

گاهنامه علمی خبری عصر رایانه

خرداد ۹۷ - شماره بیستم

#خبرهای-خوبی-در-راه_است



اعوان
مشاوران صنعت نرم افزاری



ابحثون علمی کامپیوتر



رهنمای کامیابان هشتبین

جاواکاپ برگزار می‌کند:
مسابقه

برنامه نویسی
جاوا

JCAL

هر ماه یک مسابقه

www.javacup.ir



گاہنامه علمی خبری

عصر رایانه

سال دهم، شماره ۲۰
خرداد ۱۳۹۷ - ۳۴ صفحه



صاحب امتیاز: انجمن علمی کامپیوتر و رباتیک

مدیر مسئول: تانيا طهران‌چی
سردیر: محمد حسین ریماز
صفحه آرایی: محمد عاشورلو

همراهان این شماره:

مهند زمانیان
سجاد جعفری
حسین رحمتی
محمد حسین ریماز
امیرراد کیمیابی
حسن صادقی

فهرست مطالب

سالی که گذشت	صفحه ۱
چگونه یک دانشجوی موفق باشیم	صفحه ۳
رهنمایی کالج	صفحه ۹
رویایی عمیق	صفحه ۱۱
جاوا و ماژولاریتی	صفحه ۱۵
سی‌پلاس‌پلاس	صفحه ۱۷
یونیتی شیدرلب	صفحه ۲۵
الگوی‌های طراحی در مهندسی نرم افزار	صفحه ۲۹

شورای مرکزی انجمن
سال ۱۳۹۷ - ۱۳۹۸

اعضای اصلی:

محمد عاشورلو - دبیر
محمد حسین ریماز
علیرضا قاسمی
تانيا طهران‌چی
مجید رمضان‌خانی

اعضای علی البدل:

محمد علی اردھالی
مهردی کافی

حامیان مالی



اعوان

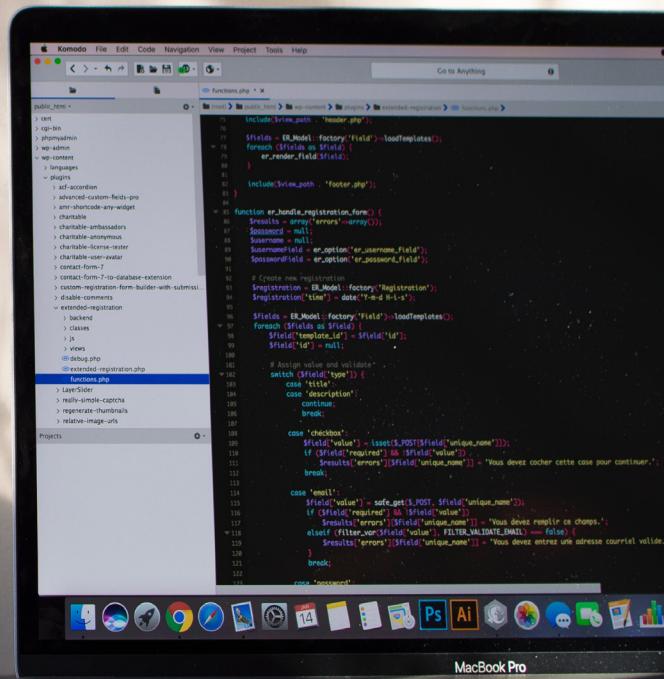
مشاوران صنعت نرم افزاری



RAHNEMA

سالی کے گذشت

محمدحسین ریماز - دبیر سابق انجمن



شود. نه ما آغازگر این شکستن
چرخ باطلیم و نه آخرین نسل آن.
نسل بعدی انجمن و ورودی‌های
جدید وارثان این مسیر هستند.
در این یک سال اتفاقات خوب
کم نبود. دوره‌ها و کلاس‌های
زیادی توسط انجمن برگزار شد
که برخی از آنها مخاطبین زیادی
هم داشت.

در این دوره انجمن به مراتب سعی کردیم ارتباط انجمن با شرکت‌های معتبر را به



به ترتیب از راست به چپ: مهدی کافی، حامد خشہ چی، محمد حسین ریماز، سجاد جعفری

نایت دانشجویان
صوولا مشکل آن
انشجو به موجود
ه حضورش فقط
یک و شیرینی و
ذایی هست، تف
سا چرا هنوز زحم
خاپ و آماده ساز
می کشیم؟ چرا زد
لاس هایی را مر
کر می کنیم برای
تواند مفید باشد
جود کوبیده
مدن به این
وج بی محبتی و
توجهی باز هم
یش می رویم؟
ون باور داشتم
ه شکستن
ن چرخه باید
جا ی شروع

داستان این دوره انجمن هم به سر رسید و با انتخابات، افراد و نیروهای جدید سکان هدایت انجمن علمی کامپیووتر را به دست گرفته‌اند. داستان انجمن و داستان این دانشکده، اگر نخواهیم به کل دانشگاه تعمیمش دهیم، چرخه‌ای گویا ابدی شده از آدمهای با ارزشی که برای کار و تلاش می‌آیند و در تلاطم با موج افسردگی و دلزدگی و دلمردگی، خود به همان ورطه دچار می‌شوند. خودم عقیده‌ام این بود که نشریه علمی‌ای که دو خط کد در آن پیدا نشدود اساساً نشریه علمی نیست و سعی کردم به عنوان سردبیر آن را بهبود دهم و سعی در ایجاد تغییر در این جو موجود کنم. شک که ندارم هیچ، یقین دارم که این نشریه آنچنان که باید و شاید مورد توجه و

بهانه‌های مختلف مثل برگزاری دوره، حمایت از نشریه و ... تقویت کنیم. اکنون با شرکت‌های بزرگ و معتبری مثل رهنما، بیسفن، اعوان، سحاب پرداز ارتباط داریم و امیدواریم این مسیر در آینده با حمایت بیشتر دانشکده و دانشگاه ادامه پیدا کند.

دانشجویان دانشکده نیز در این مدت بیکار نبودند! در مسابقات جاواکاپ که در آبان ماه برگزار شد نمایندگان دانشکده موفق به کسب رتبه اول انفرادی و کسب رتبه دوم بین دانشکده‌ای بعد از دانشکده کامپیوتر دانشگاه شهید بهشتی شدند.

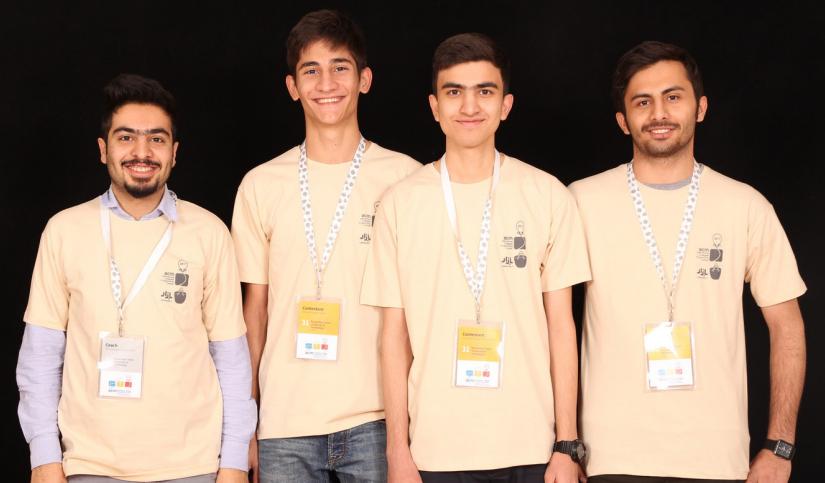
البته افتخار آفرینی بزرگ‌تر کسب ۳ سهمیه در مسابقات منطقه‌ای ACM توسط دانشجویان بود. با حمایت‌ها و پشتیبانی‌های انجمن علمی کامپیوتر، دکتر ناصر شریف، دکتر موسوی نیا و مهندس زمانیان برای اولین بار ۳ تیم از دانشکده توانستند در این مسابقات بدرخشنند. امیدواریم که این روند رو به رشد از سوی دانشجویان و انجمن علمی ادامه دار باشد. #خبر_های_خوبی_در_راه_است.



به ترتیب از راست به چپ: امیر ملکی، تانیا طهرانچی، محمد علی اردھالی، مربی: محمد مقدس.



به ترتیب از راست به چپ: سعید اوکد، سینا عباسی، علی اولیایی.



تیم تمام‌ای! به ترتیب از راست به چپ: پارسا مظاہری، مهدی محمودیان، کیوان دهقان و مربی امیر آهنگری که در اولین حضور در این مسابقات نتیجه بسیار خوبی گرفته‌اند.



حکونه

دانشجوی

موفق باشیم

قسمت اول - محمد حسین ریماز



موفقیت یه چیز فرمول داری نیست که بشه برای همه یه نسخه‌ای پیچید و گفت اگر فلان کار و بهمان کار رو بکنی قطعاً آدم موفق‌تری می‌شی. خیلی چیزا می‌تونه روی موفقیتون تاثیر بزاره که حتی خیلی‌هاش هم دست خودتون نیست؛ اینکه مثلاً چه آدمایی در مسیر زندگیتون قرار بگیر، چه فرصت‌هایی براتون ایجاد بشه یا حتی از دستتون بره. از تاثیر شانس و اقبال هم نمیشه گذشت.

یک سری اشتباه‌ها هست که به عنوان دانشجو اگر مرتکب بشید می‌تونه شما رو خیلی از مسیرتون منحرف کنه، توی این مقاله تجربیاتی که در این چند سال دانشجویی کسب کردم رو با شما خصوصاً ترم اولی‌ها درمیون می‌زارم و نظر یکی از اساتید هم در این باره می‌خوینیم.

اعتماد به نفس بالا

داشتن اعتماد به نفس بالا به خودی خود عیب حساب نمیشه، اما مشکل از جایی شروع میشه که اعتماد به نفس بالای شما که به هر دلیلی می‌تونه به وجود آمده باشد؛ مثلاً چون رنک هستید، رتبه کنکورتون از همه بیشتر بوده، توی چندتا درس ۲۰ گرفتید و از همه بهتر بودید یا رزومه خیلی قوی و سواد بالایی دارید. اعتماد به نفس بالا و غرور می‌تونه جلوی پیشرفت شما رو از دو جهت بگیره. اول اینکه شاید آدم با سوادی باشید اما قطعاً روی کل حوزه‌های مختلف علوم کامپیوتر سواد بالایی ندارید، غرور بیش از اندازه می‌تونه جلوی شما رو از یادگرفتن از دیگران بگیره. اگر به خیال خودتون بهترین برنامه نویس کل دانشگاه باشید احتمالاً دیگه غرورتون اجازه نمیده یه سوال برنامه نویسی از کسی بپرسید.

دوم اینکه اعتماد به نفس بالاتون می‌تونه دیگران رو آزار بده، می‌تونه به اونا این حس رو القا کنه که شما دارید اونها رو تحقیر می‌کنید بنابراین اگه اونا چیزی برای باد دادن به شما داشته باشن احتمالاً میل و رقبتی ندارن که با شما در میون بزارنش. هرکسی، با هر سطح سوادی چیزی برای یاد دادن به شما داره و شما هم با هر سطح سوادی که دارید می‌تونید چیزی برای یاد دادن به دیگران داشته باشید. فروتن باشید و نزارید غرور بی‌جا جلوی پیشرفت و موفقیتون رو بگیره.

سیگار

اگرهر مشکلی توی زندگیتون، روابط عاشقیتون و وضعیت تحصیلیتون دارید دلیل نمیشه یه مشکل دیگه بهش اضافه کنید و به سلامتی تون لطمہ بزنید و با دست خودتون، خودتون رو نابود کنید.

سیگار نکشید. اگر هم دوستی هم دارید که چه تفریحی چه حرفه ای سیگار می‌کشه باهаш دوست نباشید. اگر فیلم رهایی استاد ایرج ملکی رو دیده باشید احتمالاً می‌دونید که همه چی با یه نخ سیگار شروع میشه!

البته جای این نکته خالی نیست که بگم قلیون بدتر از سیگاره...

حل اصلی رو توضیح بدید که انگار روش شک دارید و از استادتون

یا کمک مدرسش میخوايد که غلط بودن روش شما رو توضیح بدە! در این حین احتمالا به اشتباهش پی میبره حتی اگه نبرد هم میتوانید بعد از کلاس خصوصی صحبت کنید و مشکل رو حل کنید، حالا یا شما اشتباه می کنید یا طرف مقابل. توی کتابهای تاریخ از کسی که استادش رو ضایع کرده هیچ کسی یادی نکرده، شاید به نظرتون جالب باشه، یا نشون بده شما چقدر با سوادید، اما کاملا در اشتباهید. یاد بگیرید که چطوری از آدمها نقد کنید و اشتباهاتشون رو بگید بدون اینکه اونها رو ناراحت کرده باشید. دانشگاه رینگ بوکس نیست که بخوايد با همه دعوا بگیرید.

