مقدمه

برنامه‌نویسی مبتنی بر اشیا (**Object-Oriented Programming** یا **OOP**) یکی از الگوهای اصلی برنامه‌نویسی است که به توسعه‌دهندگان این امکان را می‌دهد تا نرم‌افزارهای پیچیده و قابل نگهداری‌تری طراحی کنند. این الگو بر اساس مفهوم "اشیا" بنا شده است که می‌توانند شامل داده‌ها و توابع (روش‌ها) باشند.

اصول برنامه‌نویسی مبتنی بر اشیا

برنامه‌نویسی مبتنی بر اشیا بر چهار اصل اصلی استوار است:

کپسوله‌سازی (*Encapsulation*): این اصل به معنای پنهان‌سازی جزئیات داخلی اشیا و ارائه یک رابط مشخص برای تعامل با آن‌هاست. این کار باعث می‌شود که کدهای پیچیده‌تر قابل مدیریت‌تر شوند و تغییرات در یک قسمت از برنامه تأثیری بر سایر قسمت‌ها نگذارد.

وراثت (*Inheritance*): این ویژگی به برنامه‌نویسان اجازه می‌دهد تا یک کلاس جدید بر اساس یک کلاس موجود ایجاد کنند. این امر به اشتراک‌گذاری ویژگی‌ها و رفتارها کمک کرده و موجب کاهش تکرار کد می‌شود.

پولیمورفیسم (*Polymorphism*): این اصل به اشیا اجازه می‌دهد تا به شیوه‌های مختلفی رفتار کنند. با استفاده از این ویژگی، می‌توان متدهای مشابه را در کلاس‌های مختلف تعریف کرد و هر کلاس می‌تواند پیاده‌سازی خاص خود را داشته باشد.

انتزاع (*Abstraction*): این اصل به توسعه‌دهندگان این امکان را می‌دهد که تنها جزئیات مهم و ضروری را از اشیا نمایش دهند و جزئیات غیرضروری را پنهان کنند. این کار باعث ساده‌تر شدن طراحی و فهم برنامه‌ها می‌شود.

مزایای برنامه‌نویسی مبتنی بر اشیا

قابلیت نگهداری

با استفاده از کپسوله‌سازی و وراثت، برنامه‌ها به راحتی قابل تغییر و به‌روزرسانی هستند.

استفاده مجدد از کد

با وراثت، می‌توان کدهای موجود را دوباره استفاده کرد و از نوشتن کدهای تکراری جلوگیری کرد.

مدل‌سازی طبیعی‌تر

: برنامه‌نویسی مبتنی بر اشیا به توسعه‌دهندگان این امکان را می‌دهد که دنیای واقعی را بهتر مدل‌سازی کنند.

قابلیت همکاری

اشیا می‌توانند به راحتی با یکدیگر تعامل داشته باشند و این امر به توسعه نرم‌افزارهای پیچیده کمک می‌کند.

برنامه نویسی ساختاری

برنامه‌نویسی ساختاری یک رویکرد به برنامه‌نویسی است که بر اساس تقسیم برنامه به واحدهای منطقی کوچکتر و قابل مدیریت، مانند توابع و ماژول‌ها، بنا شده است. این روش بر استفاده از کنترل‌های منطقی مانند حلقه‌ها، شرط‌ها و توابع تأکید دارد و سعی می‌کند تا از پیچیدگی‌های غیرضروری جلوگیری کند. هدف اصلی برنامه‌نویسی ساختاری افزایش خوانایی، قابلیت نگهداری و تست‌پذیری کد است. با استفاده از این روش، برنامه‌نویس می‌تواند کدهای خود را سازماندهی کرده و از تکرار کد جلوگیری کند، که در نهایت به بهبود کارایی و کاهش خطاها منجر می‌شود.

