به نام خدا

آموزش سوم

در این قسمت قصد داریم نگاه دقیق تری به مفاهیم شئی گرایی که در زبان های برنامه نویسی متعددی من جمله جاوا مورد استفاده قرار می گیرند داشته باشیم.

در حقیقت شئی گرایی یک زبان برنامه نویسی نیست بلکه عبارت است از یکسری پارادایم ها و ایده هایی که توسط زبان های برنامه نویسی بسیاری مورد استفاده قرار می گیرد می شود که زبان جاوا هم یکی از آن زبان ها است. پایه و اساس تفکر OOP این بود که نگاه برنامه نویسان به برنامه نویسی به نگاه ایشان به دنیای ملموس و واقعی نزدیک تر گردد. به طور مثال کامپیوتری که شما با استفاده از آن به خواندن این مطالب آموزشی مشغول هستید یک Object یا شئی می باشد. حال این Object که در اختیار شما است با کامپیوتری که زیر دست بنده می باشد بسیار متفاوت است.

کامپیوتر شما یک شیئی است که می توانند چندین شئی دیگر را در خود جای دهد مثل هارد، سی پی یو، رم و ... . کامپیوتر شما دارای یکسری Attribute یا خصیصه می باشد که آن را از دیگر کامپیوتر ها مجزا می سازد مثلاً دارای سی پی یو دو هسته ای می باشد. در عین حال کامپیوتر شما دارای یکسری Behavior یا عملکرد نیز می باشد(در زبان انگلیسی این واژه به رفتار ترجمه می شود اما به منظور انتقال بهتر مفهوم معنای دیگری در اینجا در نظر گرفته شده است). به طور مثال کامپیوتر شما می تواند یک فایل صوتی را پخش کند که این Behavior مختص یک شیئ از نوع کامپیوتر است اما دیگر اشیاء در دنیای واقعی مثل یخچال دارای چنین Behavior نمی باشند. در واقع در برنامه نویسی شیئی گرایی، Object ها داری سه ویژگی می باشند: اول اینکه هر

Object دارای یک Identity یا هویت خاص خود است(به طور مثال دو انسان که به منزله Object هستند را در نظر بگیرید. درست است که هر دو دارای یکسری خصایص مثل داشتن دست و پا و قدرت تفکر و غیره می باشند و در عین حال عملکردهای مشابهی نیز دارند مثل حرف زدن، دویدن و ... اما دارای دو هویت مجزا می باشند مثلاً احسان و نیما.) ویژگی دوم مربوط به Attribute یا خصیصه می باشد. مثلاً قد احسان ۱۸۵ سانتی متر است اما قد نیما ۱۷۶ سانتی متر است. ویژگی سوم مربوط به Behavior شیئی است. مثلاً احسان می تواند به خوبی پیانو بنوازد اما نیما نقاش خوبی است.

حال این Object هایی که ما می سازیم می بایست ریشه در جایی داشته باشند. به عبارت دیگر Object ها به خودی خود ایجاد نمی شوند از اینرو ما با مفهومی به نام Class آشنا می شویم. در حقیقت در برنامه نویسی شیئ گرایی ما نمی توانیم از Object ها صحبت به میان آوریم بدون این که توجهی به Class ها داشته باشیم. به طور خیلی خلاصه می توان گفت که Class ها برای ساخت Object ها مورد استفاده قرار می گیرند. همانطور که در مقدمه آموزش اشاره شد، Class همانند یک نقشه ساختمان است که از روی آن می توانیم هزاران هزار خانه بسازیم. در حقیقت ما در Class تعریف می کنیم که Object ما دارای چه Attribute و Behavior باشد. به عبارتی هر Object به منزله یک Instance یا نمونه ای از یک Class است. در برنامه نویسی به این کار اصطلاحاً Instantiation یا نمونه سازی گفته می شود. ما در برنامه نویسی شیئ گرایی برای ایجاد اشیاء اول نیاز به برخورداری از Class داریم اما نکته ای که این جا می بایست مد نظر قرار دهیم این است ما به عنوان برنامه نویس الزماً مجبور نیستیم که کلیه کلاس ها را خودمان بنویسیم چرا که بسیاری از کلاس ها از قبل در دل برنامه جای داده شده اند و ما به راحتی با وارد کردن Class مد نظر می توانیم از قابلیت های آن

استفاده کنیم. اما چنانچه بخوایم Class خود را بنویسیم می بایست همواره چهار نکته را مد نظر داشته باشیم که در ذیل به آن ها اشاره خواهیم کرد:

در ایجاد یک کلاس جدید ما به عنوان برنامه نویس می بایست چهار مورد Abstraction, Polymorphism, Inheritance, Encapsulation را مد نظر قرار دهیم. به منظور به خاطر سپردن این چهار عنصر اصلی ساخت Class می توان حرف اول هر یک از چهار کلمه را گرفته و کلمه A PIE به معنی یک کلوچه را به خاطر سپرد. یک از ویژگی های OOP به کار گیری Abstraction است. در اینجا بنده ترجمه این واژه را در قالب یک مثال عرض می کنم. به طور مثال زمانیکه شما از دوست خود خواهش می کنید که لیوان آبی که روی میز است را به شما بدهد، شما فقط نمود خارجی میز را مد نظر دارید و هرگز از دوست خود نمی خواهید که لیوان آبی که روی میز قهوه ای رنگ با عرض دو متر و طول سه متر و ارتفاع یک و نیم متر را به شما بدهد بلکه صرفاً مفهوم کلی میز مد نظر شما است. در برنامه نویسی هم دقیقا همین طور است. در حین نوشتن کلاس ها دقیقاً ما می بایست یک مفهوم کلی را در نظر بگیریم. به طور مثال فرض کنیم که برنامه ای برای یک باشگاه بدنسازی می نویسیم. ما نیاز داریم تا یک Class ایجاد کنیم که معرف یکسری عناصر مثل نام، نام خانوادگی، تاریخ شروع دوره، میزان شهریه پرداختی باشد. در حقیقت در حین نوشتن Class ما به همین اکتفا کرده و به هیچ وجه در Class خود نام تک تک ورزشکاران را نخواهیم آورد. به طور خلاصه ما در ایجاد Class فقط و فقط یک مفهوم کلی را در نظر گرفته سپس Object هایی را به صورت Customized شده از روی آن Class ایجاد می کنیم.