زبان

بهترین موقعیت‌های زندگیتون رو می‌توانید با بلد نبود زبان از دست بدید. بهترین مطالب و به روزترین مسائل روز رو می‌توانید فقط با خاطر اینکه زبان بلد نیستید از دست بدید. استعدادها و قابلیت‌هاتون رو به جای اینکه در دنیای جهانی و شرکت‌های بزرگ به کار ببرید فقط با خاطر زبان توی شرکت‌های معمولی بسوزونید.

اگه زبان بلد نیستید، نمی‌توانید خوب تجربیاتتون رو با دنیا به اشتراک بزارید، نمی‌توانید دانش روز رو مطالعه کنید و با دنیای بیرون ارتباط برقرار کنید. زبان حتی از رشته‌ای که دارید درس می‌خونید و از تمام درس‌هایی که دارید پاس می‌کنید مهم‌تره. خیلی رک و روراست بخواب بگم اگر زبان بلد نباشد حداقل توی رشته کامپیوتر به هیچ موقیتی نمی‌رسید.

دoust

اگر دوستانتون هدف‌ها و دغدغه‌هاشون هم راستی با شما و هدف‌هاتون نباشن شما هم هیچ‌وقت به هدف‌هاتون نمی‌رسید. اگر به جای اینکه بشینید با دوست‌هاتون راجب یه مطلب علمی تبادل نظر کنید، یا شروع کنید به نوشتمن یه پروژه متن باز یا هر فعالیت علمی دیگه برد کنار هم دیگه قلیون بکشید، خیلی بعیده که به هدف‌های بزرگ‌تون برسید. اگه از ۵ نفر دوست صمیمیتون ۴ نفرشون به دغدغه اصلیشون این باشه که چی بپوشن و مهمونی تولدشون کیا رو دعوت کنن و فلان روز در فلان جای دانشکده کی با کی بوده احتمالا به هدف‌هاتون نمی‌رسید و شما هم تبدیل می‌شید به هموна. اگه از ۵ نفر دوست صمیمیتون ۳ نفرشون به فکر این باشن که تکنولوژی روز چیه، فلان الگوریتم چه شکلی پیاده سازی می‌شه و فلان دانشکده دنیا داره روی چه موضوعی کار می‌کنه، احتمالش خیلی بالائه که شما با انرژی بیشتری دنبال هدف‌هاتون بردید. سعی کنید دوست‌هایی داشته باشید که از خودتون بهترن، به شما انرژی میدن و باعث رشد شما می‌شن و شما هم سعی کنید همچین دوستی باشین.

می‌گن آدما میانگین ۵ نفر افراد صمیمی و نزدیکشون هستن، امیدوارم این برايند آدمای دور و برتون و خواسته ها و اهدافشون در راستای

یه مرز باریکی بین دلسوزی برای جامعه و محیطی که تو ش هستیم و بی تفاوتی کامل بهش هست. خیلی‌ها براشون مهم نیست اصن رئیس جمهور قراره کی بشه، با اصن تو دانشگاه چه خبره، در بهترین حالت هیچی نمی‌گن یا فقط احتمالا غر میزند و می‌گن این مملکت جای موندن نیست. یه سری هم که یا چپ چپن یا راست راست. قراره یه بار زندگی کنید، بهتره توی این یه بار فرصت زندگی سر از زندان در نیارید، با دوستانون با خاطر عقاب سیاسیشون دعوا نگیرید و باهم مهربون باشید. پیگیر اخبار باشید، دلسوز باشید اما خودتون رو به پای سیاست نسوزونید.

کار

کار کردن به خودی خود خیلی امر مثبتی هست. اما مشکلی که هست اگه قرار باشه کار جدی بکنید بهتون این یقین رو میدم که چندتا کلاس درس رو احتمالا شرکت نمی‌کنید چون مجبورید ساعت کاریتون رو پر کنید. ممکنه وسط امتحانا درگیر یه پروژه بشید که باید سریع تحولیش بدید و کلی مشکل دیگه که کار کردن می‌تونه به درستون لطفه بزنه. خدا رو شکر ما برنامه نویس‌ها هیچ مشکلی برای پیدا کار کردن نداریم. فقط کافیه سواد لازم رو داشته باشیم. خیلی‌ها می‌گن دانشگاه چیزهایی که به درد بازار کار بخوره رو یاد نمیده. خوب درسته، هیچ جای دنیا دانشگاه چیزی که به درد کار بخوره و مستقیم استفاده بشه رو یاد نمیدن. دانشگاه مباحث پایه‌ای هست که دونستن شرط لازمه. اگر می‌خوايد توانایی‌هایی کسب کنید که به درد بازار کار بخوره لزومی نداره حتما کار بکنید، می‌توانید یه نگاهی به آگهی‌های استخدام بکنید و ببینید چه توانایی‌ها و تکنولوژی‌هایی نیاز هست و شروع کنید به یاد گرفتنشون. اگه با سواد باشید قطعا وقتی فارغ التحصیل شدید برآتون کار هست. اگر نیاز مالی شدیدی ندارید یا کار نکنید، یا توی شرکت‌هایی کار بکنید که با ساعت کاری دانشجو ها مشکلی ندارن که اینجور شرکت‌ها خیلی کم هستن. برای کار کردن همیشه وقت هست تا ۹۰ سالگیتون می‌توانید کار بکنید ولی همیشه حوصله و توان درس خوندن و یاد گرفتن نیست.

رفتار حرفه‌ای

سعی کنید که یاد بگیرید چطوری هوشمندانه‌تر انتقاد کنید، اگر استاد یا دستیار آموزشی‌اش مطلبی رو گفتد که شما مطمئن بر غلط بودنش هستید دلیلی نداره وسط کلاس داد بزنید که داری اشتباه می‌گی. هر آدمی تحمل نقد صریح رو لزوما نداره و ممکنه نقد شما رو شخصی و توهین تلقی کنه. به جای اینکه مستقیم به یه نفر بگید اشتباه کرده، می‌توانید بگید به نظرم می‌تونه به فلان دلیل غلط باشه، یا بگید من فکر می‌کرم این موضوع به فلان شکل حل می‌شده مشکل راه حل من چیه؟ در واقع با روش‌هایی مشکل و راه

میشه همون نگه داشتن پلن های جایگزین. از طرفی درسته که توی دنیای کامپیوتر همه چیز و اعه اما تو دنیای واقعی همه

چی بده بستونه. باید حواسرون باشه که تعادل درستی بین درس

و معدل و توانایی های فنی که خیلی به معدلتون ربط نداره برقرار

کنید. طبعتا توی رشته کامپیوتر اگه همه دروستون هم ۲۰ باشه اما

توانایی فنی نداشته باشید گزینه هاتون فقط به کارهای آکادمیک

محدود میشه و آپشن های دیگه رو ممکنه از دست بدید.

عاشق باشید:

البته منظور عشق الهی یا عشق زمینی نیست. حتی یک لحظه

به این حقیقت شک نکنید که اگر واقعا عاشق کاری که می کنید

نباشید و با تمام وجودتون بهش علاقه نداشته باشید و تمام فکر

و ذکرتون اون کار نباشه، توی اونکار موفق نمیشید. احتمالا خیلی از

شما ندانسته بخاطر جو کنکور و جامعه رشترون رو انتخاب کردید.

اگه رسیدید به ترم ۲ و ۳ و هنوز هم عاشق کاری که می کنید

نشدید، قطعا یه جای کار میلینگه، اگه هنوز بعد دو سال نه با

کد زدن راحتید، نه با الگوریتم میونه خوبی دارید و نه اشتیاقی

برای یادگرفتن چیزی جدید دارید احتمالا مهمترین دلیلش اینه که

واقعا عاشق رشترون نیستید. سعی کنید بیشتر راجب کمک هایی که

رشترون میتونه به انسان ها بکنه مطلع بشید، دونستن کاربردها

میتونه دید شما رو باز کنه، درست کردن چیزهای کاربردی میتونه

شما رو علاقه مندتر کنه. در نهایت ممکنه شما برای اینکار ساخته

نشده باشید، هیچ اشکالی هم نداره اما قراره یکبار زندگی کنید

پس زودتر بزید دنبال علاقترون، حتی اگر شده رشته و دانشگاهتون

رو عوض کنید ولی دنبال چیزی بزید که واقعا عاشقش هستید.

اهداف و خواسته های شما باشه چون اگه اینطور نباشه احتمالا

شما به خواسته ها و اهدافتون نمیرسید.

تک بعدی:

نه توی بازار کار، نه برای اپلای و نه تو زندگی شخصی کسی از آدم

تک بعدی خوش نمیاد. اگه فقط یه برنامه بنویس گیکی باشید

که نه از هنر سر در میاره، نه از موسیقی، نه میدونه اینور آب چه

خبره نه اونور آب و نه چهارتا رمان درست و حسابی خونده احتمالا

موضوع زیادی برای صحبت کردن با آدم های اطرافتون ندارید. سعی

کنید توی حداقل یک رشته ورزشی کار کنید و ورزش کنید احتمالا اگه

قرار باشه کل زندگیتون رو یا روی صندلی باشید یا روی تخت خیلی

زیاد عمر نمی کنید. اما زمانتون رو مدیریت کنید و بین کارهاتون

اعتدال برقرار بکنید، قرار نیست شما یه منتقد سینمایی باشید که

کل فیلم های دهه ۵۰ میلادی رو دیده یا یه آهنگساز درجه یک و

نوازنده حاذق بشید، قراره یه برنامه نویس خوب بشید که سلیقه

خوبی توی فیلم دیدن داره و میتونه آهنگ های فشنگی با گیتارش

بنزه و هیکلش رو روی فرم نگه داره.

معدل:

اینجا دیبرستان نیست که قرار باشه همه درس ها رو با نمرات ۱۸

۱۹ پاس کنید. البته یه سری از دوستان برقی می تونن معدل ۲۰

هم تو دانشگاه بگیرن که جزو استثنای هستن. اگر کسی دم

گوشتون گفت معدل اهمیتی نداره و کسی قرار نیست با معدل به

شما کار بده، شاید توصیه بدی نکرده باشه ولی شما از یه گوش

بشنوید و از یه گوش دیگه در کنید. شنیدید میگن پل های پشت

سرتون رو خراب نکنید؟ یا به قولی همیشه یه پلن B ای داشته

باشید؟ معدل خوب دقیقا مثل اون نقشه جایگزینه، یا بخواه دقیق

تر بگم معدل نقشه اصلیه! با معدل پایین ممکنه در مقاطع بالاتر

به مشکل بخورید یا حتی از یکسری فرصت های تحقیقاتی و علمی

محروم بشید! مثلا ممکنه دوست داشته باشید در یه آزمایشگاه

تحقیقاتی کار کنید اما یکی از شروط اونجا معدل بالا باشه. برای

اپلای کردن هم که اگه معدلتون پایین باشه احتمالا هیچ شانسی

ندارید. معدل خوب منظورمون ۱۸ و ۱۹ نیست، تو دانشگاه معدل ۱۶

خوب حساب میشه و بالای ۱۷ عالیه! البته برای یکسری جاهای معدل

سال های آخر و دروس تخصصی مهم تره بنابراین هیچ موقعی برای

بهتر کردن معدلتون دیر نیست. ممکنه به خودتون بگید من که

قرار نیست تا دکترا بخونم و اپلای بکنم، بازار کار هم که معمولا

معدل خیلی برآشون مهم نیست و بیشتر توانمندی ها برآشون مهمه،

پس چرا خودم رو برای یه معدل بالا اذیت کنم؟ جواب این سوال

چگونه یک دانشجوی

موفق باشیم - قسمت دوم

مهندس مهدی زمانیان

آنچه در نوشتار پیش رو می بینید در پاسخ به درخواستی است با این مضمون که اگر به دوران دانشجویی بر می گشتبید چه کارهایی را انجام می دادید و چه کارهایی را خیر.