تفاوت برنامه نویسی ساختاری و شی گرا

برنامه‌نویسی ساختاری و برنامه‌نویسی شی‌گرا دو رویکرد متفاوت در توسعه نرم‌افزار هستند. برنامه‌نویسی ساختاری بر اساس تقسیم برنامه به توابع و واحدهای منطقی تمرکز دارد و سعی می‌کند با استفاده از کنترل‌های منطقی مانند حلقه‌ها و شرط‌ها، جریان کنترل برنامه را مدیریت کند. در مقابل، برنامه‌نویسی شی‌گرا بر اساس مفاهیم اشیاء و کلاس‌ها بنا شده است و اجازه می‌دهد که داده‌ها و رفتارها در قالب اشیاء گروه‌بندی شوند. این رویکرد به برنامه‌نویسان این امکان را می‌دهد که از ویژگی‌هایی چون وراثت، کپسوله‌سازی و پلی‌مورفیسم استفاده کنند، که به سازماندهی بهتر کد، افزایش قابلیت بازاستفاده و توسعه نرم‌افزارهای پیچیده‌تر کمک می‌کند. به‌طور خلاصه، در حالی که برنامه‌نویسی ساختاری بر ساختار و کنترل جریان تمرکز دارد، برنامه‌نویسی شی‌گرا بر مدل‌سازی دنیای واقعی و تعامل اشیاء تأکید می‌کند.

مفهوم کلاس در برنامه نویسی

در برنامه‌نویسی شی‌گرا، کلاس یک الگو یا طرح‌واره برای ایجاد اشیاء است که می‌تواند شامل داده‌ها (ویژگی‌ها) و متدها (عملکردها) باشد. به عبارت دیگر، کلاس مانند یک الگو عمل می‌کند که مشخصات و رفتارهای مشترک اشیاء را تعریف می‌کند. به عنوان مثال، اگر یک کلاس به نام "ماشین" داشته باشیم، می‌توانیم ویژگی‌هایی مانند "رنگ"، "مدل" و "سرعت" را برای آن تعریف کنیم و متدهایی مانند "حرکت" و "توقف" را به آن اضافه کنیم. سپس با استفاده از این کلاس، می‌توانیم اشیاء مختلفی از نوع ماشین بسازیم، هر یک با ویژگی‌ها و وضعیت‌های خاص خود، در حالی که از متدهای تعریف شده در کلاس استفاده می‌کنند.

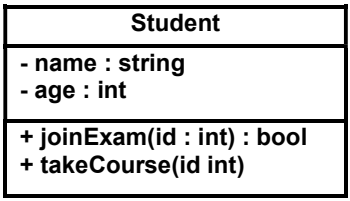
مدل سازی با UML

UML (Unified Modeling Language) یک زبان استاندارد برای مدل‌سازی سیستم‌های نرم‌افزاری است که به توسعه‌دهندگان و تحلیل‌گران کمک می‌کند تا طراحی و ساختار سیستم‌ها را به صورت بصری نمایش دهند. **UML** شامل مجموعه‌ای از نمودارها است که هر یک جنبه‌های مختلف سیستم را توصیف می‌کنند، از جمله نمودارهای کلاس، نمودارهای توالی، نمودارهای فعالیت و نمودارهای مورد استفاده. این نمودارها می‌توانند به درک بهتر تعاملات بین اجزاء سیستم، جریان کار، و ساختار داده‌ها کمک کنند. استفاده از **UML** به تیم‌های توسعه امکان می‌دهد تا ارتباطات بهتری داشته باشند و مستندات دقیق‌تری برای سیستم‌های پیچیده ایجاد کنند، که در نتیجه به بهبود فرآیند طراحی و کاهش خطاهای احتمالی کمک می‌کند.

مدل ساری کلاس

برای این کار خصوصیات و رفتارهای کلاس مشخص شده و در دو بخش جداسازی می شود + نشان دهنده **public** و – نشان دهنده **private** می باشد.

مثال : کلاس دانشجو را با استفاده از **UML** نمایش دهید.



