیز بخشی از کلاس Form1 هستند.

اما سؤال اینجاست اکنون که کلاس Form1 ایجاد شده است پس کجا شیء ای از آن ساخته شده که ما آن را به عنوان یک فرم می‌توانیم در هنگام اجرا ببینیم؟ فایل Program.cs را باز کنید و کد زیر را پیدا کنید :

```
Application.Run(new Form1());
```

ایجاد شیء از کلاس
Form1

؟ سؤال: قسمت مشخص شده با کد new Form1() چه چیزی را نشان می‌دهد؟

اکنون با کنجکاوی در شیء button1 می‌خواهیم بدانیم کلاس مربوط به این شیء چیست و در کجا این شیء به وجود آمده است؟ دوباره فایل Form1.Designer.cs را باز کنید. این دو خط را پیدا کنید :

```
private System.Windows.Forms.Button button1;
```

تعیین نوع شیء button1

```
this.button1 = new System.Windows.Forms.Button();
```

ایجاد شیء button1

؟ سؤال: شیء button1 از چه کلاسی می‌باشد؟

۴-۴- شیء گرایی

جهان پیرامون ما پر از اشیاء گوناگون و متنوعی است که هر یک ویژگی‌ها و رفتارهای مخصوص به خود دارند. در جهان عینی، ما معمولاً هر جسم فیزیکی را شیء قلمداد می‌کنیم. در برنامه‌نویسی این اجسام و در کل هر موجودیتی که دارای ویژگی‌ها و فعالیت‌های خاص خود باشند به صورت یک شیء می‌تواند طرح ریزی و استفاده شود. از جمله زبان‌هایی که امکان تعریف شیء و برنامه‌نویسی بر همین اساس را مهیا کرده است زبان C# می‌باشد. استفاده از مفاهیم شیء گرایی، بر قدرت برنامه می‌افزاید و طراحی آن را حرفه‌ای‌تر می‌نماید.

۴-۵- تعریف و به‌کارگیری متد

خصوصیات و وضعیت یک شیء به وسیله فیلدها و رفتارهای اشیاء در قالب متدها تعریف می‌گردند. بنابراین محل و مکان تعریف فیلدها و متدهای یک شیء در داخل یک کلاس است:



شما تا پیش از این با ویژگی‌ها و متدهای اشیاء و کلاس‌های مختلفی کار کرده اید. اما برای ایجاد آنها در کلاس، نیاز به اطلاعات و تمرین بیشتری است. بنابراین ابتدا به تمرین ایجاد متدها می‌پردازیم و بعد گام به گام پیش می‌رویم.

می‌دانید که متد مجموعه‌ای از دستورات است که عمل خاصی را انجام می‌دهد. هر متد می‌تواند تعدادی ورودی داشته باشد و حداکثر یک مقدار برگشتی یا خروجی نیز داشته باشد. برای آنکه یک متد را ایجاد کنیم آن را داخل یک کلاس می‌نویسیم. نحوه نوشتن یک متد به صورت زیر است.

(ورودی‌ها) نام متد نوع خروجی توصیف‌کننده

```
{  
    دستورات  
}
```

ورودی‌ها به شکل زیر مشخص می‌شوند:

... نام ورودی ۲، نوع ورودی ۲، نام ورودی ۱، نوع ورودی

تعداد ورودی‌ها بستگی به شرایط دارد. گاهی ممکن است یک متد نیازی به ورودی هم نداشته باشد.

برای مثال فرض کنید می‌خواهیم متدی داشته باشیم که نام و نام خانوادگی را بگیرد و نام کامل را برگرداند. همان‌طور که از صورت مثال مشخص است دو ورودی از نوع string لازم داریم و نوع خروجی نیز string است.

```
private string GetFullName(string name,string family)
```

```
{  
    return name + " " + family ;  
}
```

سؤال! کدام string نوع خروجی را مشخص می‌کند؟

در این مثال از کلمه کلیدی private برای توصیف‌کننده متد استفاده شده است که در ادامه بحث کلاس به کاربرد آن نیز خواهیم رسید. کلمه کلیدی return برای برگشت دادن یک مقدار به کار می‌رود. در این متد می‌بینیم که رشته‌های متنی نام و نام خانوادگی به همراه یک کاراکتر فاصله در میان آنها، الحاق شده‌اند و رشته متنی حاصل، برگشت داده شده است.

سؤال: در فراخوانی متد به شکل زیر، چه مقداری در متغیر fname قرار خواهد گرفت؟

```
string fname = GetFullName("محمدی", "علی")
```

کار در کارگاه ۱: بازی دقت و قدرت حافظه

می‌خواهیم یک بازی دو نفره طراحی کنیم که نفر اول یک عدد چند رقمی وارد کند و دکمه شروع را بزند و بازیکن دوم اقدام به نوشتن همان رقم به صورت برعکس نماید. با زدن دکمه پایان از طرف بازیکن دوم بازی به اتمام رسیده و نتیجه اعلام شود.

الگوریتم یا روش انجام کار: با زدن دکمه شروع، عدد وارد شده ذخیره می‌شود و متن کادر



متنی پاک می‌شود. با زدن دکمه پایان عدد وارد شده توسط کاربر با عکس رشته متنی ذخیره شده مقایسه می‌شود و پیام مناسب پیروزی یا شکست نمایش داده می‌شود. برای به دست آوردن عکس رشته متنی یک متد می‌نویسیم.

شکل ۳-۴- فرم برنامه

۱- ایجاد پروژه و فرم برنامه: پروژه جدید WFA را ایجاد می‌کنیم و اندازه فرم را به مقدار مناسب تعیین می‌نماییم.

۲- وارد کردن کنترل‌ها به فرم: در این فرم ما به کنترل‌های زیر نیاز داریم:

جدول ۲-۴- ویژگی‌های برچسب

Label	
ویژگی	مقدار
Name	label1
Text	عدد مورد نظران را وارد کنید
AutoSize	True
RightToLeft	Yes

جدول ۱-۴- ویژگی‌های کادر متن

TextBox		
ویژگی		مقدار
Name		input
Text		
Font	size	14

جدول ۳-۴- ویژگی‌های دکمه‌ها

Button		
ویژگی	مقدار	مقدار
Name	start	finish
Text	شروع	پایان
Enabled	True	False

۳- تعریف متد: ابتدا یک متد در فرم می‌سازیم که یک رشته متنی را بگیرد و عکس آن را تحویل بدهد. توجه داشته باشید که گرچه ما عدد وارد می‌کنیم اما در واقع با رشته متنی وارد شده کار می‌کنیم و هر چه که باشد آن را عکس می‌نماییم.

```
private string Reverse (string str)
{
    string result = "";
    for (int c = str.Length-1; c >= 0; c--)
    {
        result += str [c];
    }
    return result;
}
```

در این متد حرف به حرف از انتهای رشته متنی ورودی str تا ابتدای آن خوانده می‌شود و به رشته متنی result که در ابتدای کار خالی است، اضافه می‌شود. در پایان نیز رشته متنی result برگشت داده می‌شود.

اکنون متد EH رویداد کلیک دکمه‌ها را می‌نویسیم. برای دکمه start کافی است که با زدن آن دکمه finish فعال شود و خود دکمه start غیر فعال شود. همچنین نیاز است که رشته متنی وارد شده قبل از پاک شدن، در یک متغیر ذخیره شود. بنابراین ما یک متغیر در داخل کلاس فرم ایجاد می‌کنیم تا برای متدهای دیگر قابل شناسایی باشد و مقدار خود را نیز حفظ نماید.

```
public partial class Form2 : Form
{
    string goal;
    ادامه برنامه
}

```

۴- تعریف متد *EH* رویدادها: متد *EH* رویداد کلیک دکمه start را این چنین ایجاد

می نماییم:

```
private void start_Click (object sender, EventArgs e)
{
    finish.Enabled = true;
    start.Enabled = false;
    goal = input.Text;
    input.Text = "";
}

```

متد *EH* رویداد دکمه پایان را نیز به صورت زیر ایجاد می کنیم:

```
private void finish_Click (object sender, EventArgs e)
{
    finish.Enabled = false;
    start.Enabled = true;
    if (Reverse(goal) == input.Text);
        MessageBox.Show("آفرین! جواب درست را وارد کردید");
    else
        MessageBox.Show("متأسفانه جواب شما درست نبود!");
}

```

با کلیک دکمه پایان، خود دکمه غیر فعال می شود. دکمه شروع فعال می شود و عکس رشته متنی اصلی توسط متد *Reverse* بدست آمده، با رشته متنی ورودی توسط بازیکن دوم مقایسه شده و پیام مناسب داده می شود.