ست که می خواهم به آن بپردازم.	گر عمر دوباره‌ای داشتم حتماً از فرصت‌ها به شیوه بهتری استفاده می‌کردم و این توصیه ایست که برای دیگران هم دارم. از فرصت‌هایی که خداوند در اختیارمان می‌گذارد به خوبی استفاده کنیم چرا که فرصتها از دست می‌روند چه خوب استفاده کنیم و چه بد و معلوم نیست آیا فرصت دیگری به این خوبی در اختیارمان باشد یا خیر.
کسب دانش	اینکه در راستای امور دانشجویی و مهندسی کامپیوتر چگونه می‌توان از فرصتها به خوبی استفاده کرد را در سه راستا شرح می‌دهم:
- کسب دانش	- مهارت آموزی
- ایجاد و گسترش دایره روابط اجتماعی	- آماده شدن برای کار با افراد مختلف
کسب دانش	کسب دانش
مهارت آموزی	مهارت آموزی
آماده شدن برای کار با افراد مختلف	آماده شدن برای کار با افراد مختلف

آنچه در نوشتار پیش رو می بینید در پاسخ به درخواستی است با این مضمون که اگر به دوران دانشجویی بر می گشتبد چه کارهایی را انجام می دادید و چه کارهایی را خیر اینکه چه کارهایی می بایست انجام می دادم یا خیر، گستره متنوعی از کارها را شامل می شود که در این ها مجال فرصت پرداختن به آن ها نیست و صرفا به گوشه اندکی از این موارد که مرتبط با بخش دانشجویی و مهندسی است می پردازم. بدیهی است که این موضوع، بخش کوچکی از زندگی فردی و اجتماعی ما را شامل می شود و موارد مهمتری وجود دارد که می بایست در اولویت بالاتر به آن ها پرداخت.

"فرصتها می گذرد چون گذر ابرها،
 فرصتهای نیک را غنیمت شمrid." این فرمایش مولا علی(ع) جانمایه کلامی

برای موفقیت در بازار کامپیوتر است. ولی اگر به پیشرفت در این حوزه می‌اندیشید و می‌خواهید توانایی‌های خود را بهبود دهید و حوزه‌های مختلف را تجربه کنید لازم است توانایی‌های دیگری غیر از برنامه‌نویسی نیز داشته باشد. برای نمونه ممکن است برخی از شما درس‌های ریاضی را نامرتبط با بازار کار بینند ولی واقعیت این است که برای کار در بسیاری از شاخه‌های مهندسی کامپیوتر، داشتن دانش ریاضی و آمار و احتمال شرط لازم است از آن جمله است هوش مصنوعی، یادگیری ماشین، بینایی ماشین، رباتیک و مهندسی داده.

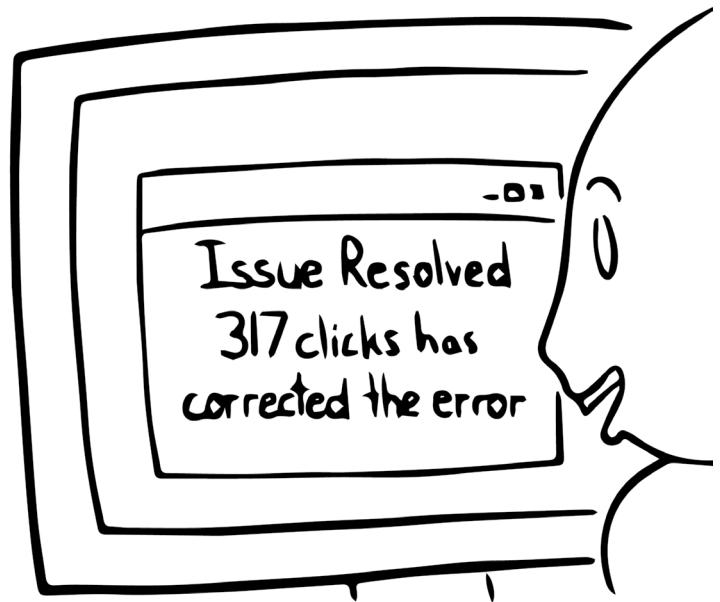
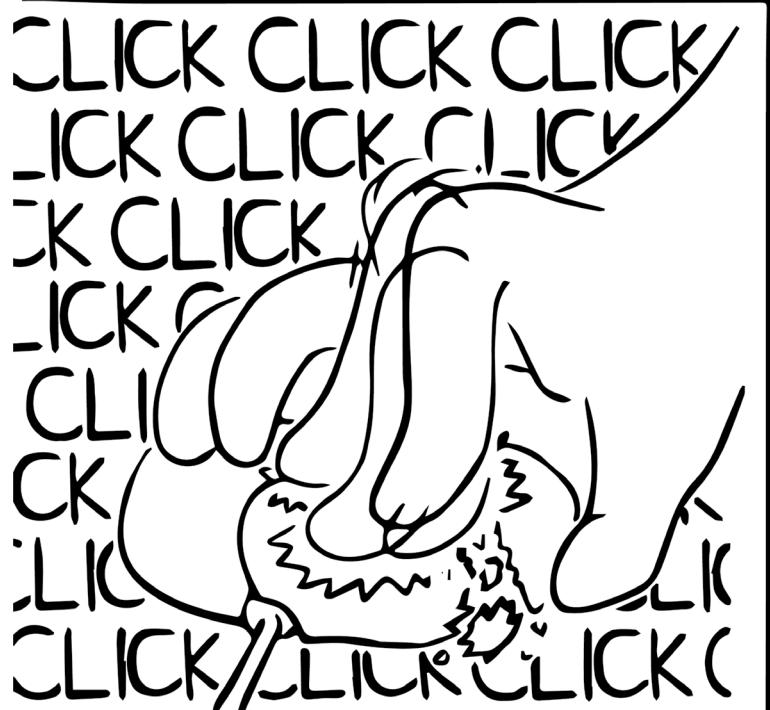
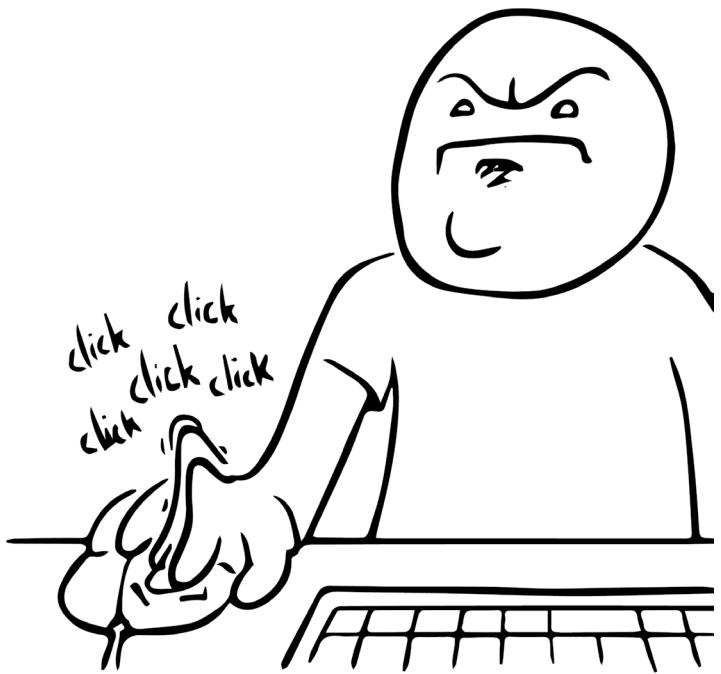
مهارت آموزی

در این دوره از زندگی‌تان تلاش کنید دانش خود را به محک تجربه در آورید و از این رهگذر مهارت‌های لازم را برای کار و پژوهش را به دست آورید. حل مسائل واقعی‌تر را تجربه کنید و از این رهگذر اعتماد به نفس خود را در مواجهه با مسائل پیش رو تقویت کنید. تجربه مسائل واقعی‌تر کاربرد مطالب آموخته شده را به شما نشان می‌دهد و این انگیزه شما را در کسب دانش بیشتر می‌کند. از سوی دیگر به مسائلی بر می‌خورید که با دانش فعلی‌تان جوابه‌ای مناسبی برای آن‌ها ندارید و این موجب می‌شود در پی پاسخ به پرسشهای‌تان در دروس مرتبط باشید و با دید بهتری با درسها مواجه خواهید شد و ضرورت آن‌ها را بیشتر متوجه خواهید شد. کسب مهارت و تجربه کاری به ساختن و بهبود منظومه فکری شما کمک می‌کند چرا که برای مواجهه با مسائل واقعی معمولاً به طیف متنوعی از دانش‌ها نیاز دارید و هر یک از مطالب جایگاه مناسب خود را در ذهن شما می‌یابد. برای به دست آوردن مهارت و تجربه کاری می‌توانید کارهای داوطلبانه در زمینه مهندسی کامپیوتر انجام دهید یا به صورت کارآموزی، پاره وقت یا پروژه‌ای به کار پردازید. مشکلی که در این وادی دیده می‌شود افراط در این موضوع است. باید مراقب باشید که به دست آوردن تجربه کاری به قیمت لطمہ به روند آموزشی‌تان به دست نیاید. بعد از فارغ التحصیلی فرصت کافی برای تجربه کارهای مختلف خواهید داشت ولی دوره آموزشی‌تان تکرار نمی‌شود، این موضوع را به خاطر داشته باشد.

ایجاد و گسترش دایره روابط اجتماعی:

یکی از سرمایه‌های مهمی که در دوران دانشجویی به دست می‌آورید یافتن دوستان و آشنایانی است که می‌توانند در آینده شما تاثیرگذار باشند. تلاش کنید روابط خوبی با دانشجویان، استادان و کارکنان محل‌هایی که در آن‌ها کسب مهارت می‌کنید ایجاد کنید و این روابط را گسترش دهید. گرفتن یک مشاوره کوتاه و مفید یا پیشنهادی سازنده از طرف یک دوست می‌تواند کار شما را بسیار جلو ببرد. خیلی وقت‌ها شما کارها یا پروژه‌های جدید را به واسطه آشنایان خود که مهارت‌های شما را می‌شناسند و به شما اعتماد دارند به دست می‌آورید. تلاش کنید دایره دوستان و آشنایان خود را تحکیم داده و گسترش دهید.

موارد فوق توصیه‌های من بود در این مجال که امیدوارم مورد استفاده شما قرار گیرد.



وقتی برنامه هنگ می‌کنه!



رهنما کالج

استعدا را با ترندھای روز جهان آشنا کند و آنها را بدون سردرگمی در قدم گذاشتن در مسیر درست یاری کند و تجربیات خودش را در اختیار نیروهای مشتاق و با انگیزه قرار دهد.

شرکت‌های مختلف دیگه هم هستن که خودشون دوره های کارآموزی و کارورزی به این شکل برگزار می‌کنن؟ به نظرتون چرا شرکت‌ها دارن همچین کاری می‌کنن؟ واحد کارآموزی دانشگاهی کافی نیست؟ به طور کلی مدتی است که شرکت‌ها به تربیت و پرورش منابع انسانی تمايل پیدا کرده‌اند و ترجیح میدن که نیروی حرفه‌ای برای موقعیت‌های شغلی مورد نیازشون رو درون سازمان آموزش بدن؛ به همین خاطره که مدام شاهد تولد دوره‌های متتنوع کارآموزی هستیم. این کار از چند منظر هم برای شرکت‌ها هم برای کارجوها موثر و مفیده. اول این که صاحبان مشاغل می‌توونن مطابق چارچوب‌ها و فرهنگ سازمانی خودشون منابع انسانی مورد نیازشون رو از ابتداء آموزش بدن و بدون اتلاف انرژی، مهارت‌های دقیق مدنظرشون رو به کارآموز یادن بدن و در اون احساس وفاداری نسبت به سازمان رو ایجاد بکنن. همینطور کارآموز هم می‌توانه با انتخاب فضای کاری مورد علاقه‌ش را در مسیر درستی قرار بگیره و بدون اینکه مجبور باشه از این شاخه

با این همه رهنما کالج محدودیت سنی ندارد و پذیرای افراد مشتاق در هر سن در مسیر درست یاری کند و تجربیات خودش را در اختیار نیروهای مشتاق و با سالی است.

چرا رهنما کالج رو برگزار می‌کنید و هدف برگزار کننده هاش چی هست؟ هدف رهنما کالج در وهله اول توانمند کردن کارجویان و تسهیل مسیر ورودشون به بازار کار است. رهنما کالج این فرصت رو در اختیار شرکت‌کنندگان دوره قرار میده که دانش خودشون رو بسنجند و مهارت‌هایشون رو در بوته‌ی آزمایش بذارن و ببینن که چندمرده حلچاند، اون‌ها در طول دوره می‌توانند مسئولیت‌پذیری‌شون رو تخمین بزنند و با اعضای تیم‌شون یک پروژه مشترک رو به ثمر برسونند. همه‌ی این مراحل طی می‌شن تا کارآموزها خودشون رو به چالش بکشن و طی چند هفته به شکل نمادین یک شمای کلی از کار تیمی و به عهده داشتن مسئولیت بخشی از پروژه کار رو تجربه کنند.

از سوی دیگر هدف رهنما کالج پرورش تفکر کارآفرینی، تقویت قوه‌ی خلاقیت و نوآوری، تشویق کارآموزان به داشتن جسارت خلق ایده‌های نو و ساختن راه حل‌های جدید است.

