در این مقاله الگوی Observer معرفی خواهد شد و ساختار آن تشریح خواهد شد.

**نقش**

حالتی را تصور کنید که چند شی در برنامه وجود دارد که تغییر حالات یک شی دیگر را دنبال می کنند و با توجه به هر تغییر عملی را انجام می دهند. شاید اولین راه حلی که در این حالت، یافت شود این است که اشیا در یک حلقه شی مورد نظر را بررسی کنند تا اگر تغییری به وجود آمده بود با توجه به آن کاری انجام دهند. اما این بدترین راه حل است و باعث افت کارایی و همچنین طراحی بد برنامه خواهد شد. برای حل این مسئله الگوی Observer معرفی می شود.  
الگوی Observer یک ارتباط بین اشیا ایجاد می کند به طوری که وقتی یکی از اشیا تغییر حالت می دهد همه اشیا مرتبط با آن از این تغییر حالت اطلاع پیدا می کنند. معمولاَ در یک مجموعه از اشیا مرتبط این چنینی، یک منتشر کننده(publisher) و جود دارد و چندین شی وجود دارد که از تغییر حالت شی منتشر کننده آگاه می شوند.

**تشریح**

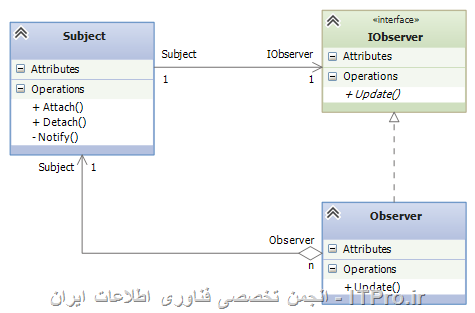
در دنیای کامپیوتر امروزی از این الگو بسیار استفاده شده است. به عنوان مثال حالتی را تصور کنید که در یک سایت آمار فروش به صورت یک نمودار نمایش داده می شود. وقتی که یک عمل فروش انجام می شود این نمودار باید با توجه مقدار حجم فروش ها خود را بروزرسانی کند. حال شاید بتوان گفت که این کار را می توان بدون استفاده از الگویی پیاده سازی کرد ولی وقتی نیاز به الگوی Observer احساس می شود که در زمان اجرا بخواهیم مجموعه اشیائی را که تغییر شی منتشر کننده را دنبال می کنند را تغییر بدهیم، یعنی چند شی به این مجموعه اضافه کنیم و یا از آن کم کنیم. الگوی Observer یک الگوی **رفتاری**است.   
از استفاده های دیگری که از این الگو استفاده شده است مدیریت رویداد های یک برنامه است. به طور مثال وقتی در زبان #C در رویداد **formLoad**متدی را فرخوانی می کنیم وقتی که این رویداد در فرم رخ بدهد فرم همه ی توابع و اعمالی را که برای آن تعریف شده است را آگاه می کند که اجرا شوند.   
  
به طور کلی هرگاه بخواهیم با تغییر حالت یک شی، اشیا دیگر عمل خاصی را انجام دهند از این الگو استفاده می شود. 

**موارد استفاده**

از این الگو در موارد زیر استفاده می شود:   
وقتی که یک abstraction دارای دو جنبه باشد که یک به دیگری وابسته باشد و هر کدام از این جنبه ها در اشیا جداگانه باشند و بتوانند به طور مستقل عمل کنند و تغییرات مورد نیاز خود را روی اشیا خود اعمال کنند.   
وقتی که با به وجود آمدن یک تغییر در یک شی لازم باشد که اشیا دیگر با توجه به تغییر به وجود آمده عمل خاصی انجام دهند و همچنین مشخص نیست که چندتا از اشیا باید عمل مورد نظر را انجام دهند.   
وقتی که لازم باشد که یک شی بتواند به اشیا دیگر بدون توجه به ماهیت آن و یا شناختن آن اطلاع رسانی نماید.