توضیح : در این مثال نام و سن دانشجو بصورت **private** تعریف شده و دو متد **joinExam** و **takeCourse** بصورت **public** تعریف شده اند.

پیاده سازی کلاس در سی شارپ

مثال : کلاس دانشجو را در سی شارپ پیاده سازی کنید.

Student.cs > ...

```
class Student{
    private string name="Farhad";
    private double mark=19.5;
    private int age=19;
    public string GetName(){
        return  this.name;}
    public double GetMark(){
        return this.mark;}
}
```

توضیح : این کلاس دارای خصوصیت نام از نوع رشته ، نمره از نوع عدد اعشاری و سن از نوع عدد صحیح می باشد، دو متد **GetName,GetMark** برای بازگرداندن مقادیر ذخیره شده در **name,mark** که بصورت **private** هستند استفاده شده است.

متدها سازنده

متدهای سازنده (**Constructors**) در زبان برنامه‌نویسی سی‌شارپ، نوع خاصی از متدها هستند که به منظور ایجاد و راه‌اندازی نمونه‌های یک کلاس مورد استفاده قرار می‌گیرند. این متدها به طور خودکار هنگام ایجاد یک شیء از کلاس فراخوانی می‌شوند و معمولاً برای تنظیم مقادیر اولیه اعضای کلاس و انجام هرگونه عملیاتی که برای آماده‌سازی شیء لازم است، به کار می‌روند. متدهای سازنده دارای نامی مشابه نام کلاس هستند و می‌توانند پارامترهایی را برای تخصیص مقادیر اولیه به اعضای کلاس دریافت کنند. در صورت عدم تعریف هیچ متد سازنده‌ای، سی‌شارپ به‌طور خودکار یک سازنده پیش‌فرض بدون پارامتر ایجاد می‌کند. همچنین، امکان تعریف سازنده‌های چندگانه با پارامترهای مختلف نیز وجود دارد که به توسعه‌دهندگان این امکان را می‌دهد که شیء را به روش‌های مختلف ایجاد کنند.

پیاده سازی متد سازنده در سی شارپ

Student.cs > ...

```
class Student{
    private string name;
    public Student(){  }
    public Student(string newname){
        this.name=newname;
    }
}
```

همانطور که دیده می شود کلاس **Student** دارای دو متد سازنده است که یکی بدون پارامتر بوده و متد سازنده دوم رشته ای با نام **newname** را بعنوان پارامتر گرفته و خصویت **name** را مقدار دهی می کند.

متد مخرب

متد مخرب (**Destructor**) در زبان برنامه‌نویسی سی‌شارپ، نوع خاصی از متد است که به منظور آزادسازی منابع و انجام عملیات تمیزکاری هنگام از بین رفتن یک شیء (**object**) استفاده می‌شود. متد مخرب به طور خودکار زمانی فراخوانی می‌شود که شیء از حافظه حذف می‌شود، به‌ویژه زمانی که **Garbage Collector** (جمع‌کننده زباله) در حال پاک‌سازی اشیاء غیرقابل دسترسی است. متد مخرب در سی‌شارپ با نام کلاس، به همراه علامت "~" (ت **tilde**) تعریف می‌شود. درون این متد، می‌توان کدهایی برای آزادسازی منابع غیرمدیریتی مانند فایل‌ها، اتصالات شبکه یا پایگاه‌داده‌ها قرار داد.

یک نکته مهم در مورد متدهای مخرب این است که در سی‌شارپ معمولاً نیازی به استفاده از آن‌ها نیست، زیرا **Garbage Collector** به‌طور خودکار منابع را مدیریت می‌کند. با این حال، در مواردی که منابع غیرمدیریتی وجود دارند، استفاده از متد مخرب می‌تواند مفید باشد.

پیاده سازی متد مخرب در سی شارپ

Student.cs > ...

```
class Student{
    ~Student(){
        // Disconnect connections
    }
}
```