توسعه و بهبود برنامه

می‌خواهیم دکمه‌ای در فرم بگذاریم که اگر کلیک شود متن ابتدایی و عکس آن را به صورت پیام نشان دهد.



شکل ۴-۴- نمایش جواب

دکمه‌ای با مشخصات زیر به فرم اضافه می‌کنیم.

جدول ۴-۴- ویژگی‌های دکمه

Button	شیء
ویژگی	مقدار
Name	showAnswer
Text	نمایش جواب
Enabled	True

متد رویداد کلیک آن را به صورت زیر می‌نویسیم.

```
private void showAnswer_Click(object sender, EventArgs e)
{
    MessageBox.Show("می‌باشد "+ Reverse(goal) + " و عکس آن برابر با " + goal + " متن وارد شده برابر با");
}
```

همان طور که می‌بینید از مزایای متد آن است که ما یک بار آن را تعریف می‌کنیم و بارها آن را می‌توانیم استفاده کنیم. همچنین اگر بخواهیم آن را در برنامه‌ای دیگر استفاده کنیم به راحتی می‌توانیم متد را در برنامه مورد نظر کپی کنیم و به کار ببندیم.

توسعه و بهبود برنامه

برای اطمینان از ورود اعداد در کادر متن از یکی از روش‌های زیر می‌توان استفاده کرد:

- ۱- برای آنکه فقط با اعداد کار شود ابتدا بررسی کنید که آیا رشته متنی وارد شده یک عدد است یا خیر؟
- ۲- از طریق کدنویسی کادر متنی را طوری تغییر دهید که فقط عدد وارد شود.

۴-۶ استفاده از کلاس و شیء

مثال ۴-۱: یک ساعت را به عنوان یک شیء در نظر بگیرید. می‌خواهیم ساده‌ترین ویژگی‌ها و عملیات مربوط به یک ساعت را مشخص و در یک کلاس تعریف کنیم.



شکل ۴-۵

الگوریتم یا روش انجام کار: برای نمایش ساعت، از سه ویژگی به نام ساعت، دقیقه و ثانیه استفاده می‌شود. مدل ساعت (عقربه‌ای یا دیجیتال)، رنگ، شکل ظاهری، جنس و قیمت از ویژگی‌های دیگر ساعت هستند اما برای نمایش زمان فقط به سه ویژگی اول نیاز داریم.

سؤال: به جز نمایش زمان و تنظیم ساعت که از جمله عملیاتی است که روی ساعت تعریف می‌شود، چه عملیات دیگری به فکر شما می‌رسد؟

در این مثال، نام کلاس را Clock انتخاب می‌کنیم. روش نوشتن نام کلاس، روش پاسکال و روش نوشتن نام فیلدها، روش کوهان شتری است (قطعه برنامه ۱-۶).

Class Clock

```
{
```

فیلدها

```
byte hour, minute, second;
```

متدها

```
public void SetClock (byte h, byte m, byte s)
```

```
{
```

```
hour = h;
```

```
minute = m;
```

```
second = s;
```

```
}
```

```
public void ShowClock ( )
```

```
{
```

```
MessageBox.Show(hour.ToString ( ) + " : " + minute.ToString( )
```

```
+ " " + second.ToString ( ));
```

```
}
```

قطعه برنامه ۱-۶_ کلاس ساعت

سه متغیر از نوع byte برای نگهداری مقدار ساعت، دقیقه و ثانیه استفاده شده است. متغیرهای hour، minute و second **فیلدهای کلاس** نامیده می‌شوند. متد SetClock() برای تنظیم ساعت استفاده شده که دارای سه پارامتر است. همان‌طور که از خط عنوان متد دیده می‌شود، سه عدد به عنوان پارامتر به این متد وارد می‌شود و با اجرای متد مذکور، ویژگی‌های شیء ساعت، مقدار دهی جدید شده و تنظیم می‌شود. به نوع خروجی این متد توجه کنید! کلمه کلیدی void می‌گوید که متد، هیچ مقداری را بر نمی‌گرداند. همان‌طور که می‌بینید در متد، دستور return استفاده نشده است و مقداری بازگشت داده نمی‌شود^۱

۱- در متدهای که خروجی ندارند دستور return برای خروج از متد می‌تواند به کار رود.

متد ShowClock() وظیفه نمایش ساعت بر روی صفحه را به عهده دارد. این متد خروجی ندارد.

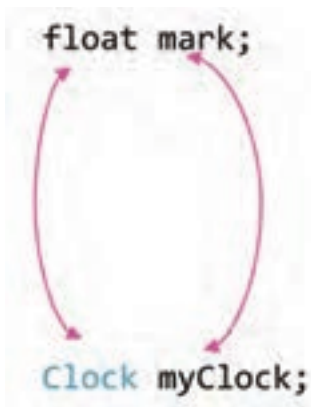
❓ **سؤال:** از چه دستوری برای نمایش ساعت استفاده شده است؟

❓ **سؤال:** آیا کلاسی که ایجاد می کنیم باید در کلاس فرم باشد؟

کلاس می تواند در هر جای دلخواه غیر از داخل متدها تعریف شود :

نکته

توجه شود، اگر کلاسی را بیرون از کلاس فرم اما در همان فایل ایجاد می کنید آن را بعد از کلاس فرم بنویسید چون برای محیط طراحی فرم اشکال به وجود می آید. هرچند در زمان اجرا خطایی رخ نخواهد داد.



شکل ۴-۶

به خاطر دارید که برای تعریف یک متغیر ابتدا نوع آن و سپس نام متغیر را می نوشتیم. مثلاً برای تعریف متغیری از نوع اعشاری دستور float mark را به کار می بردیم. این دستور، متغیر mark از نوع float است.

برای ایجاد شیء ساعت از نوع کلاس Clock هم از همان شیوه استفاده می کنیم :

```
Clock myClock;
```

در دستور بالا شیء myClock از نوع Clock است.

اما تعریف بالا هنوز کامل نیست و فقط مرحله اول تعریف

است. در زبان سی شارپ برای تعریف یک شیء از یک کلاس، از عملگر new استفاده می کنیم. به وسیله عملگر new در دستور زیر یک شیء ساخته می شود اما این شیء نامی ندارد که هنگام استفاده، آن را به کار ببریم.

```
new Clock();
```

در دستور بالا، شیء ایجاد شده و حافظه ای برای آن تخصیص داده می شود اما چگونه می توان از آن استفاده نمود؟

برای اینکه شیء مورد نظر نامی داشته باشد تا بتوان از این نام در برنامه استفاده کرد، دستور بالا را به صورت زیر تصحیح می کنیم :

```
myClock = new Clock();
```

بنابراین برای استفاده از شیء ایجاد شده در دستوره‌های بعدی می‌توانیم به صورت زیر عمل نماییم

`Clock myClock;` ← مرحله اول

`myClock = new Clock();` ← مرحله دوم

می‌توانید دو خط فوق را به صورت زیر خلاصه کنید :

`Clock myClock = new Clock();`

۴-۷- دسترسی به فیلدها و متدهای یک شیء

پس از ایجاد شیء می‌توانید به فیلدها و متدهای شیء دسترسی داشته باشید و آنها را فراخوانی کنید. برای دسترسی به اجزا و اعضای یک شیء، کافی است نام شیء و سپس نام عضو (فیلد یا نام متد) را به دنبال آن ذکر کنیم که با علامت نقطه از یکدیگر جدا می‌شوند.

نام عضو. نام شیء

برای مثال برای تنظیم ساعت، متد `SetClock()` و برای نمایش ساعت از فراخوانی متد `ShowClock()` استفاده می‌کنیم.

کار در کارگاه ۲



مطابق شکل ۴-۷ فرمی با دو دکمه برای شیء ساعت، آن را طراحی کنید و کدهای موردنیاز را بنویسید.

شکل ۴-۷

الگوریتم یا روش انجام کار :

۱- ایجاد پروژۀ جدید WFA ایجاد می‌نماییم.

۲- اضافه کردن کنترل‌ها : ابتدا دو groupBox ایجاد می‌کنیم و مقدار Text آنها را خالی

می‌کنیم. سپس طبق شکل ۷-۴ به این دو groupBox، کنترل‌های زیر را اضافه می‌کنیم.

