نام و نام خانوادگی : محمد آقائی به نام خدا شماره دانشجویی : ۰۱۲۲۱۰۳۳۷۲۰۰۰۵

تحقیقات درس برنامه نویسی سمت سرور: GC- Destructors -constructor-YANGI-Kiss-Dry-Solid

Garbage Collection (GC)

به طور کلی، Garbage Collection (GC) چیزی نیست جز به دست آوردن مجدد حافظه اختصاص داده شده به اشیایی که در حال حاضر در هیچ بخشی از برنامه ما استفاده نمی شوند.

هنگامی که یک شی در سی شارپ ایجاد می کنیم،یک حافظه برای شی در حافظه heap تخصیص داده می شود. حافظه heap به طور کامل توسط Common Language Runtime (CLR) در چارچوب دات نت مدیریت می شود. تخصیص حافظه و توزیع در heap توسط CLR انجام می شود. همیشه برای هر چیزی محدودیتی وجود دارد، در چنین مواردی حافظه نیز محدود است. ما باید مقداری حافظه را در heap پاک کنیم تا CLR بتواند حافظه را به اشیاء تازه ایجاد شده اختصاص دهد .

Garbage Collection چه زمانی اتفاق می افتد؟

Garbage Collection در صورتی اتفاق می افتد که حداقل یکی از شرایط زیر برآورده شود. این شرایط به شرح زیر است:

اگر سیستم حافظه فیزیکی پایینی دارد، Garbage Collection ضروری است.

اگر حافظه تخصیص داده شده به اشیاء مختلف در حافظه heap از یک آستانه از پیش تعیین شده فراتر رود، Garbage Collection اتفاق می افتد.

اگر متد GC.Collect فراخوانی شود، Garbage Collection اتفاق می افتد. با این حال، این روش تنها در شرایط غیرعادی فراخوانی می شود، زیرا به طور معمول Garbage Collection به طور خودکار انجام می شود.

فازهای Garbage Collection

در Garbage Collection عمدتاً ۳ فاز وجود دارد. جزئیات در مورد این موارد به شرح زیر است:

فاز علامت گذاری: لیستی از تمام اشیاء زنده در طول مرحله علامت گذاری ایجاد می شود. این کار با دنبال کردن رفرنس ها از تمام اشیاء روت انجام می شود. تمام اشیایی که در لیست اشیاء زنده نیستند به طور بالقوه از حافظه heap حذف می شوند.

فاز جابجایی : رفرنس های همه اشیایی که در لیست تمام اشیاء زنده بودند در مرحله جابجایی به روز می شوند تا به مکان جدیدی که در مرحله فشرده سازی اشیاء به آنجا منتقل می شوند اشاره کنند.

فاز فشرده سازی: heap در مرحله فشرده سازی فشرده می شود، زیرا فضای اشغال شده توسط اشیا مرده آزاد می شود و اشیا زنده باقی مانده جابجا می شوند. تمام اشیاء زنده که پس از Garbage Collection یا جمع آوری زباله باقی می مانند به ترتیب اصلی خود به سمت انتهای قدیمی حافظه heap منتقل می شوند.

نام و نام خانوادگی : محمد آقائی　　　　به نام خدا　　　　شماره دانشجویی : ۰۱۲۲۱۰۳۳۷۲۰۰۰۵

تحقیقات درس برنامه نویسی سمت سرور: GC- Destructors -constructor-YANGI-Kiss-Dry-Solid

Destructors – Constructor

Constructor : سازنده کلاس یک متد از نوع Public می باشد که دقیقا هم نام با نام کلاس می باشد. این متد دارای خروجی نمی باشد ولی میتواند ورودی داشته باشد. ورودی این متد در واقع همان پارامترها یا ورودی کلاس می باشد. در زمان ساختن نمونه از کلاس ها باید ورودی ها را مقدار دهی کنیم.

Destructors :

وقتی شما یک نمونه از یک کلاس را ایجاد می کنید در همان لحظه اتوماتیک سازنده کلاس اجرا می شود. حالا وقتی آبجکت می خواهد از حافظه خارج شود مخرب کلاس به صورت اتوماتیک اجرا می شود و دستورات درون آن اجرا می شود. برای تعریف مخرب کلاس می توانید به صورت زیر عمل کنید.

[illegible]

KISS “Keep It Simple, Stupid”

افراد دیگری که بخواهند کد شما را مورد ارزیابی قرار دهند در آینده با استقبال بیشتری این کار را انجام میدهند. اصل KISS توسط Kelly Johnson پایه گذاری شده است و سیستم های خوب به جای پیچیدگی، به سمت ساده سازی پیش میروند.از این رو سادگی، کلید طلایی طراحی است و باید از پیچیدگی های غیرضروری دوری کرد.

YAGNI “You Aren’t Gonna Need It”

گاهی اوقات تیم های توسعه و برنامه نویسان در مسیر پروژه تمرکز خود را بر روی قابلیت های اضافه ی پروژه که "فقط الان به آن نیاز دارند" یا "در نهایت به آن نیاز پیدا میکنند" میگذارند.در یک کلام :اشتباه است!در اکثر مواقع شما به آن نیاز پیدا ندارید و نخواهید داشت. "شما به آن نیازی ندارید"

اصل YAGNI قبل از کدنویسی بی انتها و بر پایه ی مفهموم "آیا ساده ترین چیزی است که می تواند احتمالا کار کند" قرار دارد.حتی اگر YAGNI را جزوی از کدنویسی بی انتها بدانیم، بر روی تمام روش ها و فرآیند های توسعه قابل اجرا است.با پیاده سازی ایده‌ی "شما به آن نیازی ندارید" میتوان از هدر رفتن وقت جلوگیری کرد و تنها رو به جلو و در مسیر پروژه پیش رفت.