با توجه به رشد روزافزون کسب و کارهای استارت‌آپی و تغییر ساختار سنتی بازار، شرکت رهنما قصد دارد نیروهای جوان با

شرکت رهنما چیه و کارش چیه؟ یکی از سوالاتی که هیچ‌موقع نمی‌شه بهش جواب قطعی داد، اینه که رهنما چیه و چه کار می‌کنه! به خاطر این که رهنما شرکت نوآور و پویاییه که از ایده‌های تازه استقبال می‌کنه و همیشه در حال فکر کردن به کارهای نو و خلق ارزش‌های جدیده! اما اگه به شکل اجمالی بخوایم به سوالتون جواب بدیم باید بگیم که رهنما توی سه حوزه عمده فعالیت می‌کنه: خدمات ارزش افزوده موبایل، تبلیغات دیجیتال و سرمایه‌گذاری روی ایده‌ها و استارت‌آپ‌ها. ولی یادتون نره که رهنما فقط همین نیست.

رهنما کالج چیه و مناسب چه کسایی هست؟

رهنما کالج، بخش آموزشی شرکت رهنما است که عهده‌دار آماده سازی کارآموزان و توانمند ساختن آن‌ها برای ورود به بازار کار است. مخاطب این دوره دانشجویان یا فارغ‌التحصیلانی هستند که تمايل دارند در رشته‌ی مورد نظرشان حرفه‌ای بشوند، روی پروژه‌های واقعی کار کنند و تجربه‌ی کار تیمی کسب کنند. آن‌ها طی یک دوره‌ی چند هفته‌ای با متدورهای حرفه‌ای آموزش می‌بینند و ضمن آن در کارگاه‌های اخلاق حرفه‌ای و مهارت‌های نرم شرکت می‌کنند تا بتوانند برای ورود به بازار کار و تعاملات جدی آماده شوند.

آدم‌های مختلفی با خصوصیات خاص وارد رهنما میشون، چقدر رهنما کالج روی این افراد و توانمندی‌هاشون تاثیر گذار بوده؟

کاری که رهنما کالج برای کارآموزها میکنند اینه که اونا رو در فضای واقعی کار قرار میده؛ بچه ها وقتی که در تیم قرار می‌گیرن و تقسیم مسئولیت میکنن هر کدام به اندازه‌ی خودشون وظیفه به انجام رسوندن بخشی از کار رو تقبل می‌کنن و در اون فرایند یاد می‌گیرن که چطور با هم دیگه همکاری پیشبرنده داشته باشن. در این بین هر کسی برای به ثمر رسوندن پروره اگر اخلاق ناسازگاری هم داره کنار میگذاره، فرد گرایی از بین میره و همه برای تحقق بخشیدن به یه هدف مشترک به یک مای بزرگ می‌پیوندند. درست مثل سنگ های کف رودخونه که در اثر هم‌حواری با هم صیقلی میشون و زاویه‌های تیزشون از بین میره!

می‌افته. امکانی که کمتر در محیط‌های آکادمیک مهیا میشه. کارآموزایی که دوره‌ی رهنما کالج رو طی می‌کن در کنار منتورهای حرفه‌ای با جزییات کار و فوت و فن تحويل پروره آشنا میشون و در کنار همه‌ی مباحث آموزش مرتبط با موضوع دوره، در کارگاه‌های تعاملی مهارت‌های محیط کار هم شرکت می‌کنن و نحوه‌ی درست ارتباطات در محیط کاری رو یاد می‌گیرند.

آیا بعد از رهنما کالج میشه توی رهنما موند و پیشنهادهای کاری داشت؟ در صورتی که کارآموزها دوره رو با موفقیت پشت سر بگذارن و در تحويل پروره هم خوب عمل بکنن و تمایل داشته باشن که کار کنن، رهنما برای مصاحبه دعوتشون می‌کنه و در صورتی که موقعیت شغلی مناسب با تخصص و علاقه‌شون موجود نباشه، با موافقت خودشون به سایر شرکتها معرفی میشون تا بتونن مناسبترین شغل با روحیات و استعدادشون رو پیدا کنن.

به اون شاخه بپره و وارد حوزه‌های بی ارتباط به تخصص و علاقمندیش بشه، در راه درستی که اون رو به بهترین جایگاه شغلی هدایت میکنه قرار بگیره. تفاوتی که در دوره های کارآموزی شرکت‌ها و دانشگاه وجود داره در عملگرایانه بودن اون هاست. به این معنی که یه کارآموز بعد از طی کردن دوره‌ی آموزشیش در یک شرکت به احتمال خیلی زیاد آمادگی به عهده گرفتن مسئولیت‌های شغلی رو داره و چون در فضای واقعی کار قرار گرفته توانایی تعامل و برقرار کردن ارتباط حرفه‌ای در فضای کار رو داره، این اتفاق در فضای دانشجویی و آزمایشگاهی دانشگاه‌ها به ندرت رخ میده.

تو رهنما کالج چه چیزایی میشه یادگرفت که با کلاس و دانشگاه نمیشه یادشون گرفت؟

آموزش‌های رهنما کالج محدود به مباحث تئوری و درسی صرف نیست؛ بلکه همون طور که پیشتر گفتیم به کسب تجربه‌های عملی در خلال کار تیمی و تحويل پروره‌های واقعی اتفاق

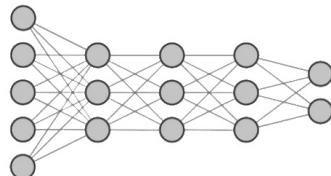
رویای عمیق - Deep Dream

حسین رحمتی

بلافضلله درک می‌کنیم که این شئ مربوط به چه چیزی است. این کار برای مغز در مدت زمان بسیار کوتاهی اتفاق می‌افتد، ولی باید درک کنید که همین فرآیند برای کامپیوتر را اعم از تصویر یا صدا به مفاهیم

مصنوعیست. پیشینه‌ی این روش به قرن ۱۹ میلادی برمی‌گردد. زمانی که Cajal عصب‌شناسی مشهور به نام اینکه چه فعل و افعالی دست به برای اولین بار ساختار سلول‌های مغزی یک پرنده را مورد بررسی قرار دست هم می‌دهند تا مغز ما چیزی داد. چیزی که این فرد مشاهده کرد بسیار پیچیده‌تر خواهد بود.

برای انجام این کار (تشخیص) بین پیکسل‌های تصویر و مفهوم مرتبط با آن، یک شبکه‌ای از نورون‌های متصل به هم ایجاد می‌شود که می‌تواند در کو ر تکس‌های بینایی بدن



BIRD

باشد و یا امروزه به صورت مصنوعی با استفاده از مدل‌های شبکه‌های عصبی مصنوعی در داخل کامپیوترها. در این شبکه، پیکسل‌ها از لایه‌های ابتدایی تا لایه‌های انتهایی با استفاده از سیناپس‌های مرتبط‌کننده نورون‌ها

شبکه‌ای به مانند شبکه‌های الکترونیکی بود که نورون‌های مختلف را به هم وصل کرده بودند. این نورون‌ها مانند یک‌سری عناصر محاسباتی بودند که اطلاعات را به هم پاس می‌دادند. هرگاه که ما شئ‌ای را می‌بینیم،

آدم‌ها از سال‌های خیلی دور تا به امروز همیشه سعی داشته‌اند که رموز عمل‌کردی مغز را پیدا کنند. برای اولین بار ساختار سلول‌های مغزی یک پرنده را مورد بررسی قرار دست هم می‌دهند تا مغز ما چیزی تبدیل می‌کند و تشخیص می‌دهد یا چگونه مفاهیم متعدد را به هم می‌آمیزد و چیزی را خلق کند.

در چند قرن اخیر به لطف پیشرفت‌های علوم کامپیوترا، ریاضی و هوش مصنوعی و با استفاده از تکنیک‌های مختلف امکان تشخیص (recognition) برای کامپیوترا به طور قابل توجهی فراهم شد فراهم شد. یکی از این تکنیک‌ها که امروزه بیش از قبل توجه افراد را به خود جلب کرده، استفاده از شبکه‌های عصبی

معمولاً از مجموعه تصاویر و شبکه آموزش دیده برای تشخیص حیوانات برای ایجاد مدل های deep dream استفاده می شود، ولی می توان از هر مجموعه و شبکه دیگری نیز استفاده کرد.

در زیر نمونه ای از تصاویری که با استفاده از این روش



ایجاد شده اند را می بینیم:

با ترکیب این پروسه و زوم کردن در تصویر خروجی و خروجی گرفتن دوباره از شبکه عصبی به تکرار پیاپی آن می توان ویدئوهایی بوجود آورد که در هر مرحله شبکه با استفاده از خروجی مرحله قبل به generate کردن آیتم ها می پردازد.

این فرآیند فقط به تکنولوژی های تصویری منحصر نمی شود، بلکه میتوان به صورت ترکیبی از آن استفاده کرد. مثلاً محققی با استفاده از یک دوربین عکاسی و شبکه عصبی آموزش دیده روی مجموعه های شعری قرن ۲۰^م از روی محتوای تصویری خروجی دوربین، شعر generate می کند.

می شوند تا در انوها مجموعه ای آن پیکسل ها را به مفهوم آن شئ مرتبط کنند. در این مدل ما سه متغیر اساسی خواهیم داشت، پیکسل های ورودی (X)، وزن ها W ها و مفهوم خروجی از شبکه (Y). می توان رابطه ای این متغیرها را به صورت ساده شده ریاضی به شکل $Y = W^* X$ مدل کرد. البته عملیات انجام گرفته بسیار پیچیده است که برای ساده سازی مفهوم به شکل $Y = W^* X$ بیان شده است.

می دانیم که در هر معادله ای سه متغیره را با در اختیار داشتن دو تا از متغیرها، متغیر سوم را می توان به دست آورد. در مدل تشخیص شئ، ما با استفاده از X ها و W ها، Y را که بیانگر مفهوم شئ بود، بدست می آورديم (شاید این سؤال پیش بیاید که W ها از کجا حاصل می شوند. می دانیم که عملیات W^* یک عملیات پیچیده است و امکان وارونگی ندارد، ولی میتوانیم با بازی کردن با قوانین جبری، W ها با تخمین از رابطه $Y = W^* X$ و مینیمایز کردن E ها بدست آورد که خارج از بحث مورد نظر ماست).

حدود ۲ سال پیش محققان حاضر در تیم Machine Intelligence گوگل این مساله را مطرح کردند که اگر

رابطه فوق را به صورتی باز نویسی کنیم که W و Y مولفه هایی باشند که در اختیار داریم و X را بخواهیم بدست آوریم چه حاصل می شود. برای مثال ماشین میداند که جسم مورد نظر "پرنده" است و می داند با چه ساختار شبکه ای و چه وزن هایی این "پرنده" حاصل شده (شبکه آموزش دیده) و حال قصد ایجاد تصویری برای این موضوع دارد. با استفاده از همان پروسه مینیمایز کردن Error ها که بالاتر توضیح داده شد، می توان تصویر را بوجود آورد (Creation) و به نتایج جالب و قابل تأملی دست پیدا کرد.

شاید برای شما سؤال باشد که چرا تصاویر خروجی به حالتی هستند چندین point of view در یک مجموعه کنار هم قرار گرفته اند، دلیل این اتفاق این است که شبکه طوری طراحی شده تا از ابهامات دوری کند. مثلاً اینکه یک صورت در یک ژست باشد یا در یک ژست دیگر. همین باعث می شود که اگر در هنگام بازسازی، تصویری به عنوان راهنمای شبکه ندهیم، تصاویر به صورت در هم برهم به نمایش می آیند.

Java 9 - Jigsaw

محمد حسین ریماز

پس از مدت ها انتظار و تأخیر های بسیار نسخه جدید جاوا، یعنی جاوا ۹، به صورت رسمی معرفی شد. در مقاله گذشته در مورد یکی از قابلیت های جدید جاوا ۹ یعنی `Shell` صحبت کردیم. در این مقاله به صورت ساده، انقلابی ترین و جنجالی ترین اتفاقی که در اکوسیستم جاوا رخ داده، یعنی پروژه `Jigsaw` را بررسی خواهیم کرد.

منطقی جدا کننده ای بین آنها برقرار نمی شود. می توان سناریوهایی را تصور کرد که از یک کلاس با چند ورژن مختلف در این `jar` فایل ها موجود باشند. مشکلی که به وجود خواهد آمد این است که در هنگام اجرای برنامه `Java Class` که در `Loader` تنها یک ورژن از این کلاس ها را می تواند بارگذاری کند و این می تواند باعث ابهام در نحوه کار کردن برنامه و حتی ایجاد خطأ در حین اجرای برنامه شود و شما سرگردان برای پیدا کردن مشکل باشید. در حقیقت این سناریو آنقدر تکرار شده که حتی نام مخصوص به خود را دارد! جهنم `jar` فایل ها یا `JAR` که می توانید در سایت ^۱`DZone` در مورد آن بخوانید.