جدول ۵-۴- ویژگی‌های دکمه

Button		
ویژگی	مقدار	مقدار
Name	setClockBtn	showClockBtn
Text	تنظیم ساعت	نمایش ساعت

جدول ۶-۴- ویژگی‌های کنترل عددی افزایشی کاهشی

NumericUpDown			
ویژگی	مقدار	مقدار	مقدار
Name	hourBox	minuteBox	secondBox
Minimum	۰	۰	۰
Maximum	۲۳	۵۹	۵۹

۳- تعریف کلاس و شیء ساعت : کلاس Clock که در قطعه برنامه ۱-۶ آمده را بعد از کلاس فرم می‌نویسیم. در درون کلاس فرم، متغیر myClock را از جنس Clock تعریف می‌کنیم.

```
public partial class Form1: Form
{
    Clock myClock = new Clock();
```


۴- اضافه کردن متد *EH* رویدادها : متد *EH* رویداد کلیک دکمه `setClockBtn` را به

صورت زیر می نویسیم :

```
private void setClockBtn_Click(object sender, EventArgs e)
{
    myClock.SetClock((byte)hourBox.Value,(byte)minutBox.Value,(byte)
    secondBox.Value);
}
```

سؤال: علت آنکه ما برای تبدیل نوع، از عبارت `byte` استفاده کرده ایم چیست؟

متد *EH* رویداد کلیک دکمه `showClockBtn` را نیز به صورت زیر می نویسیم :

```
private void showClockBtn_Click(object sender, EventArgs e)
{
    myClock.ShowClock();
}
```

در نتیجه این طراحی، اگر کادرهای عددی را به صورت ۰:۳۱:۹ تنظیم کنیم و دکمه تنظیم ساعت را بزنیم و سپس دکمه نمایش ساعت را کلیک کنیم کادر زیر نمایش داده خواهد شد (شکل ۴-۸).



شکل ۴-۸

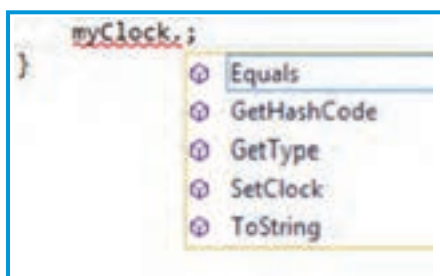
۴-۸- توصیف کننده‌ها

در کلاس Clock توصیف کننده متد ShowClock را از private به public تغییر دهید. سپس ماوس را روی کد فراخوانی متد ShowClock قرار دهید، شرح متد به صورت شکل ۴-۹ ظاهر خواهد شد.

```
private void showClockBtn_Click(object sender, EventArgs e)
{
    myClock.ShowClock();
}
void Clock.ShowClock()
Error:
'WindowsFormsApplication3.Clock.ShowClock()' is inaccessible due to its protection level
```

شکل ۴-۹- خطای عدم دسترسی به متد فراخوانی شده

خط زیر متد نشان دهنده خطا است. وقتی ماوس را روی فراخوانی متد می‌بریم راهنما به ما می‌گوید این متد به دلیل سطح محافظتی غیر قابل دسترس است! حتی اگر بخواهیم دوباره آن را بنویسیم خواهیم دید که دیگر این متد در لیست هوشمند دیده نمی‌شود.



شکل ۴-۱۰- لیست هوشمند شیء myclock

تفاوت دو توصیف کننده را ببینید :

جدول ۴-۷

مثال	توضیح	توصیف کننده
مانند متغیرهای hour, minute و second که private هستند و می توان توسط دستورات داخلی کلاس Clock به آنها دسترسی داشت اما نمی توان از بیرون به این متغیرها دست پیدا کرد.	دسترسی از بیرون به متغیر یا متدی که با این توصیف کننده مشخص شده، وجود ندارد و در واقع فقط داخل کلاس (شیء) قابل استفاده است. اگر متد یا متغیری توصیف کننده نداشته باشد به صورت پیش فرض private خواهد بود.	private
مانند متدهای ShowClock و SetClock در کلاس Clock	به یک متد یا متغیر از شیء ای که در کلاس به صورت public تعریف شده است می توان از بیرون کلاس، دسترسی داشت.	public

کاردرگاه ۳

مثال ۴-۲: می خواهیم برای یک بازی جنگی فضایی^۱ که در آن سفینه های^۲ مختلفی وجود دارد و با هم در نبرد^۳ هستند، کلاسی برای سفینه تعریف کنیم. پروژه ای از نوع WFA ایجاد کرده (شکل ۴-۱۱) و روی فرم یک دکمه با ویژگی های مشخص شده در جدول ۴-۸ قرار دهید.



شکل ۴-۱۱

۱- Space Wars

۲- Spaceships

۳- Battle

جدول ۸-۴- ویژگی دکمه



شکل ۱۲-۴- فرم مثال ۲-۴

شیء Button	
ویژگی	مقدار
Name	ShowButton
Text	show

الگوریتم یا روش انجام کار: یک سفینه فضایی ویژگی‌های مختلفی دارد، که بسته به نوع و سناریوی بازی، بعضی از آنها اهمیت پیدا می‌کند. در یک سفینه جنگی، ویژگی‌ها قدرت آتش^۱ و اندازه مهمات^۲ و در یک سفینه باری^۳ ویژگی ظرفیت باربری^۴ اهمیت دارد. ویژگی‌های مشترکی نیز در هر دو نوع سفینه مانند سرعت^۵، سپردفاعی^۶ و اندازه سوخت^۷ وجود دارد. برای سادگی کار، فعلاً ویژگی مشترک بین سفینه‌ها را در یک کلاس تعریف می‌کنیم. بنابراین سه فیلد برای ویژگی‌های اندازه سرعت سفینه، قدرت سپر دفاعی و اندازه سوخت به صورت عدد صحیح بین ۰ تا ۱۰۰ (به عنوان درصد) تعریف می‌کنیم.

بنابراین تعریف فیلدهای کلاس سفینه چنین خواهد بود:

```
public class SpaceShip
{
    public int fuel;
    public int shield;
    public int speed;
}
```

۱- Fire Rate

۲- Armor

۳- Cargo Spaceship

۴- Storage

۵- Speed

۶- Shield

۷- Fuel

علاوه بر تعریف فیلدها، نیاز به انجام عملیات بر روی سفینه است. یک عمل معمول در سفینه، پر کردن^۱ سوخت سفینه و با کامل کردن سلامتی^۲ سفینه است. با توجه به تعریف کلاس فوق، سه روش در اختیار داریم:

۱- استفاده از دستور انتساب و دسترسی به فیلدهای شیء

۲- استفاده از یک متد برای تغییر دادن مقدار فیلدها

۳- استفاده از ویژگی (Property) (راه حل استاندارد و بهینه در C#)

برای استفاده از هر یک از سه روش فوق ابتدا باید شیء از نوع کلاس سفینه ساخت. به دلیل آنکه ما می‌خواهیم شیء سفینه را در طول برنامه نگهداری و استفاده کنیم، در کلاس فرم متغیری از نوع کلاس سفینه تعریف کرده و آن را ایجاد می‌کنیم.

```
public partial class Form1 : Form
```

```
{
```

```
    SpaceShip basicSpaceShip; → تعریف متغیر
```

```
    public Form1 ( )
```

```
    {
```

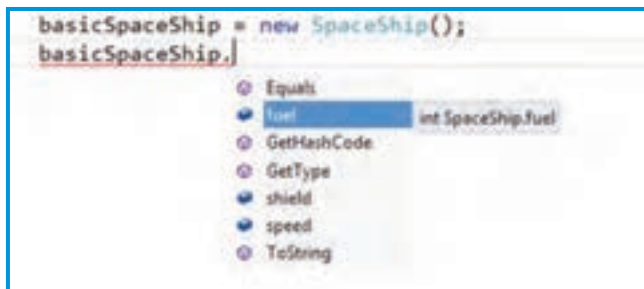
```
        InitializeComponent ( );
```

```
        basicSpaceShip = new SpaceShip ( ); → ایجاد شیء سفینه
```

```
    }
```

روش اول - استفاده از دستور انتساب و دسترسی به فیلدهای شیء: با توجه به اینکه

فیلدهای کلاس از نوع public تعریف شده اند، در خارج از کلاس سفینه نیز می‌توان به آنها دسترسی داشت و آنها را مقداردهی کرد.