تاکنون برای خیلی از ما پیش آمده که سرما خورده ایم. به پزشک مراجعه می کنیم و دارو دریافت می کنیم. ممکن است برخی دارو ها در قالب کپسول باشند. در حقیقت

وظیفه کپسول نگهداری داروی داخل آن و محافظت از آن است. در برنامه نویسی شئی گرایی هم وظیفه Encapsulation نیز همین می باشد که Attribute ها و Behavior های موجود در یک Object را در کنار یک دیگر نگه دارد اما این در حالی است که موضوع به همین مسئله خلاصه نمی شود. در حقیقت وظیفه Encapsulation کمی فراتر از این هم هست. مفهوم Encapsulation این امکان را به ما می دهد تا از خصایص یک Object هر آنچه را که ما تمایل داریم نمایش داده شود، در معرض دید دیگر بخش های برنامه قرار گیرند و به عبارتی دیگر بخش های برنامه ما فقط به بخش هایی از یک Object دسترسی خواهند داشت که ما آن ها را پنهان نکرده ایم . سوالی که در اینجا ممکن است برای برخی دوستان پیش آید این است که چه لزومی دارد که ما چیزی را در Class خود ایجاد کنیم سپس خودمان را از دسترسی به آن منع کنیم. در حقیقت ما با این کار می خواهیم وابستگی مابین بخش های مختلف برنامه را به حداقل برسانیم به نحوی که ایجاد یک تغییر کوچک در بخشی از برنامه منجر به تخریب دیگر بخش ها نگردد. حال سوال دیگری که ممکن است برای برنامه نویسان مبتدی پیش آید این است که چقدر ما می بایست بخش هایی از Class را پنهان سازیم و پاسخی که می توان داد این است که هرچه بیشتر بهتر. در حقیقت ما برای اینکه در برنامه های نسبتاً بزرگ با سردرگمی کمتری مواجه شویم نیاز داریم تا فقط بخشی از Class را در معرض دید دیگر بخش های برنامه قرار دهیم که ضروری است.

Inheritance یا وراثت این امکان را در برنامه نویسی شیئ گرایی به ما می دهد تا به جای اینکه یک Class را از اول بنویسیم، شرایطی را فراهم کنیم تا برخی ویژگی های Class جدیدی که می خواهیم ایجاد کنیم را از Class دیگری به ارث ببرد. در مقدمه آموزش مثال زدیم که در آپارتمان ده طبقه ما یک Class برای کلیه طبقات وجود دارد. حال اگر بخواهیم که برخی طبقات ما سه خوابه یا چهار خوابه باشند به هیچ وجه نیازی

نیست که یک Class از پایه بنویسیم بلکه به سهولت می توانیم یک Class با خصوصیت سه یا چهارخوابه بنویسیم که دیگر خصوصیات خود را از Class اصلی یا Superclass به ارث ببرد. حال چنانچه ما تغییری در Superclass ایجاد کنیم، تغییر ایجاد شده در Class های سه خوابه و چهار خوابه نیز اعمال خواهد شد. نکته ای که در اینجا می بایست مد نظر قرار دهیم این است که در زبان برنامه نویسی جاوا ما فقط می توانیم از یک Class اصلی چیزی را به ارث ببریم.

اصطلاح Polymorphism به معنی چند فرمی است. برای روش شدن این مطلب به ذکر مثالی اکتفا می کنیم. به طور مثال حیوان سگ را در نظر بگیریم. این حیوان چنانچه داده ای از جنس بوی گربه به حس بویایی اش منتقل شود پارس می کند. چنانچه داده ای از جنس گوشت به حس بویایی اش منتقل شود بزاق دهانش ترشح می شود و چنانچه داده ای از جنس بوی صاحبش به حس بویایی اش منتقل شود دم تکان می دهد. در هر سه حالت این حس بویایی سگ است که فعالیت می کند و تنها تفاوت در نوع داده ای است که به حس بویایی سگ منتقل می شود. در زبان برنامه نویسی جاوا علامت + دقیقاً چنین ویژگی دارا است. چنانچه ما دو متغییر از جنس عددی را با علامت + جمع کنیم حاصل جمع آن دو عدد را به ما خواهد داد. مثلاً 12=5+7 اما اگر دو متغییر از جنس String یا کلمه را با یکدیگر جمع کنیم آن دو کلمه را در کنار یکدیگر قرار خواهند گرفت مثل Hello+World=HelloWorld

پس از صحبت پیرامون اصول برنامه نویسی شیئی گرایی، امیدواریم که مفهوم OOP را به خوبی درک کرده باشید. حال نوبت به آشنایی با انواع Error ها و Comment ها در زبان برنامه نویسی جاوا می رسد که در آموزش آتی به طور مفصل مورد بررسی قرار خواهد گرفت.