اما `classpath` یک مشکل دیگر هم دارد. یک اصلی در طراحی سامانه ها وجود دارد که می گوید اگر قرار است خراب شوی، همان اول خراب شو. ممکن است در محیط توسعه شما، کتابخانه ها و کلاس هایی وجود داشته باشند که به هر دلیلی این کتابخانه ها در محیط اجرایی وجود نداشته باشند. از آنجایی که تا زمانی که از کلاس استفاده نشده باشد، در حافظه

کاهش دهیم. پس کدهای خود را خوانا می نویسیم، اسم کلاس ها را قابل فهم انتخاب می کنیم و کدهای مرتبط را درون پکیج های مشخص قرار می دهیم تا آنها را از هم جدا کنیم. همه این کارها برای خواناتر شدن کدها و مشخص بودن وظایف و هدف هریک از این اجزا است. اما اگر سیستم بزرگی داشته باشیم که ۱۰۰ ها پکیج در کنار هم باشند آن وقت چی؟ احتمالا باید با ابزارهای مختلف، `Dependency` بین این پکیج ها را پیدا کنیم و بفهمیم آنها چه ارتباطی با یکدیگر دارند. به نظر چندان راه حل خوانایی نیست. هدف ما خوانایی بالاتر است، حتی اگر به قیمت چند خط کد اضافه زدن و انتخاب اسامی طولانی تر باشد. برای همین پکیج ها و مکانیزم فعلی چندان کارا به نظر نمی رسد. ما مشکلی که در جاوا وجود دارد، `classpath` و نحوه اجرای برنامه ها است. در حال حاضر تمامی کتابخانه ها و `jar` فایل هایی که استفاده کرده اید، اطلاعات مربوط به کلاس ها و ... در درون یک فایل `classpath` گنده قرار می گیرد و دیگر هیچ سطح

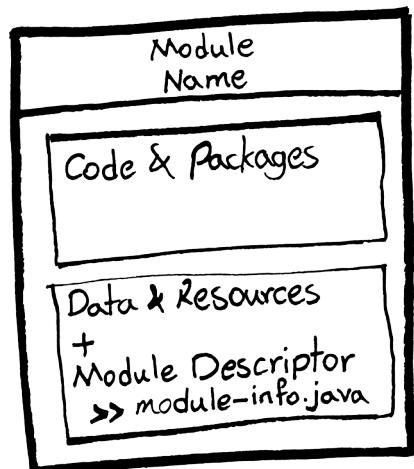
اگر در این چند ماهه اخبار مرتبط با جاوا ۹ را دنبال کرده باشید، حرف و حدیث های زیادی در مورد مازولاریتی (`Modularity`، بستر مازولار جاوا (`Java Module Platform System`) و مازول ها (`Module`) شنیده اید. اما اصلا همه این جار و جنجال ها برای چیست؟ چرا مازولار شدن `JDK` انقدر اتفاق مهمی است؟ مازولاریتی چه کمکی به ما می کند و به چه دردی می خورد؟ چرا به مازولاریتی احتیاج داریم و چگونه می توانیم پروژه مازولار خودمان را بسازیم؟ احتمالا بخش زیادی از این پرسش ها با خواندن این مقاله رفع خواهد شد و قدمی برای حرکت به سوی جاوا ۹ برای شما خواهد بود.

چیستی و چرايی: نگهداری (Maintainability) یکی از مهم ترین مواردی است که در فرایند توسعه و رشد یک سیستم نرم افزاری نقش ایفا می کند. ما برنامه هایی مقیاس پذیر، خوانا با قابلیت استفاده مجدد و قابل فهم می خواهیم تا بتوانیم هزینه نگهداری سیستم های نرم افزاری خود را

ماژول java.base را داریم که تمامی ماژول‌های دیگر خواه و ناخواه به آن وابسته هستند. این گراف یک خاصیت جالب دیگر نیز دارد. احتمالاً شنیده‌اید که Circular Dependency یا وابستگی حلقه‌ی^۳ طراحی مناسبی نیست و می‌تواند در دسرزا باشد. در واقع این گراف، یک گراف جهت‌دار بدون دور یا DAG است و شما نمی‌توانید بین ماژول‌هایتان وابستگی حلقه‌ی داشته باشید.

عکس زیر در ساده‌ترین شکل نشان می‌دهد یک ماژول چه اجزائی دارد: ماژول خود را می‌نویسیم، به صورت واضح بیان می‌کنیم که ماژولمان از چه ماژول‌های دیگری استفاده می‌کند و چه پکیج‌هایی از این ماژول برای دیگر ماژول‌ها قابل دسترسی و دیدن است. به عنوان مثال می‌توانید java.ماژول بینید. ماژول SQL چه ماژول‌هایی را استفاده می‌کند و کدام پکیج خود را برای دیگر ماژول‌ها در دسترس قرار داده است:

شكل زیر بیان کننده



Packages		
Exports		
Package	Description	
java.sql	Provides the API for accessing and processing data stored in a data source (usually databases).	
javax.sql	Provides the API for server side data source access and processing from the Java™ Platform, Standard Edition.	
javax.transaction.xa	Provides the API that defines the contract between the transaction manager and a resource manager driver in JTA transactions.	

Indirect Exports		
From	Packages	
java.logging	java.util.logging	
java.xml	javax.xml.catalog javax.xml.datatype javax.xml.namespace javax.xml.parsers javax.xml.transform.sax javax.xml.transform.stax javax.xml.transform.stream javax.xml.org.w3c.dom.ranges org.w3c.dom.traversal org.w3c.dom.views org.xml.sax org.xml.sax	

Modules		
Requires		
Modifier	Module	Description
transitive	java.logging	Defines the Java Logging API.
transitive	java.xml	Defines the Java API for XML Processing (JAXP), the Streaming API for XML.

بارگذاری نمی‌شود، پس احتمالاً وقتی در خانه و در خواب راحت هستید، سیستم شما بخاطر JavaClassDefError کرش خواهد کرد و از خواب ناز خود بیدار خواهید شد، پس چه بهتر که همان ابتدای اجرای برنامه متوجه این ایراد شوید و آن را رفع کنید. اما بزرگترین اشکال این است که تمامی کلاس‌های موجود در classpath به هم‌دسترسی دارند. این نقض آشکاری از اصل محصورسازی (Encapsulation) است.

مکانیزمی که نبودش باعث شده بسیاری از JDK API‌های داخلی توسعه‌دهندگان استفاده شود sun.misc. مانند Base64Encoder یا sun.misc.Unsafe که همگی جزو Internal API‌های JDK است. هدف ما این است که کدها، متدها و کلاس‌هایی که قرار نیست از بیرون از آنها استفاده شود را پنهان کنیم. اما با مکانیزم فعلی چنین امری ممکن نیست. پس ما نیاز به سطح بالاتر و قوی‌تری از محصورسازی (Strong Encapsulation) نیاز داریم.

Runtime Image داشته باشید که دیگر چیزهای اضافی‌ای که از آنها استفاده نکرده‌اید را ندارد. با این قابلیت، حجم برنامه‌ها کم شده و شما می‌توانید از دستگاه‌های Embedded که حافظه محدودتری دارند هم استفاده کنید.

ماژول‌ها می‌توانند این مشکلات را حل کنند. اما اصلاً ماژول چیست؟ ماژول دارای یک اسم است، کدهای مرتبط و پکیج‌ها را در درون خود نگه می‌دارد و کامل است به این معنی که یک کار مشخص شده در دسترس دیگران قرار می‌دهد. یک ماژول به صورت صریح بیان می‌کند برای کار کردن به چه ماژول‌هایی نیاز دارد و کدام بخش از پکیج‌هاییش برای دیگر ماژول‌ها قابل دسترسی است. اینگونه به سادگی می‌توانیم وابستگی بین ماژول‌ها را تشخیص دهیم، بر روی Internal API‌هایمان کنترل داشته باشیم و در صورت نبود یک ماژول در آغاز برنامه از نبود آن فوراً مطلع شویم و در صورت وجود هرگونه تداخلی به سرعت به خطاب برخوریم.

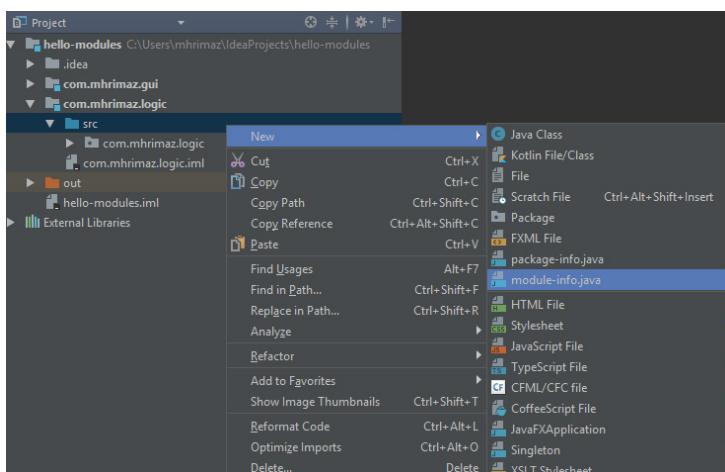
این گراف حال و هوای این روزهای JDK است. در پایین پایین

ساده‌ترین توصیف از یک مژول است:

module-info.java

```
module <module-name> {
    exports <package1>;
    exports <package2>;
    exports <package3>;
    ...
    requires <module1>;
    requires <module2>;
    requires <module3>;
}
```

بعد از ساختن یک مژول نیاز دارد که فایل module-info.java را به صورت جداگانه بسازید. برای ساختن این فایل روی src کلیک کنید و از قسمت new گزینه new گزینه module-info.java را انتخاب کنید:



پروژه‌ای با دو مژول به نام‌های com. و com.mhrimaz.logic ساخته‌ایم که ساختار این پروژه به شکل زیر است:

در مژول Greeting دو کلاس com.mhrimaz.logic وجود دارد.

: Greeting کلاس محتويات

```
package com.mhrimaz.logic;

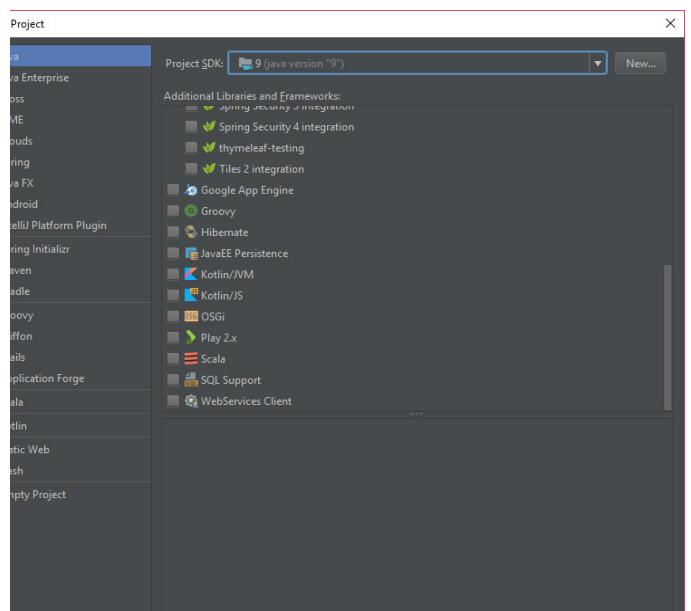
public class Greeting {

    public static String sayHello(String name) {
        return "Hello, " + name;
    }
}
```

چگونه؟

قبل از همه چیز نیاز دارد که جاوا ۹ یا ۱۰ را دانلود و نصب کنید. حال می‌خواهیم با استفاده از IDEA اولین پروژه مژول خود را بسازیم:

```
$ java -version
java version "9"
Java(TM) SE Runtime Environment (build 9+181)
```



برای ساختن یک مژول بر روی پروژه خود راست کلیک کنید

```

package com.mhrimaz.logic.internals;
public class InternalGreeting {
    public static String sayHello(String name){
        return "Hello, This Greeting is internal dear "+ name;
    }
}

```

محفویات فایل :module-info

```

module com.mhrimaz.logic {
    exports com.mhrimaz.logic;
}

```

همانطور که می‌بینید در فایل module-info ابتدا نام مازول آورده شده و سپس پکیج com.mhrimaz.logic (دقیق کنید که هر چند نام این پکیج با نام مازولمان یکسان است اما شما فقط می‌توانید پکیج‌ها را برای مازول‌های دیگر قابل دسترسی کرده‌ایم و دیگر به پکیج com.mhrimaz.logic.internals دسترسی نداده‌ایم.