شکل ۱۳-۴- لیست هوشمند شیء سفینه

همان طور که در شکل ۱۳-۴ مشاهده می‌کنید در لیست ظاهر شده در IntelliSense نام هر سه فیلد fuel, shield, speed وجود دارد. علاوه بر سه فیلد، نام چهار متد نیز دیده می‌شود. آیا این متدها را ما تعریف کرده‌ایم؟ این متدها برای تمام اشیاء وجود دارند. علت وجود آنها را با پیشرفت در حوزه برنامه‌نویسی و تحقیق درخواهید یافت.

اکنون دستورات پر کردن سوخت، سرعت و سپر دفاعی کامل سفینه را با استفاده از دستورات انتساب بر روی فیلدها مشاهده می‌کنید.

```
namespace classSpaceShip
{
    public partial class Form1 : Form
    {
        SpaceShip basicSpaceShip;|
        public Form1()
        {
            InitializeComponent ();
            basicSpaceShip = new SpaceShip ();
            basicSpaceShip.fuel = 100;
            basicSpaceShip.speed = 100;
            basicSpaceShip.shield = 100;
        }
    }
}
class SpaceShip
{
    public int fuel;
    public int speed;
    public int shield;
}
```

مقداردهی
فیلدهای
شیء سفینه

برنامه ۶-۲- مقداردهی فیلدها در مثال ۶-۲

با اجرای برنامه ۶-۲، در داخل فیلدهای شیء basicSpaceShip عدد ۱۰۰ به منزله مقدار صد درصد آن ویژگی‌ها قرار می‌گیرد.

برای اطلاع از مقدار فیلدها، در متد EH رویداد کلیک دکمه showButton دستوری بنویسید که مقدار فیلدها را در پنجره پیام نمایش دهد. (شکل ۴-۱۴)

```
private void showButton_Click(object sender, EventArgs e)
{
    string s = "speed: " + basicSpaceShip1.Speed+
        "\n fuel: " + basicSpaceShip1.Fuel+
        "\n shield: " + basicSpaceShip1.Shield;
    MessageBox.Show(s, "مقدار خصوصیات سفینه");
}
```

در صورت کلیک دکمه showButton پنجره پیام مانند شکل ۴-۱۴ ظاهر می‌شود.



شکل ۴-۱۴

روش دوم — استفاده از یک متد برای تغییر دادن مقدار فیلدها: روش انتساب و تغییر

دادن مقدار فیلدها، چندان مناسب نیست. ممکن است در جای دیگری از کد برنامه، دستوری بخواهد از این کلاس استفاده کند و برای پر کردن سوخت سفینه، به جای اعداد ۰ تا ۱۰۰ از محدوده دیگری استفاده کند. مثلاً برای بالا بردن سوخت سفینه عدد ۵۰۰ یا ۱۰۰۰ و حتی بالاتر را استفاده کند که در این صورت ساختار برنامه را به هم می‌زند. برای این کار بهتر است که دسترسی به فیلدهای یک کلاس را محدود کنیم و اجازه ندهیم برنامه‌ای خارج از کلاس بتواند به آنها دسترسی داشته باشد. برای این منظور دسترسی آنها را از نوع private تعریف می‌کنیم که حوزه شناخت آنها فقط محدود به کلاس شود. در این صورت لازم است متدهایی را برای تغییر دادن مقدار فیلدها پیش بینی کنیم تا عددی که می‌خواهد در فیلد قرار گیرد را کنترل کرده و مقدار مناسب را در آن فیلد قرار دهند. نوع دسترسی

این متدها معمولاً باید **public** باشد تا برنامه‌های دیگر بتوانند از طریق فراخوانی این متدها، فیلدها را به طور صحیح مقداردهی نمایند.

این متدها را در کجای برنامه تعریف کنیم و یا تایپ کنیم؟ این متدها همان رفتارها و عملیات شیء مورد نظر ما هستند؛ بنابراین متدهای مربوطه در داخل کلاس قرار می‌گیرند.

```
public class SpaceShip
{
    private int fuel;
    private int shield;
    private int speed;
    public void Recharge ( )
    {
        fuel = 100;
        shield = 100;
        speed = 100;
    }
}
```

تعریف متد
در کلاس

هنگامی که شیء از نوع کلاس سفینه ایجاد می‌شود و می‌خواهید به اجزای شیء دسترسی داشته باشید، به لیست IntelliSense دقت کنید که دیگر خبری از سه فیلد نیست. چرا؟

```
InitializeComponent();
basicSpaceShip = new SpaceShip();
basicSpaceShip.
}
  Equals
  GetHashCode
  GetType
  Recharge void SpaceShip.Recharge()
  ToString
```

شکل ۱۵-۴

با توجه به مطالب گفته شده و تغییری که در کلاس سفینه فضایی دادیم، برنامه چنین خواهد شد :

```
namespace classSpaceShip
{
    public partial class Form1 : Form
    {
        SpaceShip basicSpaceShip;
        public Form1()
        {
            InitializeComponent ();
            basicSpaceShip = new SpaceShip ();
            basicSpaceShip.Recharge ();
        }
    }
}
class SpaceShip
{
    private int fuel;
    private int speed;
    private int shield;
    public void Recharge ( )
    {
        fuel = 100;
        speed = 100;
        shield = 100;
    }
}
```

همچنین برای اطلاع از مقدار یک فیلد، نیز می‌توانید متدی بنویسید که خروجی آن مقدار فیلد را به برنامه اصلی بازگرداند. در تعریف زیر، سه متد برای اطلاع از مقدار فیلدهای شیء نوشته شده است:

```
public class SpaceShip
{
    private int fuel;
    private int shield;
    private int speed;
    public void Recharge ( )
    {
        fuel = 100;
        shield = 100;
        speed = 100;
    }
    public int GetFuel ( )
    {
        return fuel;
    }
    public int GetShield ( )
    {
        return Shield;
    }
    public int GetSpeed ( )
    {
        return Speed;
    }
}
```

بنابراین در این روش، اگر بخواهید هر فیلد را به صورت مجزا مقداردهی کنید یا مقدارش را بخوانید باید برای هر فیلد دو متد بنویسید: یک متد برای انتساب مقدار به فیلد و متد دیگری برای اطلاع از مقدار فیلد. یعنی برای سه فیلد، باید شش متد بنویسیم. آیا این روش کمی پردردسر نیست؟

برای دسترسی به مقدار فیلدها که private شده‌اند باید متدهای نوشته شده را در برنامه اصلی فراخوانی کنیم. بدنهٔ متد EH رویداد کلیک دکمه showButton به صورت زیر تغییر خواهد کرد :

```
string s = "speed: " + basicSpaceShip.GetSpeed ()+
          "\n fuel: " +basicSpaceShip.GetFuel( )+
          "\n shield: " +basicSpaceShip.GetShield ( );
MessageBox.Show(s, "مقدار خصوصیات سفینه");
```

روش سوم - استفاده از ویژگی (Property): با توجه به مشکلاتی که در روش‌های اول و دوم داشتیم راه حلی که در زبان C# پیش بینی شده است استفاده از قابلیت ویژگی (Property) است. در این روش متدی نوشته می‌شود که شبیه یک فیلد مورد استفاده برنامه نویس قرار می‌گیرد. روش نوشتن نام ویژگی‌ها مانند نام متدها، روش پاسکال است. مثلاً تعریف یک ویژگی برای دسترسی کنترل شده به فیلد سوخت چنین است :

```
public class SpaceShip
{
    private int fuel;
    private int shield;
    private int speed;

    public int Fuel
    {
        get { return fuel; }
        set { fuel = value;}
    }
}
```

به روش تعریف ویژگی Fuel دقت کنید. باید قبول کنیم که در تعریف یک ویژگی، ترکیبی از روش تعریف فیلد و روش تعریف متد با هم استفاده شده است. معمولاً نام ویژگی‌ها را مانند نام فیلدها انتخاب می‌کنند با این تفاوت که روش نوشتن نام ویژگی از روش پاسکال است. کلمات زرروده

value، set، get و return در تعریف یک ویژگی استفاده می‌شود این چهار کلمه کلیدی در جدول زیر توضیح داده شده است :

جدول ۹-۴- کلمات کلیدی مورد استفاده در تعریف ویژگی

کلمه کلیدی	شرح کار
get	بخشی از تعریف ویژگی را مشخص می‌کند که برای اطلاع از مقدار یک فیلد به کار می‌رود.
set	بخشی از تعریف ویژگی را مشخص می‌کند که برای انتساب یک مقدار به یک فیلد به کار می‌رود. در این بخش می‌توانیم با استفاده از دستور if مقداری که قرار است در فیلد قرار گیرد را کنترل و بررسی کنیم.
value	مقداری است که در برنامه معین شده و قرار است در فیلد قرار گیرد.
return	دقیقاً عملکردی مانند مقدار برگشتی متدها به برنامه اصلی را دارد.