هر زمان اضطراب ناشناخته ای در کد حس کردید نشانه ی یک امکان اضافی بدون مصرف در این زمان است.احتمالا شما فکر میکنید یک زمانی این امکان اضافی را نیاز دارید.آرامش خود را حفظ کنید! و تنها به کارهای موردنیاز پروژه در این لحظه نگاه کنید. شما نمیتوانید زمان خود را صرف بررسی آن امکان اضافی کنید چون در نهایت مجبور به تغییر، حذف یا احتمالا پذیرفتن هستید ولی در نهایت جزو امکانات اصلی محصول شما نیست.

نام و نام خانوادگی : محمد آقائی به نام خدا شماره دانشجویی : ۰۱۲۲۱۰۳۳۷۲۰۰۰۵

تحقیقات درس برنامه نویسی سمت سرور: GC- Destructors -constructor-YANGI-Kiss-Dry-Solid

DRY“Don’t Repeat Yourself”

تا الان چندین بار به کد های تکراری در پروژه برخورد کرده اید؟ اصل DRY توسط Andrew Hunt و David Thomas در کتاب The Pragmatic Programmer پایه گذاری ده است.خلاصه ی این کتاب به این موضوع اشاره میکنید که "هر بخش از دانش شما در پروژه باید یک مرجع معتبر، یکپارچه و منحصربفرد داشته باشد". به عبارت دیگر شما باید سعی کنید رفتار سیستم را در یک بخش از کد مدیریت کنید.

از سوی دیگر زمانی که از اصل DRY پیروی نمیکنید، در حقیقت اصل WET که به معنای Write Everything Twiceیا We Enjoy Typing دامن گیر شما شده است! (لذت بردن از وقت تلف کردن)

استفاده از اصل DRY در برنامه نویسی بسیار کارآمد است. مخصوصا در پروژه های بزرگ که کد دائما در حال نگهداری و توسعه است

SOLID

یکی از مشکلاتی که طراحی نامناسب برنامه های شی گرا برای برنامه نویسان ایجاد می کند موضوع مدیریت وابستگی در اجزای برنامه می باشد. اگر این وابستگی به درستی مدیریت نشود مشکلاتی شبیه موارد زیر در برنامه ایجاد می شوند:

برنامه ی نوشته شده را نمی توان تغییر داد و یا قابلیت جدید اضافه کرد. دلیل آن هم این است که با ایجاد تغییر در قسمتی از برنامه، این تغییر به صورت آبشاری در بقیه ی قسمت ها منتشر می شود و مجبور خواهیم بود که قسمت های زیادی از برنامه را تغییر دهیم. یعنی برنامه به یک برنامه ی ثابت و غیر پیشرفت تبدیل می شود. (این مشکل را Rigidity می نامیم.)

تغییر دادن برنامه مشکل است و آن هم به این دلیل که با ایجاد تغییر در یک قسمت از برنامه، قسمت های دیگر برنامه از کار می افتند و دچار مشکل می شوند. (این مشکل را Fragility می نامیم.(

قابلیت استفاده مجدد از اجزای برنامه وجود ندارد. در واقع، قسمت های مجدد برنامه ی شی گرای شما آنچنان به هم وابستگی تو در تو دارند که به هیچ وجه نمی توانید یک قسمت را جدا کرده و در برنامه ی دیگری استفاده کنید. (این مشکل را Immobility می نامیم.(

اصول SOLID که قصد رفع کردن این مشکلات و بسیاری مسائل گوناگون را دارد عبارت اند از:

- Single Responsibility Principle اصل مسئولیت واحد
- Open-Closed Principle اصل باز-بسته
- Liskov Substitution Principle اصل جایگزینی لیسکوف
- Interface Segregation Principle اصل جداسازی رابط
- Dependency Inversion Principle اصل وارونگی وابستگی

نام و نام خانوادگی : محمد آقائی به نام خدا شماره دانشجویی : ۰۱۲۲۱۰۳۳۷۲۰۰۰۵

تحقیقات درس برنامه نویسی سمت سرور: GC- Destructors -constructor-YANGI-Kiss-Dry-Solid

S مخفف Single responsibility principle یا SRP به معنی اینکه هر کلاس بایستی فقط یک کار انجام دهد نه بیشتر,

O مخفف Open/closed principle یا OCP به معنی اینکه کلاس ها جوری نوشته بشن که قابل گسترش باشند اما نیاز به تغییر نداشته باشند.

L مخفف Liskov Substitution Principle یا LSP به مفهوم اینکه هر کلاسی که از کلاس دیگر ارث بری میکند هرگز نباید رفتار کلاس والد را تغییر دهد.

I مخفف Interface Segregation Principle با ISP به مفهوم اینکه چند اینترفیس کوچک و خورد شده همیشه بهتر از یک اینترفیس کلی و بزرگ است.

D مخفف Dependency inversion principle یا DIP به معنی اینکه از اینترفیس ها به خوبی استفاده کن!