محفویات برنامه اصلی MainApplication که یک برنامه ساده JavaFX است.

```

package com.mhrimaz.gui;
import com.mhrimaz.logic.Greeting;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class MainApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        Label label = new Label(Greeting.sayHello("Java"));
        StackPane pane = new StackPane();
        pane.getChildren().add(label);
        Scene scene = new Scene(pane);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

```

برنامه ساده‌ای که نوشته‌ایم فقط یک پنجره است که بر روی آن hello, Java نوشته می‌شود. این مازول در نگاه اول نیاز به دو مازول دیگر JavaFX یعنی javafx.controls و javafx.base دارد. همینطور برای دسترسی به کلاس Greeting نیاز به دسترسی به مازول دارد. پس احتمالاً محفوظات فایل module-info این مازول به شکل زیر است:

```

module com.mhrimaz.gui {
    requires javafx.base;
    requires javafx.controls;
    requires com.mhrimaz.logic;
}

```

با اجرای برنامه به خطای زیر برمی‌خوریم:

```

Caused by: java.lang.IllegalAccessException: class com.sun.javafx.application.LauncherImpl
(in module javafx.graphics) cannot access class com.mhrimaz.gui.MainApplication
(in module com.mhrimaz.gui) because module com.mhrimaz.gui does not export com.mhrimaz.gui to module
javafx.graphics

```

در خطابه وضوح توضیح داده شده که مازول برای انجام عملیات‌های خود نیاز به دسترسی به مازول

دارد، به همین دلیل ما باید پکیج com.mhrimaz.gui از ماثول com.mhrimaz.gui قابل دسترس و export کنیم. (همانطور که می‌بینید می‌توانید یک پکیج را برای ماثول‌های خاصی قابل دسترس کنید).

محتويات فایل module-info به شکل زیر خواهد شد:

```
module com.mhrimaz.gui {  
    requires javafx.base;  
    requires javafx.controls;  
    requires com.mhrimaz.logic;  
    exports com.mhrimaz.gui to javafx.graphics;  
}
```

حال بدون مشکل می‌توانیم برنامه خودمان را اجرا کنیم. نتیجه بیشتر شبیه یک باگ در پیاده سازی JavaFX است، اما در هر صورت خروجی‌ای که خواهید گرفت به شکل زیر است:

داستان ماثول‌ها به همین جا ختم نمی‌شود. ریزه کاری‌ها و نکات بسیار زیادی می‌توان در ماثولاریتی گفت. برای مطالعه بیشتر می‌توانید به دو کتاب خوب و جدید Java 9 Revealed و Java 9 Modularity مراجعه کنید. اگر می‌خواهید کمی بیشتر تمرین کنید و با ماثول‌ها دست و پنجه نرم کنید پیشنهاد می‌شود که به این^۳ مخزن در گیت‌هاب نگاهی بیاندازید.

The screenshot shows a Java code editor with a dark theme. On the left, there is a vertical line of numbers from 13 to 30. Lines 13 to 23 contain Java code for a `start` method. Lines 24 to 30 are empty. To the right of the code editor is a window titled "Hello, Java" with a red border. The window has standard minimize, maximize, and close buttons at the top right. The title bar also contains the text "Hello, Java". A yellow circle highlights the minimize button on the window.

```
13  @Override  
14  public void start(Stage primaryStage) throws Exception {  
15      Label label = new Label(Greeting.sayHello( name: "Java" ));  
16      StackPane pane = new StackPane();  
17      pane.getChildren().add(label);  
18  
19      Scene scene = new Scene(pane);  
20      primaryStage.setScene(scene);  
21      primaryStage.show();  
22  }  
23 }  
24  
25  
26  
27  
28  
29  
30
```

1 - <https://dzone.com/articles/what-is-jar-hell>

2 - https://en.wikipedia.org/wiki/Circular_dependency

3 - <https://github.com/AdoptOpenJDK/jdk9-jigsaw>

CPP 17

امیرراد کیمیابی

در این مقاله قصد داریم به معرفی C++17 بپردازیم که به بیاوریم:

```
#include <experimental/optional>
        تابع زیر را در نظر داشته باشید:
std::experimental::optional<string>
create(bool      b)
{
if(b)
return "NEW STRING";
    return {};
}
```

این تابع با ورودی `true` مقداری را در شی `optional` قرار می‌دهد و در غیر این صورت آن را خالی می‌گذارد. این تابع را تست می‌کنیم:

```
create(false).value_or("empty");
```

تابع `value_or` مقدار یک `optional` را چک می‌کند، اگر حامل مقداری بود آن را برگردانده و در غیر این صورت مقدار ورودی خود تابع بازگردانده می‌شود. که اینجا حاصل `empty` خواهد بود.

برای دسترسی به مقدار شی میتوان از عملگر * استفاده کرد:
if(auto str=create(true))
{
std::cout<<"string value is"<<*str<<std::endl;
}
به کمک تابع `reset` نیز می‌توان مقدار شی `optional` را خالی گذاشت.

آن `C++1z` نیز می‌گویند. `C++17` آخرین بازنگری استاندارد ISO/IEC ۲۰۱۷ است. این استاندارد در مارچ ۲۰۱۷ به مرحله DIS رسید و سپس به صورت همگانی تایید شد. نسخه استاندارد آن در دسامبر ۲۰۱۷ صادر و چاپ شد.

`cpp17` دارای تغییرات بسیاریست و ویژگی‌های جدیدی را معرفی کرده است که لیستی از آنها را می‌توانید در این لینک ببینید. ما در اینجا به بررسی بعضی از ویژگی‌های جدید و بهبود یافته این بازنگری می‌پردازیم. توجه کنید برای استفاده از `C++17` شما نیاز به نسخه‌ای مناسب از کامپایلر مدنظر خود دارید که لیست نسخه‌های مختلف کامپایلر و تناسب آنها با `C++17` را می‌توانید از اینجا ببینید. مثال‌های آورده شده همگی با کامپایلر `g++` از `gcc` نسخه ۷.۲.۰ تest شده اند.

std::optional

این کلاس برای مدیریت یک شی است که می‌تواند حامل مقدار خاصی باشد یا نباشد (در بعضی زبانهای دیگر به صورت مشابهی کلاسی با نام `Nullable` وجود دارد). یکی از استفاده‌های رایج برای چنین کلاسی می‌تواند تابعی باشد که ممکن است در میان عملیات متوقف شده و با موفقیت انجام نشود و مقداری را نتواند برگرداند. همچنین ذکر این نکته نیز ضروری است که هر زمانی که شیئی از این کلاس دارای مقدار باشد، آن مقدار به عنوان بخشی از خود شی `<option>` می‌باشد، به این معنی که هیچ تخصیص حافظه پویایی `T` می‌باشد، و این `wrapper` یک شی را مدل سازی می‌کند و نه یک اشاره‌گر.

برای استفاده از `std::optional` باید هدرفایل پایین را در برنامه

```
switch(char c=getchar();c)
{
    case 'a':...
}
```

با این کار می‌توان حوزه متغیرها را کوتاه‌تر نگه داشت. از دیگر خوبی‌های این روش در کد استفاده از mutex هاست. یک mutex در سازنده خود lock می‌کند و هنگامی که خارج از حوزه می‌شود، mutex را در مخرب خود آزاد می‌کند. پیش از این نیاز به تعریف حوزه خاصی با {} بود که مکان آزاد سازی mutex مشخص شود یا حتی ممکن بود آزاد کردن mutex فراموش شود.

درباره تایپ کلاس template توسط کامپایلر

بسیاری از کلاس‌ها در C++ بر پایه تایپ‌هایی هستند که می‌توان نوع آنها را به هنگام فراخوانی سازنده فهمید، اما باید در نسخه‌های قبلی حتماً نوع کلاس بیان می‌شد. حال در C++17 کامپایلر با استفاده از فراخوانی سازنده به صورت خودکار نوع کلاس template را در می‌یابد.

```
std::pair my_pair (123, "abc");
// std::pair<int, const char*>
```

فرض کنید کلاس پایین را نوشتیم:

```
template <typename T1, typename T2,
          typename T3>
class my_wrapper {
T1 t1;
T2 t2;
T3 t3;
public:
explicit my_wrapper(T1 t1_, T2 t2_, T3 t3_)
: t1{t1_}, t2{t2_}, t3{t3_}
{}
```

حال با ساختار جدید کامپایلر می‌توانیم یک شی به این صورت بسازیم:

```
my_wrapper wrapper {1.23, 123, "abc"};
// pre c17++ my_wrapper<int, double,
// const char *> wrapper {1.23, 123, "abc"};
```

اما این متند برای variadic template ها چه کاری می‌کند؟ برای مثال:

کلاس std::variant نوعی union است. یعنی یک شی از این کلاس در هر لحظه مقدار آن تنها از یکی از تایپ‌های آورده شده است یا در مورد ارور مقداری ندارد. همانند optional هر شی variant که variant مقداری از نوع T است آن شی T داخل خود شی variant نگه داشته شده است و نمی‌تواند حافظه پویا اضافی اختصاص دهد. یک variant همچنین نمی‌تواند اشاره گر آرایه یا تایپ void را در خود نگه دارد.

برای استفاده از این کلاس باید هدفایل زیر را در کد خود بیاورید:

```
#include <variant>
```

این کلاس به این صورت تعریف شده است:

```
template <class... Types>
```

```
class variant;
```

پس می‌توانیم از هر تعداد تایپ دلخواه برای ساخت یک شی از این کلاس استفاده کنیم:

```
variant<int,float> v=1.0f;
```

با استفاده از تابع get می‌توان به مقدار این تابع دسترسی داشت:

```
int i=std::get<int>(v);
```

اما اگر تابع get را با تایپ float برای v فراخوانی کنیم چه؟

در این هنگام به دلیل دسترسی به نوعی که در variant مقدار ندارد، اکسپشن bad_variant_access ایجاد می‌شود.

با استفاده از تابع holds_alternative می‌توان چک کرد که مقدار حاضر در variant از نوع تایپ داده شده است یا نه:

```
variant<string,int> c1="abc"s;
```

```
cout<<boolalpha<<
```

```
holds_alternative<string>(c1)<<endl;
```

به کمک تابع index از خود کلاس variant نیز می‌توان به اندیس تایپی که در حال حاضر مقدار دارد دسترسی پیدا کرد:

```
variant<string,bool> v3=move(string("abc"));
```

```
cout<<"the index held by v3 is "<<
```

```
v3.index()<<endl;
```

if-initializer

در C++17 می‌توان علاوه بر شرط داخل بلوک‌های if و switch در آنها متغیر نیز تعریف کرد که حوزه آن محدود به خود بلوک است و پس از خارج شدن از if یا switch از حوزه خارج می‌شود.

به همچنین باید گفت که علاوه بر فرم ... args+ صورت ... +args نوشته که در مورد تابع جمع تفاوتی نمی‌کند اما بسته به اپراتورهای دیگر می‌تواند بسیار متفاوت باشد. اگر نقطه‌ها در سمت راست باشند به آن right fold و اگر در سمت چپ به آن left fold می‌گویند.

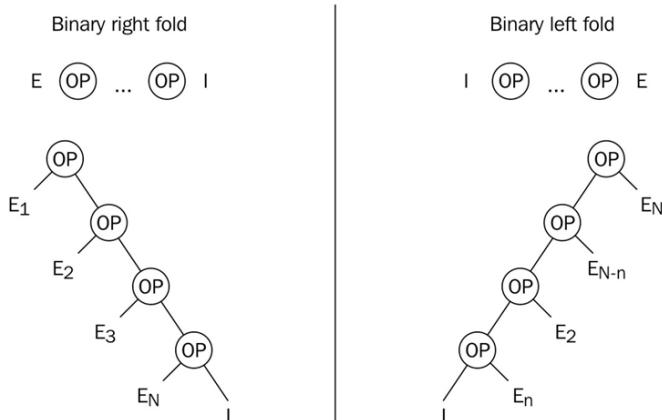
برای بیان تفاوت این دو می‌توانیم بگوییم که تابع بالا در حالت left fold به صورت $((4+5)+(3+2)+1)$ ارزیابی می‌شود و در حالت right fold به صورت $(5+(4+(3+2)))+1$ ارزیابی می‌شود.