در تعریف ویژگی Fuel برای سادگی کار، کنترلی بر روی value صورت نگرفته است اما در تعریف زیر برای مقادیر ویژگی‌های دیگر چنین کاری انجام شده است :

```
public class SpaceShip
{
    private int fuel;
    private int shield;
    private int speed;

    public void Recharge ( )
    {
        fuel = 100;
        shield = 100;
        speed = 100;
    }
}
```

```
public int Fuel
{
    get { return fuel; }
    set { fuel = value;}
}
```

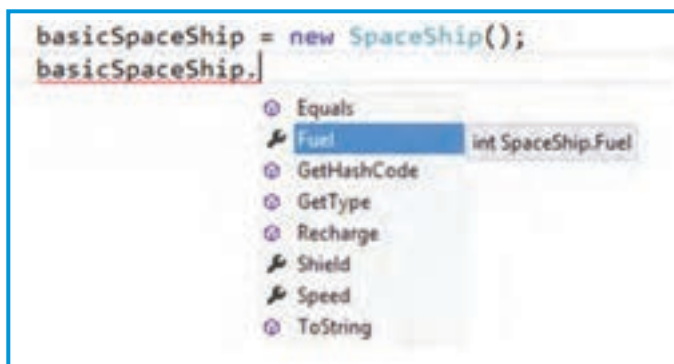
```
public int Shield
{
    get { return shield; }
    set { if ((value<=100) && (value>=0))
        shield=value;
    }
}
```

```
public int Speed
{
    get { return speed; }
    set { if ((value<=100) && (value>=0))
        speed=value;
    }
}
```

آیا می‌توانید تشخیص دهید هر کادر رنگی معرف چه عضوی از یک کلاس است؟ پاسخ شما باید یکی از کلمات فیلد، متد و ویژگی باشد.

به تدریج که جلو می‌رویم، تعریف کلاس سفینه کامل می‌شود و شما با مفاهیم پایه‌ای در تعریف یک کلاس و زبان شیء‌گرایی آشنا می‌شوید.

اکنون نوبت نمایش ویژگی‌های سفینه است. از ویژگی‌ها مانند یک متغیر برای اطلاع از وضعیت سفینه و همچنین تغییر مقدار فیلدها استفاده می‌کنیم.



شکل ۴-۱۶

در لیست هوشمند (IntelliSense)، ویژگی‌ها با علامت آچار مشخص می‌شوند. نمونه ای از برنامه اصلی چنین خواهد بود:

```
public partial class Form1 : Form
{
    SpaceShip basicSpaceShip;
    public Form1()
    {
        InitializeComponent();
        basicSpaceShip = new SpaceShip();
    }
    private void showButton_click(object sender, EventArgs e)
    {
        string s = "speed: " + basicSpaceShip.Speed +
            "\n fuel: " + basicSpaceShip.Fuel +
            "\n shield: " + basicSpaceShip.Shield;
        MessageBox.Show(s, "مقدار خصوصیات سفینه");
    }
}
```

مثال ۳-۴: می‌خواهیم در برنامه اصلی دو شیء از نوع سفینه ایجاد کرده و اندازه سرعت یکی ۱۰۰ و دیگری را برابر ۵۰ تنظیم کنیم.

الگوریتم یا روش انجام کار: برای ایجاد دو سفینه کافی است دو متغیر مرجع مختلف تعریف کرده و با استفاده از عملگر new، عمل ایجاد شیء را انجام دهیم. سپس ویژگی Speed هر یک را مانند یک متغیر معمولی با دستور انتساب مقدار دهی کنیم.

```
public partial class Form1 : Form
{
    SpaceShip basicSpaceShip1, basicSpaceship2;
    public Form1()
    {
        InitializeComponent ();
        basicSpaceShip1 = new spaceShip ();
        basicSpaceShip2 = new spaceShip ();
        basicSpaceShip1.Speed = 100;
        basicSpaceShip2.Speed = 50;
    }
}
```

سؤال؟ در برنامه بالا مقدار سوخت و سپر دفاعی هر یک از سفینه‌ها چقدر است؟

برای پاسخ به سؤال کافی است برنامه را اجرا کرده و روی دکمه showButton کلیک کنیم تا مقدار تمام ویژگی‌های سفینه ۱ را در پنجره پیام نمایش دهد.

```
private void showButton_click (object sender, EventArgs e)
{
    string s = "speed: " + basicSpaceShip1.Speed+
        "\n fuel: " + basicSpaceShip1.Fuel+
        "\n shield: " + basicSpaceShip1.Shield;
    MessageBox.Show(s, "مقدار خصوصیات سفینه");
}
```



شکل ۱۸-۴- خروجی مثال ۳-۴

سؤال؟ چرا مقدار فیلدهای سوخت و سپر دفاعی صفر چاپ شد؟ چگونه آنها مقداردهی شدند؟ همان طور که به یاد دارید، چنانچه متغیری تعریف کنیم و آن را مقداردهی نکنیم، مترجم خطا می‌دهد. اما در تعریف کلاس سفینه، مقدار فیلدها را مقداردهی نکردیم اما مترجم ایرادی نگرفته است.

پاسخ این سؤالات چنین است که به هنگام ایجاد یک شیء، یک متد که به نام متد سازنده^۱ معروف است به طور خودکار و پیش فرض اجرا می‌گردد و فیلدهای شیء را مقداردهی و معین می‌کند. اگر ما متدی را به عنوان متد سازنده معرفی نکنیم، هنگام ایجاد شیء اگر فیلدهای مقداردهی اولیه نشده باشد مقدار پیش فرض به آن داده می‌شود. مقدار پیش فرض برای فیلدهای نوع عددی، عدد صفر و برای فیلدهای رشته ای مقدار تهی یا null است.

۴-۹- متد سازنده

یکی از راه‌های مقداردهی اولیه به فیلدها و در واقع راه اصلی آن استفاده از متد سازنده است. متد سازنده، متدی است که در هنگام ایجاد شیء فراخوانی شده و عملیات آن انجام می‌شود. در واقع هرگاه بخواهیم در هنگام ایجاد یک شیء، عملیاتی انجام شود از متد سازنده استفاده می‌کنیم. این عملیات می‌تواند مقداردهی اولیه فیلدها یا هر دستور دیگری باشد.

در تعریف یک کلاس می‌توانیم متد سازنده‌ای به دلخواه و طبق نیاز برنامه تعریف کنیم که در این صورت این متد با ایجاد شیء به طور خودکار اجرا می‌گردد. نام متد سازنده هم نام با نام کلاس است و نوعی به عنوان خروجی نباید برای آن معین شود. متد سازنده می‌تواند با پارامتر و یا بدون پارامتر باشد.

^۱ Constructor

علاوه بر آن می‌توان چندین متد سازنده، داشت یعنی چندین متد هم نام با نام کلاس داشته باشیم که تفاوت آنها در پارامترهای ورودی متد است.

مثلاً فرض کنید می‌خواهیم با ایجاد یک سفینه، مقادیر همه فیلدها برابر ۵۰ شود. در این صورت خواهیم داشت :

```
public class SpaceShip
```

```
{
```

```
    private int fuel;
```

```
    private int shield;
```

```
    private int speed;
```

```
    public Spaceship ( )
```

متد سازنده

```
{
```

```
        fuel = 50;
```

```
        shield = 50;
```

```
        speed = 50;
```

```
}
```

```
    public void Recharge ( )
```

```
{
```

```
        fuel = 100;
```

```
        shield = 100;
```

```
        speed = 100;
```

```
}
```

ادامه تعریف کلاس



شکل ۱۹-۴

اگر در برنامه اصلی یک سفینه ایجاد کنیم متد سازنده به طور خودکار اجرا شده و مقدار فیلدها برابر ۵۰ می‌شود. بنابراین با اجرای همان برنامه قبل پنجره پیام مانند شکل ۱۹-۴ خواهد بود.