**ساختار**

برای تشریح ساختار این الگو از عبارات زیر استفاده شده است:   
**Subject**: شیئی است که اشیا تغییر حالات آن را دنبال می کنند.   
**Observer**: شی یا اشیایی هستند که تغییر حالات Subject را دنبال می کنند.   
ساختار این الگو به این صورت است که هر شی Observer که بخواهد از تغییر حالت شی Subject آگاه باشد باید خود را در لیست دنبال کنندگان Subject ثبت کند. برای این کار شی Observer متد Attach شی Subject را فراخوانی می کند و خود شی را به عنوان ورودی متد می فرستد و با این کار در ساختمان داده شی Subject ثبت می شود و هر گاه که رویدادی در Subject رخ دهد، این شی متد Notify خود را فراخوانی می کند که با این کار به همه Observerها اطلاع داده می شود. اگر یک Observer نخواهد که دیگر رویدادها را دنبال کند متدل Detach مربوط به Subject را فراخوانی می کند و خود را به عنوان ورودی به آن می فرستد و با این کار از ساختمان داده ی Subjectحذف می شود و دیگر رویداد ها به آن اطلاع داده نخواهد شد. همه ی اشیای Observer اینترفیس IObserver را پیاده سازی می کنند. نمودار Class diagram این الگو به این شکل است.



**مدل های Push و Pull**

برای ارتباط بین Subject و observer دو مکانیزم وجود دارد. مکانیزم اول که Push نام دارد به این شکل است که Subject وقتی می خواهد به observer اطلاع دهد تغییرات را با تمام جزئیات می فرستد و وقتی که observer این تغییرات را دریافت می کند همه ی اطلاعات را برای این که خود را update کند دارد و دیگر نیاز به اطلاعات بیشر از طرف subject نیست.   
مکانیزم دوم که pull نام دارد این گونه عمل می کند که subject پیامی شبیه به آگهی به همه ی observer ها می فرستد و هر کدام از observer ها که بخواهد خود را آپدیت در خواست خود را به subject می فرستد تا اطلاعات مورد نیاز را بدست آورد.   
الگوی observer از هر دو مدل گفته شده پشتیبانی می کند.   
از مزایای این الگو می توان موارد زیر را نام برد:   
این الگو باعث کم کردن وابستگی observer و subject به یکدیگر می شود و در اصطلاح این دو موجودیت در برنامه tightly couple نخواهند بود و بنابراین می توان این دو موجودیت را در لایه های مختلف برنامه تعریف کرد و از این الگو استفاده نمود.   
این الگو از ارتباط broadcast پشتیبانی می کند.

**برخی از مسائل پیاده سازی الگوی Observer**

در ادامه به برخی از مسائلی که ممکن است در هنگام استفاده و پیاده سازی از این الگو پیش بیاید پرداخته می شود. 

* همانظور که گفته شد هر subject باید لیست هر کدام از observer ها را داشته باشد و برای این کار در پیاده سازی یک ارجاع به شی مورد نظر را ذخیره کند. به منظور ذخیره این ارجاع ها باید ساختمان داده ای در نظر گرفته شود که سرعت بالایی داشته باشد که در مواقعی که تعداد observer ها زیاد است باعث کندی برنامه نشود.
* دنبال کردن بیش از یک subject توسط observer: برای این که یک observer بتواند بیش از یک subject را دنبال کند باید متد update خود را به گونه ای تعریف کند که با هر دوی subject ها سازگاری و همخوانی داشته باشد. برای subject راحت ترین کار این است که در فراخوانی متد update خود را به عنوان پارامتر به سمت observer بفرستد و خود observer تشخیص دهد که از طرف کدام یک از subject ها پیام را دریافت کرده است.
* وقتی که عمل حذف یک subject انجام می شود، این عمل نباید باعث به وجود آمدن ارجاع های بی فایده (Dangling reference) شود. همچنین باید ساز و کاری اندیشیده شود که وقتی که یک observer حذف می شود نیز این مشکل پیش نیاید. برای حل این مشکل در طرف observer باید هنگامی که observer در حال از بین رفتن است خود را از لیست observer های یک subject حذف کند، به این شکل که در Destructor کلاس خود متد Detach مربوط به subject را فراخوانی نماید.
* می توان رویدادهایی را که در subject به وجود می آیند را گروه بندی کرده و این قابلیت را به observer ها بدهیم که هنگام ثبت خود به عنوان observer مشخص کنند که قصد دنبال کردن کدام گروه را در subject دارند.
* به علت پیچیدگی هایی که گفته شد ممکن است برای برقراری ارتباط بین observer و subject یک شی جدید به نام ChangeManager داشته باشیم که همه ی پیچیدگی های ارتباط را به این شی محول کنیم. که با این کار مدیریت مسائل گفته شده بسیار آسان تر خواهد بود. البته این کلاس ChangeManger می تواند یک نمونه از الگوی Mediaor باشد و به طور طبیعی باید تنها یک ChangeManager بین یک مجموعه observer و subject موجود باشد که برای این کار می توان از الگوی Singleton استفاده کرد.