اگر تابع بالا بدون پارامتر صدا شود چه اتفاقی می‌افتد؟

برای بسیاری از اپراتورها این یک اور می‌باشد، زیرا مقداری برای انجام عملیات وجود ندارد (تعداد عملوندها کافی نیست). اما اگر بتوانیم مقداری را به صورت پایه به این جمع اضافه کنیم این مشکل رفع می‌شود:

```
template <typename ... Ts>
auto sum(Ts ... ts)
{
    return (ts + ... + 0);
}
```

به چنین fold-ی با مقدار اولیه binary fold می‌گویند. باز می‌توان گفت که می‌توان صفر اضافه شده را هر دو طرف عبارت fold نوشت. با توجه به شکل زیر تفاوت دو فرم نوشتاری مشخص تر می‌شود:



مثالی دیگر با استفاده از fold expression ها برای محاسبه

تعداد رخداد یک الگان در یک container

```
template<typename R,typename ... T>
auto num_occurrence(const R& range,
                    T ... values){
    return (std::count(std::begin(range),
                      std::end(range),values )+ ...);
}
```

```
template <typename T>
struct sum {
    T value;
};

template <typename ... Ts>
sum(Ts&& ... values) : value{values + ...} {};
```

حال فرض کنید شئی بسازیم با تعداد پارامتر دلخواه از نوعهای مختلفی که قابل تبدیل شدن به هم هستند. تایپ T چه خواهد بود؟

برای حل این مسئله یک راهنمای فهم تایپ مستقیم قرار خواهیم داد:

```
template <typename ... Ts>
sum(Ts&& ... ts) -> sum<std::common_type_t<Ts...>>;
```

حال می‌توانیم از این کلاس به همان صورت قبلی شی بسازیم:

```
sum s {1u, 2.0, 3, 4.0f};
```

```
sum string_sum {std::string{"abc"}, "def"};
```

الان در کد بالا s با وجود فراهم شدن پارامترهایی از نوع unsigned, float, double و std::string با وجود const char[] تایپ مشترک در نظر می‌گیرد و برای s نیز با وجود std::string در نظر گرفته می‌شود.

Fold Expressions

از زمان C++11 و با آمدن variadic template pack ها می‌توان توابع یا کلاس‌هایی ساخت که هر چند تعداد پارامتر را می‌پذیرند. حال می‌توان با کمک fold expression تمامی این پارامترها را در یک عبارت کنار هم آورد و توابع جالبی نوشت.

برای مثال تابعی که تمام ورودی‌هایش را جمع می‌کند:

```
template <typename ... T>
auto sum(T ... args)
{
    return (args+ ... );
}

std::cout<<sum(5,4,3,2,1)<<std::endl;
//outputs 15
```

به عملیات بالا با اپراتور (+) می‌گویند. C++17 به

طور کلی پارامترها را با اپراتورهای زیر fold می‌کند:

```
+ - * / % ^ & | = < > << >> += -= *= /= %= ^= &= |= <<= >>= == != <= >= && || , .* ->*
```

Book book("A Tale of Two Cities","Charles Dickens",2000,1);

با استفاده از این قابلیت می تواند فیلدهای این struct را دریافت کرد:

```
auto [name,author,pages,id]=book;
cout<<"name \t"<<"author \t"<<
"pages \t"<<"id"<<endl;
cout<<name<<"\t"<<author<<"\t"<<
pages<<"\t"<<id<<endl;
```

یکی دیگر از موردهای خوب برای استفاده از این قابلیت استفاده از آن برای پیمایش اشیای یک container و دریافت بخش های مختلف آن است. مثلاً می توان key و value آیتم

های یک map را به صورت زیر دریافت کرد:

```
std::map<string,int> dictionary;
for (auto & [key,value]: dictionary)
cout<<key<< " : " << value<<endl;
```

توجه کنید که اگر تابعی داشته باشد که تعداد متغیرهای structured فراهم شده در سمت چپ یک عبارت که به صورت binding در آمده با تعداد متغیرهای بازگردانده شده سمت راست مساوی نباشد، برنامه شما درست نمی باشد و ارور زمان کامپایل رخ می دهد.

Structured Binding

در C++17 ویژگی جدید اضافه شده است که شباهت بسیاری به unpacking در زبانهای دیگر دارد. structured binding نوعی syntactic sugar است که با استفاده از فهم تایپ اتوماتیک توسعه کامپایلر به شما اجازه می دهد که مقدارهایی که از نوع struct یا tuple::pair هستند را به متغیرهای مستقل نسبت دهید.

تابع زیر را در نظر بگیرید:

```
std::pair<int,int> divide(const int &a,
                           const int &b)
{
    int remainder=a%b;
    int fraction=a/b;
    return std::pair<int,int>(fraction,
                               remainder);
}
```

قبل از C++17 برای فراخوانی این تابع و استفاده از مقادیر آن باید اینگونه عمل می کردید:

```
const auto result(divide(10,3));
std::cout<<"fraction is "<<
result.first<<"and remainder is "<<
result.second<<std::endl;
```

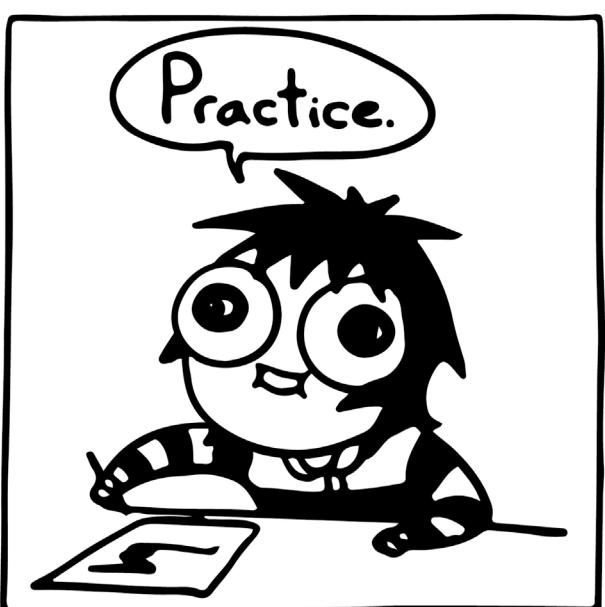
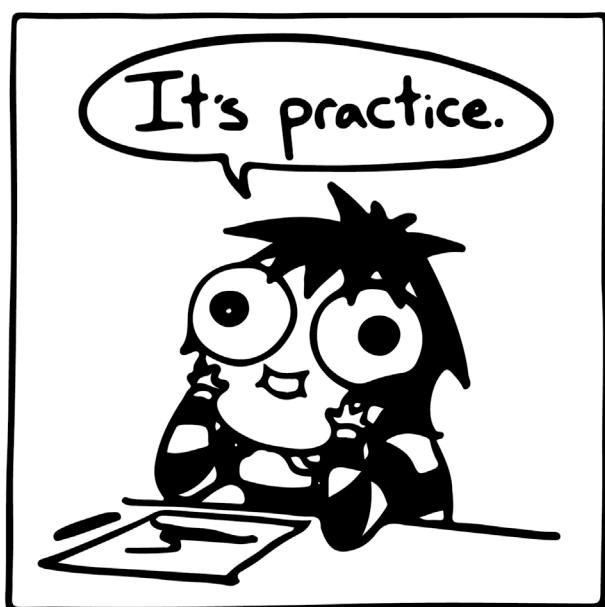
اما با وجود structured binding اینگونه می نویسیم:

```
auto [fraction,remainder]=divide(12,4);
std::cout<<"fraction is "<<fraction<<
" and remainder is "<<remainder<<std::endl;
از این قابلیت می توان حتی برای struct ها نیز استفاده کرد.
```

تابع زیر را در نظر بگیرید:

```
struct Book
{
    std::string title;
    std::string author;
    unsigned long pages;
    int id;
    Book(std::string nm,
          std::string author,
          unsigned pages,int id)
        :title(nm),author(author),
         pages(pages),id(id)
    {
    };
};
```

حال یک شی از این struct ایجاد می کنیم:



© Sarah Andersen

چطوری انقدر خوب کد میزنی؟

Unity ShaderLab

حسن صادقین



که برای عملیات‌های گرافیکی که بر روی

از اطلاعات دیگر که باید دانست
دستگاه‌های مختلف اشکال
است که در مجله قبلی بررسی کردیم
که شامل ماتریس‌های Model, View, Projection
بود که با ضرب پیاپی این
ماتریس‌ها در نقطه مورد نظر آن را به
فضای نمایش صفحه که ۲ بعدی است
انتقال می‌دهیم.

اطلاعات دیگر شامل تمامی دانش‌های درس هندسه تحلیلی است که باید تا حدی آنها را دانست.

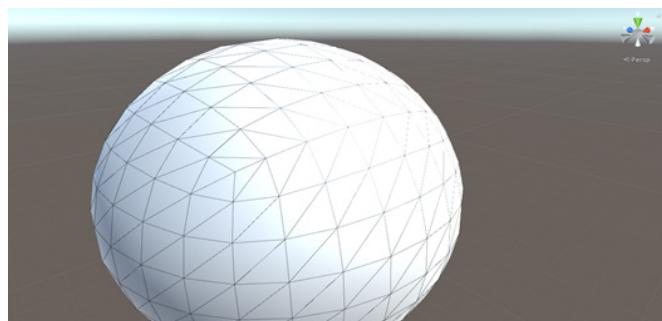
Graphics Pipeline

یک مدل مفهومی در CG است که مراحل تبدیل یک صحنه^۳ بعدی به^۲ بعدی را شرح می‌کند که این مراحل شامل یکسری مراحل قابل برنامه نویسی و یکسری غیرقابل برنامه نویسی است، که ما به ترتیب مراحل قابل برنامه نویسی که باید در shader ها نوشته شوند می‌برازیم.

تک تک پیکسل ها و راس های ما اجرا می شوند عالیست .
Shader ها برنامه هایی هستند که بر اساس Graphics Pipeline به اجرا در می آیند و باعث می شوند که اشکال سه بعدی در صفحه نمایش دو بعدی با افکت های خاص به اجرا درآیند .

اطلاعات اولیه

در CG یکسری اصطلاحات اولیه وجود دارد که دانستن آنها خالی از لطف نیست از جمله اینکه تمامی اشکالی که می‌سازیم وجود دارد همگی از اشکال اولیه تشکیل شده‌اند. این اشکال شامل نقطه، خط و مثلث هستند. برای مثال کرده نیز از مثلث‌های کوچک و زیاد تشکیل شده است همانند شکل زیر:



در کامپیوتر گرافیک یا به اختصار CG GPU ها برنامه هایی هستند که در shader ها اجرا می شوند و عملیات سایه زنی را شبیه سازی می کنند. این عملیات سایه زنی شامل تولید نور مناسب، سایه، رنگ هر پیکسل و تولید افکت های خاص می شود.

چون عملیات‌های گرافیکی نیازمند پردازش زیاد هستند CPU‌ها به علت تعداد کم پردازنده توانایی پردازش این عملیات‌ها را ندارند. CPU‌ها حداکثر در بازار امروز ۱۸ هسته هستند که این هسته دارای توان پردازشی بالا هستند اما عملیات‌های گرافیکی سبک ولی به تعداد بسیار زیادی هستند برای مثال برابر با ۷۳۷۷۲۸۰ مگابایتی^{۱۰۲۴*۷۲۰} صفحه یک

پیکسل داریم که باید در هر فریم
بر روی تک تک این پیکسل ها
عملیات های گرافیکی را انجام دهیم
که با ۱۸ هسته CPU اصلاً توانایی
چنین کاری را نداریم. در عوض GPU
ها داری تعداد هسته های زیاد
اما با توان پردازشی، کم هستند

Vertex Shader

می‌شود که این رنگ شامل نور، سایه، درخشندگی و... است. این شیدر کاری با راس‌ها ندارد و فقط بر روی پیکسل‌ها کارهایی را انجام می‌دهد. این شیدر تقریباً هزینه‌بر ترین شیدر در GP است که برای بهبود در fps بهتر است این مرحله به خوبی نوشه شود.

Compute Shader

یک شیدر جدا از GP است که هیچ تاثیری هم به صورت مستقیم بر روی پیکسل‌ها و راس‌ها ندارد. این شیدر تنها شباهتی که با دیگر شیدرها دارد استفاده از زبان یکسان HLSL و یا GLSL است. از کاربردهای آن می‌توان به استفاده از قابلیت موازی سازی GPU در زمینه‌های عمومی (به غیر از گرافیک) است که از این حیث می‌توان آن را به OpenMP و یا CUDA شبیه کرد.

Unity ShaderLab

در یونیتی شیدرها را می‌توان با زبان HLSL نوشت، در یونیتی یکسری شیدر به عنوان include file در نظر گرفته شده است که در آن‌ها برخی از مسائل پیاده سازی شده است و ما کافی است از آن‌ها استفاده کنیم. روش دیگری نیز وجود دارد که مخصوص خود یونیتی است و مقداری راحت‌تر است و محدود به یکسری از کارهای مرحله Fragment است و مراحل کامل GP را پشتیبانی نمی‌کند.