مثال ۴-۴: می‌خواهیم در هنگام ایجاد یک سفینه، اگر مایل بودیم اندازه سوخت، سپر دفاعی و سرعت سفینه با اعداد دلخواه تعیین شود.

الگوریتم یا روش انجام کار: چون می‌خواهیم در هنگام ایجاد شیء، مقداردهی صورت گیرد لذا باید از متد سازنده بدین منظور استفاده کنیم. بنابراین کافی است که متد سازنده دیگری به تعریف کلاس مثال قبل اضافه نماییم. متد سازنده جدید باید دارای سه پارامتر ورودی باشد که هر یک از فیلهای سفینه با پارامتر مربوطه مقداردهی شود:

```
public Spaceship (int f , int sh , in sp)
{
    Fuel = f;
    Shield = sh;
    Speed = sp;
}
```

برنامه اصلی چنین خواهد بود:

```
public partial class Form1 : Form
{
    SpaceShip basicSpaceShip1, basicSpaceship2;
    public Form1()
    {
        InitializeComponent ();
        basicSpaceShip1 = new spaceShip (100,80,90); ← سازنده با پارامتر ورودی
        basicSpaceShip2 = new spaceShip ( ); ← سازنده بدون پارامتر ورودی
    }
    private void showButton_click (object sender, EventArgs e)
    {
        string s = "safineh1\t\t safineh2 \n" +
            "speed: " + basicSpaceship1 . Speed +
            "\t\t speed: " + basicSpaceShip2 . Speed +
            "\n fuel: " + basicSpaceShip1 . Fuel +
            "\t\t fuel: " + basicSpaceship2 . Fuel +
            "\n shield: " + basicSpaceShip1 . Shield +
            "\t\t shield: " + basicSpaceShip2 . Shield;
        MessageBox . Show(s, "مقدار خصوصیات سفینه");
    }
}
```



شکل ۲۰-۴- خروی مثال ۴-۴

در این برنامه basicSpaceShip1 براساس سازنده دوم که دارای ۳ ورودی برای مقادیر speed, shield و fuel است ساخته شده ولی basicSpaceShip2 براساس سازنده اول ساخته شده است که ورودی ندارد و مقادیر تمام ویژگی‌ها را ۵۰ قرار می‌دهد (سازنده بدون ورودی را سازنده پیش فرض می‌گویند) و نتیجه کلیک دکمه showButton نمایش پنجره پیام مانند شکل ۲۰-۴ است. (t\معادل کلید tab عمل کرده و چند کارکتر فاصله در پیام قرار می‌دهد.) به قطعه کد زیر دقت کنید.

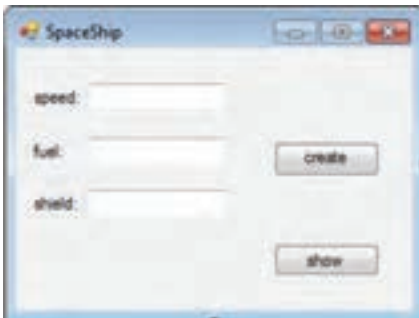
```
basicSpaceShip1 = new SpaceShip(100,80,90);
basicSpaceShip2 = new SpaceShip.SpaceShip(int f, int s, int sh) (+ 1 overload(s))
```

سؤال: چرا در کنار توضیح ظاهر شده به وسیله IntelliSense پیام (Overload +۱) دیده می‌شود؟

دیده می‌شود؟

بهبود و توسعه برنامه

۱- فرمی مانند شکل ۲۱-۴ ایجاد کرده و از سه کادر متنی برای مقداردهی ویژگی‌های سفینه و دکمه CreateButton برای ایجاد یک سفینه استفاده کنید. کاربر تنها اجازه ایجاد یک سفینه را داشته باشد و امکان ورود کارکترهای غیر عددی در کادرهای متنی نباشد.



شکل ۲۱-۴- فرم توسعه و بهبود مثال ۴-۴



۲- سه سفینه بسازید. برای نمایش اطلاعات هر یک از آنها تصویری روی فرم قرار دهید. در صورت کلیک هر یک از تصاویر اطلاعات سفینه نظیر آن در برجسب نمایش داده شود.

شکل ۲۲-۴ فرم نمایش اطلاعات سه سفینه

نکته

بهترین الگو برای شیء گرابی با مفهوم خاص آن در برنامه نویسی و ساز و کار مستقل اشیاء و تعامل آنها با یکدیگر همین جهان پیرامون ماست. خوب به هر شیء و ویژگی‌ها و رفتارهای آن بنگرید، درباره آن فکر کنید و سعی کنید آن را به صورت یک کلاس پیاده‌سازی کنید. تمرین زیاد باعث می‌شود که در هنگام برنامه، آسانتر اشیای موجود در سیستم و ویژگی‌هایشان را تشخیص بدهید و کلاس مربوط به آنها را بنویسید.



شکل ۲۳-۴

کار در کارگاه ۵: مساب پس انداز شفصی

مثال ۵-۴: می‌خواهیم یک حساب شخصی داشته باشیم که بتوانیم به آن مبلغ دلخواه خود را واریز کنیم یا از آن برداشت کنیم و در هر لحظه نیز مبلغ پس انداز ما مشخص باشد.



شکل ۲۴-۴- فرم حساب پس انداز شخصی

الگوریتم یا روش انجام کار: برای داشتن این حساب پس انداز، ابتدا یک کلاس حساب (account) می‌سازیم که دارای عملیات افزایش و کاهش باشد و همچنین مقدار پس انداز را نیز بتوانیم از آن بگیریم. با زدن دکمه واریز، مبلغ وارد شده به حساب اضافه می‌شود و با زدن دکمه برداشت، مبلغ وارد شده از حساب کم می‌شود. باید توجه کرد که تغییر در حساب پس انداز تنها از طریق این دو عملیات، انجام می‌شود.

۱- ایجاد پروژه: ابتدا یک پروژه ایجاد می‌کنیم و فرم ایجاد شده را به صورت زیر مقداردهی می‌نماییم.

جدول ۱۰-۴- ویژگی‌های فرم

Form		
ویژگی	مقدار	
Name	accountForm	
Text	حساب پس انداز من	
Size	width	۴۹۰
	height	۲۶۰

۲- اضافه کردن کنترل‌ها : کنترل‌های فرم را نیز به صورت زیر اضافه می‌کنیم.

جدول ۴-۱۱- ویژگی‌های GroupBox‌ها

GroupBox		
ویژگی	مقدار	مقدار
Name	groupBox۱	groupBox۲
Text	مبلغ پس انداز شده	واریز - برداشت

کنترل برچسب را در groupBox1 اضافه می‌کنیم.

جدول ۴-۱۲- ویژگی‌های برچسب

Label		
ویژگی	مقدار	
Name	savingLbl	
Text		
Background	white	
AutoSize	False	
RightToLeft	Yes	
Font	Size	۱۸
Size	width	۲۰۰
	height	۶۲

یک برچسب، کادر متن و دکمه در group Box2 قرار دهید و ویژگی‌های آنها را مطابق جداول زیر تنظیم کنید.

جدول ۴-۱۳- ویژگی‌های برچسب

Label	
ویژگی	مقدار
Name	label۱
Text	مبلغ :
AutoSize	True
RightToLeft	Yes

جدول ۴-۱۴- ویژگی‌های کادر متن

TextBox	
ویژگی	مقدار
Name	amountTb
Text	
RightToLeft	Yes
width	۱۹۱

جدول ۴-۱۵- ویژگی‌های دکمه

Button		
ویژگی	مقدار	مقدار
Name	addBtn	reduceBtn
Text	وازیز	برداشت

۳- تعریف کلاس و شیء: پس از طراحی واسط کاربری، به سراغ کدها می‌رویم. در ابتدا درون کلاس فرم، کلاس account را برای حساب پس‌انداز ایجاد می‌کنیم و پس از تعریف کلاس، یک شیء از آن می‌سازیم.

```
public partial class accountForm : Form
```

```
{
```

```
    class Account
```

```
    {
```

```
        private uint saving;
```

```
        public uint Saving { get {return saving; } }
```

```
        public void Add(uint amount) { saving += amount; }
```

```
        public void Reduce(uint amount) { if (amount <= saving) saving -= amount; }
```

```
        public Account (uint amount) { saving = amount; }
```

```
    }
```

```
Account myAccount = new Account (0);
```



موارد قابل توجه کلاس Account :

- ✓ فیلد saving برای نگهداری پس‌انداز تعریف شده است.
- ✓ نوع uint فقط اعداد صحیح بزرگ‌تر یا مساوی صفر را می‌پذیرد. دلیل انتخاب این نوع برای مقدار پس‌انداز این است که پس‌انداز منفی قابل قبول نیست.
- ✓ ویژگی Saving فقط متد get دارد بنابراین فقط خواندنی است.
- ✓ متد Add برای اضافه کردن مقداری به پس‌انداز ایجاد شده است.
- ✓ متد Reduce برای برداشت از حساب تعریف شده است.
- ✓ متد سازنده کلاس، یک ورودی دارد تا مبلغ اولیه افتتاح حساب مشخص شود.

همان‌طور که گفته شد برای مقدار حساب پس‌انداز و مقادیر واریزی یا برداشتی از uint استفاده شده است. اما این نوع، مقدار زیادی را در خود جای می‌دهد که برای حساب، استفاده‌ای نمی‌شود.

آیا نوع کوچک‌تری را می‌شناسید که بتوان جایگزین آن کرد؟

در مورد ایجاد حساب myAccount باید گفت که به دلیل نیاز به نگهداری آن در طول برنامه و

استفاده از آن در رویدادهای مختلف، در خود کلاس فرم accountForm و به عنوان یک فیلد سراسری آن تعریف شده است. ضمناً از نوع ایجاد آن مشخص است که حساب دارای مقدار اولیه ° است. پس از فراخوانی متد مقداردهی اولیه فرم (InitializeComponent()) در متد سازنده فرم accountForm، برچسب پس انداز (savingLbl) را مقداردهی می کنیم تا در همان ابتدای ایجاد فرم، مقدار پس انداز مشخص باشد.

```
public accountForm()  
{  
    InitializeComponent();  
    savingLbl.Text = myAccount.Saving.ToString() + "ریال";  
}  
  
نمایش مقدار پس انداز در برچسب  
متد رویدادهای کلیک دو دکمه واریز و برداشت را به صورت زیر اضافه می نمایم.
```

```
private void addBtn_Click(object sender, EventArgs e)  
{  
    myAccount.Add(uint.Parse(amountTb.Text));  
    savingLbl.Text = myAccount.Saving.ToString() + "ریال";  
}
```

```
private void reduceBtn_Click(object sender, EventArgs e)  
{  
    myAccount.Reduce(uint.Parse(amountTb.Text));  
    savingLbl.Text = myAccount.Saving.ToString() + "ریال";  
}
```

در متد رویداد کلیک دکمه واریز، متد Add از شیء myAccount برای اضافه کردن مقدار وارد شده در کادر متنی amountTb اجرا می شود. اما محتوای کادر متنی با متد uint.Parse تبدیل به نوع uint می شود. در خط دوم مقدار پس انداز به وسیله برچسب نمایش داده می شود. در متد رویداد کلیک دکمه برداشت، دستورات مشابه دکمه واریز است با این تفاوت که به جای متد Add از متد Reduce استفاده می شود.

توسعه و بهبود برنامه

در کادر متنی مبلغ، هر مقداری از جمله حروف الفبایی و اعداد منفی را می‌شود وارد کرد که باعث خطا در برنامه می‌شوند. برنامه را طوری تغییر دهید که در صورت ورود مقادیر غیرمجاز، پیام مناسبی از طرف برنامه داده شود.

الف) درستی یا نادرستی عبارات زیر را تعیین کنید :

- ۱- ویژگی، بیانگر عملیات یا رفتار شیء در کلاس است.
- ۲- هر شیء که ساخته می‌شود به میزان مورد نیاز، فضایی از Ram به آن اختصاص داده می‌شود.
- ۳- متدها به مقدار لازم، ورودی دارند.
- ۴- متدها بیش از یک مقدار برگشتی یا خروجی دارند.
- ۵- اگر نوع خروجی متدی Void باشد، متد مقداری را برنمی‌گرداند.
- ۶- برای انتساب مقداری به یک فیلد کلاس، از کلمه کلیدی get استفاده می‌شود.
- ۷- در تعریف کلاس، اگر مقدار فیلدها مقداردهی نشود مترجم خطا می‌گیرد.
- ب) جاهای خالی را با عبارت مناسب پر کنید :
- ۸- در متد، برای برگرداندن مقدار خروجی از کلمه کلیدی استفاده می‌شود.
- ۹- در زبان سی‌شارپ برای ایجاد یک شیء از یک کلاس، از عملگر استفاده می‌شود.
- ۱۰- برای اطلاع از مقدار یک ویژگی در کلاس، در هنگام تعریف آن، کلمه کلیدی به کار برده می‌شود.
- ۱۱- هنگام ایجاد شیء مقدار پیش‌فرض برای فیلدهای رشته‌ای که مقداردهی اولیه نشده است.
- ج) به سؤالات زیر پاسخ دهید :
- ۱۲- تفاوت کلاس و شیء را با مثال بنویسید.
- ۱۳- تفاوت دو توصیف‌کننده public و private را بنویسید.
- ۱۴- متد سازنده چه متدی است؟

تمرینات برنامه‌نویسی فصل چهارم

۱- برنامه‌ای برای نمایش اطلاعات دانش‌آموز بنویسید. برای نوشتن این برنامه کارهای زیر را انجام دهید:

- کلاسی برای دانش‌آموز شامل ویژگی‌های فردی (کد دانش‌آموز - نام - نام خانوادگی - نام پدر) و نمرات ۳ درس دانش‌آموز تعریف کنید.
- سه متد سازنده، تعریف کنید که متد اول مقدار یک نمره، متد دوم مقدار دو نمره و متد سوم مقدار سه نمره را تعیین کند.
- متدی در کلاس قرار دهید که مقدار برگشتی آن، معدل این سه نمره باشد.
- از پرچسب برای نمایش ویژگی‌های فردی، نمره‌ها و معدل و اطلاعات دانش‌آموز بر روی فرم استفاده کنید. برای ایجاد شیء مربوط به هر دانش‌آموز، از یکی از متدهای سازنده استفاده نمایید.



۲- مثال ساعت در متن درس را به صورت زیر تغییر دهید.

- یک شیء Timer به فرم اضافه کرده، ویژگی interval آن را برابر با ۱۰۰۰ قرار دهید تا در هر ثانیه رویداد آن اجرا شود. این timer در ابتدا فعال است.
- متدی به نام AddSecond به کلاس ساعت اضافه نمایید که زمان سپری شده را محاسبه نماید.

راهنمایی: این متد ثانیه‌ها را می‌شمارد و در هر فراخوانی یک ثانیه به مقدار زمان اضافه می‌کند. با رسیدن ثانیه به ۶۰ باید مقدار دقیقه یک واحد اضافه شود و همین‌طور مقدار ساعت نیز در

هنگام رسیدن دقیقه به ۶۰ یک واحد افزوده شود.

○ در رویداد Tick تایمر، متد AddSecond شیء ساعت را فراخوانی کنید.

تحقیق : می توان timer را درون کلاس ساعت ایجاد نمود. مزیت و چگونگی این کار را

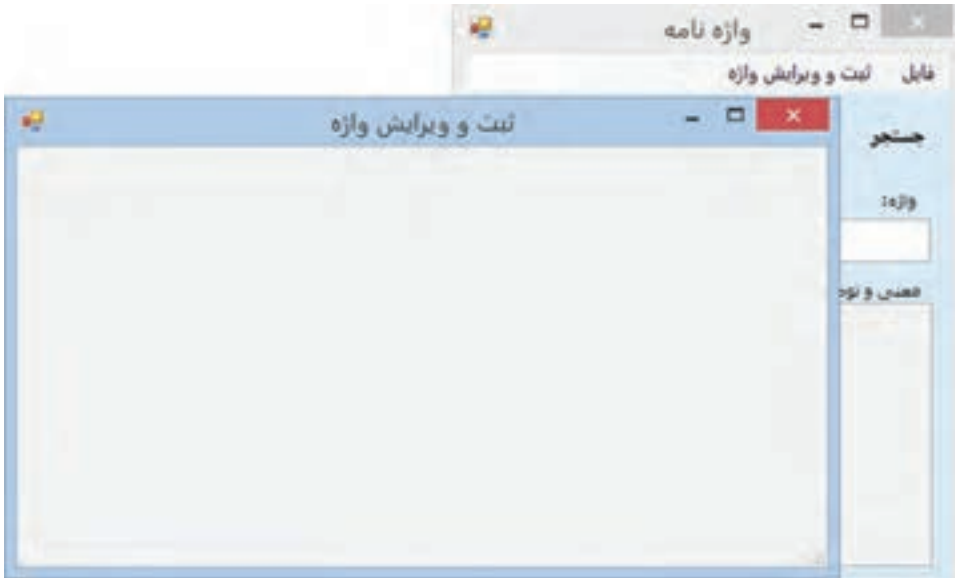
بررسی کنید.

تکمیل پروژه

در این فصل با تعریف کلاس فرم، فضایی فراهم می‌کنیم تا در آن امکان افزودن و ویرایش واژه‌ها صورت گیرد.

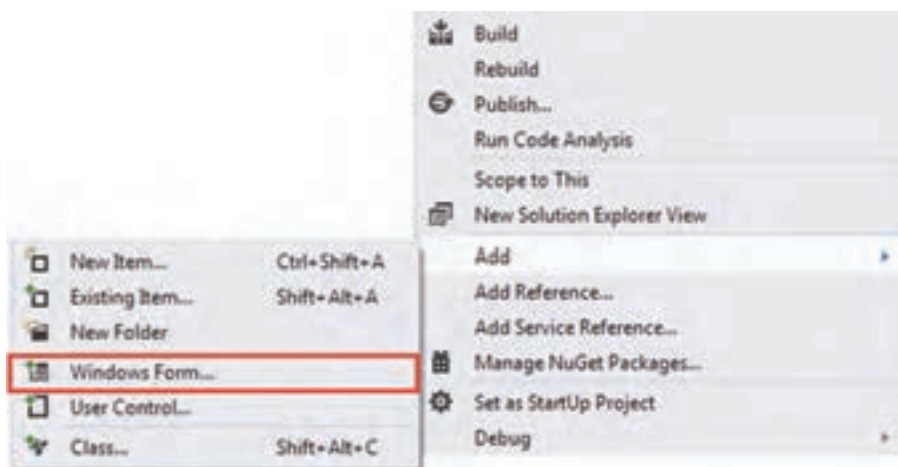
مرحله چهارم:

در این مرحله می‌خواهیم با انتخاب گزینه «ذخیره جستجو» فرمی باز شود تا در مراحل بعدی بتوان از این فرم برای اضافه کردن واژه‌های جدید یا ویرایش واژه‌های موجود از آن کمک گرفت. همچنین می‌خواهیم قسمتی از کد ذخیره جستجو هم در این مرحله نوشته شود.



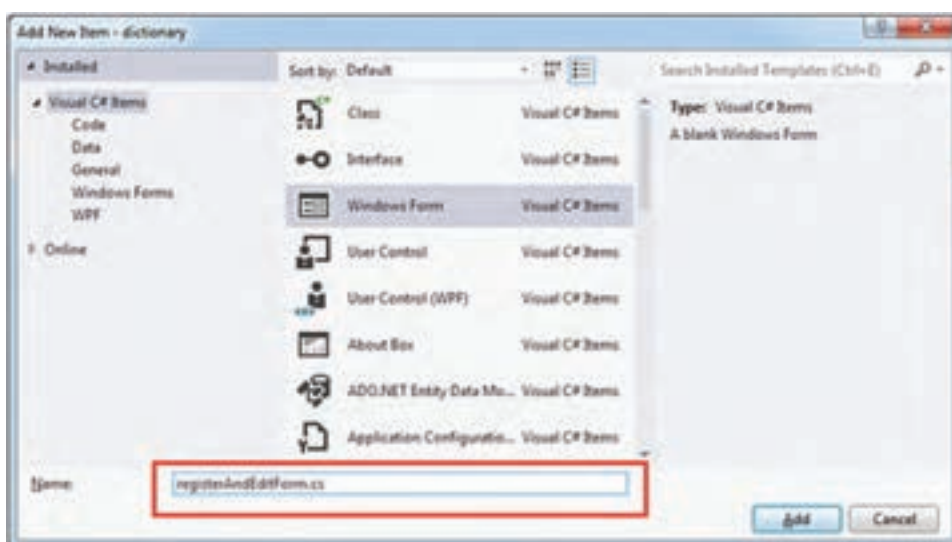
مراحل کار: برای ایجاد فرم جدید باید ابتدا کلاس فرم جدید را تعریف کرد. این کار با استفاده از امکانات ویژوال استودیو بسیار آسان است.

۱- روی نام پروژه کلیک راست نمایید تا منوی آن نمایش داده شود. گزینه Windows Form را از منوی Add کلیک نمایید.



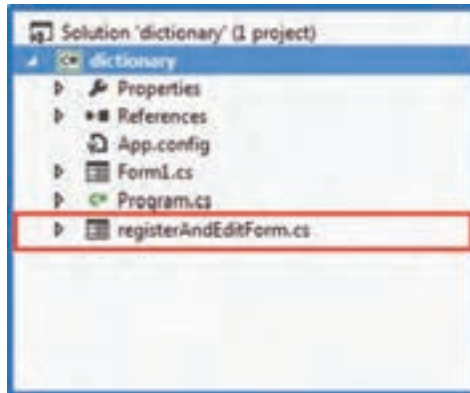
افزودن فرم جدید

۲- با کلیک این گزینه، کادر افزودن بخش جدید باز می‌شود و به صورت خودکار گزینه Windows Form انتخاب شده است.



ذخیره فرم

با وارد کردن نام فرم جدید به صورت `registerAndEditForm.cs` و زدن دکمه Add فرم جدید ایجاد خواهد شد.



ویژگی‌های فرم جدید را به صورت زیر تغییر دهید :

شیء Form		
ویژگی		مقدار
Name		registerAndEditForm
Text		ثبت و ویرایش واژه
Size	Width	519

۳- متد EH رویداد کلیک گزینه "ثبت و ویرایش واژه" در فرم اصلی را به صورت زیر تغییر

دهید .

```
private void registerAndEditMenuItem_Click(object sender,
EventArgs e)
{
    registerAndEditForm regAndEdit = new registerAndEditForm();
    regAndEdit.ShowDialog();
}
```

در خط اول کد داخل متد، یک نمونه (شیء) از کلاس registerAndEditForm ساخته می‌شود و در متغیر regAndEdit قرار می‌گیرد.

با استفاده از متد ShowDialog در خط دوم، فرم ساخته شده (regAndEdit) نمایش داده می‌شود. نحوه نمایش این فرم به صورتی است که مانند یک کادر محاوره ای نمایش داده می‌شود و فرم اصلی قابل دسترسی نخواهد بود

سؤال: بررسی کنید اگر به جای متد ShowDialog از متد Show استفاده کنید چه تفاوتی در نمایش فرم دارد؟

۴- برای گزینه «ذخیره جستجو» در منو باید کادر ذخیره فایل باز شود تا محل و نام فایل ذخیره از کاربر پرسیده شود. در فصل بعد خواهید آموخت چگونه فایل را ذخیره کنید. فایلی که توسط این نرم افزار ذخیره می‌شود حاوی اطلاعات واژه مورد جستجو است. متد EH کلیک گزینه «ذخیره جستجو» را که قبلاً ساخته اید به صورت زیر تغییر دهید.

```
private void saveMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "text file* *.txt";
}
```

در خط اول این متد، یک شیء از کلاس SaveFileDialog ساخته می‌شود و آدرس آن در متغیر sfd قرار داده می‌شود. در خط دوم ویژگی Filter طوری مقداردهی می‌شود که ذخیره فایل با پسوند txt باشد.

واژگان و اصطلاحات انگلیسی فصل چهارم

ردیف	واژه انگلیسی	معنی به فارسی
۱	Account	
۲	Add	
۳	Armor	
۴	Battle	
۵	Cargo Spaceship	
۶	Class	
۷	Constructor	
۸	Fire Rate	
۹	Get	
۱۰	Initialize	
۱۱	Null	
۱۲	Object	
۱۳	Property	
۱۴	Private	
۱۵	Public	
۱۶	Reduce	
۱۷	Reset	
۱۸	Return	
۱۹	Set	
۲۰	Shield	
۲۱	Speed	
۲۲	Storage	
۲۳	Value	
۲۴	Void	