فرم کلی شیدرها در یونیتی به شکل زیر است:

```
Shader "GroupName/Name"
{
    Properties {
        _A("name in editor", Type ) =
initialValue
        _B("name in editor", Type ) =
initialValue
    }
    Subshader {
        Tags{
            }

        //Example
        float4 _A;
        float _B;
        Pass{
            CGPROGRAM
            #pragma vertex ver
            #pragma fragment frag
            //Code
            ENDCG
        }
    }
}
```

اوین مرحله قابل برنامه‌نویسی در GP است که ورودی آن تمامی راس‌هایی است که در صحنه داریم. درواقع این shader به ازای تمامی راس‌هایی که در صحنه داریم یک بار اجرا می‌شود و محاسباتی را روی آن راس‌ها انجام می‌دهد و به عنوان خروجی نیز راس‌های جدید تولید می‌کند. تمامی تبدیلهای مختصاتی از محلی به جهان بازی و همینطور دوربین بهتر است در این مرحله انجام شود.

از جمله کارهای دیگر که می‌توان در این مرحله انجام داد می‌توان به متحرک کردن راس‌ها اشاره کرد که می‌توان material هایی متحرک ساخت به مانند آب.

Geometry Shader

این مرحله که انتخابی است و نیز به نسبت جدید است و از DirectX10 به بعد قابل دسترس و برنامه نویسی است، به عنوان ورودی اشکال هندسی اولیه از جمله نقطه، خط و مثلث را دریافت می‌کند، به عنوان مثال یک مثلث دارای ۳ راس است که این ۳ راس به عنوان یک ورودی به این شیدر در نظر گرفته می‌شود.

خروجی این شیدر نیز همان اشکال هندسی اولیه است اما با تعداد متفاوت، برای مثال یعنی مثلث‌های کمتر یا بیشتر. از این شیدر برای ایجاد آرت استایلهایی مانند LowPoly، شبیه سازی پشم، شبیه سازی لباس و... استفاده می‌شود.

Tessellation Shader

این شیدر جدیدترین عضو از GP است که در DirectX11 و OpenGL4.0 اضافه شده است. این شیدر دارای دو قسمت قابل برنامه نویسی به نام Hull و Domain است که این اجازه را می‌دهد تا اجسام با شبکه (mesh) ساده‌تر به اجسامی با شبکه پیچیده‌تر تبدیل شوند. این افزایش جزئیات را می‌توان با (LOD) level of detail وابسته کرد که به این صورت که هر چه قدر اجسام از دوربین دورتر شوند جزئیات کمتری به نمایش درآید و بر عکس هرچه دوربین به اجسام نزدیک‌تر شود جزئیات بیشتری به نمایش درآید.

ورودی این مرحله اشکال اولیه از جمله مثلث است و که می‌توان با تقسیم کردن آنها به مثلث‌های بیشتر جزئیات بیشتری را ایجاد کرد.

Fragment Shader

آخرین مرحله در GP است که این شیدر به ازای تمامی پیکسل‌ها در هر فریم اجرا می‌شود. در این مرحله رنگ پیکسل‌ها مشخص

```

Shader "Surface/Diff&Spec" {
    Properties {
        _Tint ("Color", Color) = (1,1,1,1)
        _MainTex ("Albedo (RGB)", 2D) =
"white" {}
        _Glossiness ("Smoothness",
Range(0.5 = ((0,1
})
    SubShader {
        Tags { "RenderType"="Opaque" }
        CGPROGRAM
            #pragma surface surf Standard
fullforwardshadows
            #pragma target 3.0
            sampler2D _MainTex;
            struct Input {
                float2 uv_MainTex;
            };
            half _Glossiness;
            fixed4 _Tint;
            void surf (Input IN, inout
SurfaceOutputStandard o) {
                fixed4 c = tex2D (_MainTex, IN.uv_
MainTex) * _Tint;
                o.Albedo = c.rgb;
                o.Smoothness = _Glossiness;
                o.Alpha = c.a;
            }
        ENDCG
    }
    FallBack "Diffuse"
}

```

همانطور از کد بالا معلوم است در ابتدا با تعریف subshader تگ‌های آنرا مشخص کرده‌ایم که در این مثال این تگ به معنای رندر شدن مقصد این شیدر به عنوان یک شیء مات است، سپس با تعریف CGPROGRAM و ENDCG می‌توانیم کد مربوط به شیدر را تعریف کنیم.

در SurfaceShader ها ما دیگر طبق GP پیش نمی‌رویم بلکه فقط می‌توانیم از vertex و خود surface استفاده کنیم، که در مرحله surface تمامی نورپردازی‌ها به صورت اتوماتیک انجام می‌پذیرد و فقط کافی است متغیرهایی را مانند Albedo و Smoothness مقداردهی نماییم.

که در کد بالا تعریف شده است به آن معنی است که اگر تمامی subshader های ما نتوانستند در گرافیک مقصد اجرا شوند یک شیدر دیگر را برای اجرا در نظر بگیریم که در این مثال شیدر diffuse خود یونیتی است که با پیاده سازی کامل آن برای اکثر کارت گرافیک‌ها می‌توان آن را اجرا کرد.

: Vertex & Fragment Shader. ۲

در ابتدا باید property های خود را مشخص کنیم که در این مورد ما نیاز به یک texture و یک رنگ برای رنگ اصلی شکل

Properties آن‌ها را تغییر داد و در کد قبل از استفاده از آن‌ها باید در قسمت کد آن‌ها را تعریف کرد.
هر شیدر می‌تواند چندین SubShader داشته باشد تا کارت گرافیک مقصد بتواند از میان آنها اولین SubShader سازگار و مناسب را اجرا کند.

Tag ها مشخص کننده یکسری از ویژگی‌های هر SubShader و هر Pass هستند که برای درک بهتر آن‌ها باید آن‌ها را در عمل دید.

هر شیءی را که به آن یک شیدر دهیم به اندازه تعداد ها رندر می‌شود برای مثال اگر دو تا Pass داشته باشیم آن شیء ۲ بار رندر می‌شود.

در ابتدا باید برای compiler مشخص کنیم که از چه مراحل و چه برنامه‌ایی می‌خواهیم استفاده کنیم که این کار را می‌توان با ماکرو‌ها انجام دهیم. اگر مراحل GP را مشخص کنیم برای مثال به صورت شکل بالا این شیدر به نام unlit در یونیتی وجود دارد اما می‌توان از نوع دیگری به نام SurfaceShader استفاده کنیم که باید از ماکرو به صورت زیر استفاده کرد.

#pragma surface surf

برای آشنایی بیشتر در ادامه مقاله به پیاده سازی یک مثال می‌پردازیم

Simple Diffuse and Specular Lighting:

: Surface Shader. ۱

در ابتدا پس از مشخص کردن اسم باید property های خود را مشخص کنیم که این کار به صورت زیر انجام می‌پذیرد.

```

Shader "Example/Diffuse Texture" {
    properties {
        _MainTex ("Texture", 2D) = "white" {}
        _Tint("Color", color) = (1,1,1,1)
    }
}

```

سپس باید subshader و تگ‌های مخصوص به کار خود را تعریف کنیم، طبق آنچه که گفته شد در هر subshader می‌تواند تعدادی pass وجود داشته باشد یا اصلاً نباشد که در این صورت تمامی اشکالی که این شیدر را دارا باشند به ازای همان کدی که در subshader می‌نویسیم رندر می‌شوند.

داریم.

```

float4 vertex : POSITION;
float3 normal : NORMAL;
float uv : TEXCOORD0;
};


```

```

struct v2f{
    float4 position : SV_POSITION;
    float3 worldNormal : NORMAL;
    float3 worldPosition : TEXCOORD0;
    float2 uv : TEXCOORD1;
};

```

کلمات کلیدی که بعد از دونقطه نوشته شده‌اند semantic نام دارند که مشخص کننده آن هستند که متغیر ساخته شده که از نوع عمومی مثل float4 است چه چیزی را نشان می‌دهند برای مثال vertex یک float4 است اما برای آنکه مشخص کنیم این 4 عدد چه چیزی را مشخص می‌کنند نیاز است تا از semantic ها استفاده کنیم. سماتیک‌هایی که داری SV هستند به معنای system variable هستند که باید حتماً مشخص شوند.

همانطور که قبلاً گفته شد اولین مرحله قابل برنامه نویسی در GP مرحله Vertex است که اسم تابع آنرا به وسیله ماکروها قبل مشخص کردیم. حال باید کارهایی را که در آن انجام دهیم را مشخص کنیم.

```

v2f vert(appdata i){
    v2f o;
    o.position = UnityObjectToClipPos(i.vertex);
    o.worldPosition = mul(unity_ObjectToWorld,
    i.vertex);
    o.worldNormal = UnityObjectToWorldNormal(i.normal);
    o.ub = TRANSFORM_TEX(i.uv, _MainTex);
    return o;
}

```

در مقاله قبل مقداری اطلاعات در مورد ماتریس‌های World, View, Projection صحبت شد که این ماتریس‌ها یک راس با مختصات محلی خود را ابتدا به مختصات جهانی و سپس به فضای دید بیننده یا دوربین جا جای می‌کنند که مجموعه این کارها را با استفاده از تابع UnityObjectToClipPos انجام می‌شود.

اما در مواردی نیاز است که مختصات راس‌ها را فقط به مختصات جهانی تبدیل کرد تا بتوانیم از آن‌ها استفاده کنیم که این کار را با استفاده از ضرب یک ماتریس در راس مربوطه انجام می‌دهیم که این ماتریس را یونیتی در اختیار ما قرار داده است که در شکل بالا به نام unity_ObjectToWorld وجود دارد.

در مرحله vertex تمامی کارهایی که می‌توانیم انجام دهیم

```

Properties {
    _MainTex ("Texture", 2D) = "White" {}
    _Tint ("Color", Color) = (1,1,1,1)
}

```

سپس باید subshader و تگ‌های مخصوص به کار خود را تعریف کنیم، طبق آنچه که گفته شد در هر subshader می‌تواند تعدادی pass وجود داشته باشد یا اصلاً نباشد که در این صورت تمامی اشکالی که این شیدر را دارا باشند به ازای همان کدی که در subshader می‌نویسیم رندر می‌شوند.

```

SubShader{
    Pass{
        Tags{
            "LightMode" = "ForwardBase"
        }
        CGPROGRAM
        #pragma vertex vert
        #pragma fragment frag
        //Include file's
        #include "UnityStandardBRDF.cginc"
        #include "UnityCG.cginc"
    }
}

```

تگی که در Pass مشخص شده است برای تعیین کردن نوع نوپردازی است. که در این مورد یعنی به روش Forward و فقط منبع نور مستقیم که در ابتدا تعریف کرده‌ایم در آن تاثیر داشته باشد.

سپس با تعریف CGPROGRAM می‌توانیم کد مربوط به شیدر را بنویسیم که این کار را با مشخص کردن توابع vertex و fragment با نام‌های vert و frag انجام می‌دهیم. برای راحت‌تر کردن نوشتن کد می‌توانیم از یکسری include file که یونیتی آن‌ها را فراهم کرده است استفاده کنیم در این فایل‌ها یکسری ماکرو و توابع تعییه شده است که می‌توانیم استفاده کنیم.

دقت کنید که include file ها دارای پسوند cginc هستند و خود شیدرها دارای پسوند shader هستند. برای استفاده از property ها در کد باید آن‌ها را تعریف کرد که این کار را به شکل زیر انجام می‌دهیم.

```

sample2D _MainTex;
float4 _MainTex_ST
float4 _Tint;

```

سپس باید یکسری struct برای راحت‌کردن کار خود تعریف کنیم که این struct ها به عنوان ورودی و خروجی‌های مراحل GP هستند.

```
struct appdata{
```

دانستن زاویه تابش و بردار نرمال نیاز به زاویه دید بیننده یعنی دوربین نیز داریم. بردار زاویه دوربین به شکل را می‌توان با کم کردن مختصات دوربین و شکل مورد نظر بدست آورد.

```
float3 viewDir = normalize(_WorldSpaceCameraPos - i.worldPosition);
```

حال باید زاویه بازتاب نور را محاسبه کنیم که این کار را با reflect استفاده از دو بردار زاویه تابش و بردار نرمال توسطتابع محاسبه می‌شود. با صرب نقطه‌ایی بین زاویه دید و زاویه بازتاب می‌توان این نوع نور را محاسبه کرد اما برای آنکه از شدت آن بگاهیم می‌توانیم با به توان رساندن آن با یک عدد دلخواه این کار را انجام دهیم.

خروجی مرحله fragment یک رنگ است که در پیکسل‌ها تزریق می‌شود پس باید در این تابع یک رنگ به عنوان خروجی بازگردانیم که این کار را با جمع دو مقدار نوری که بدست آورده‌یم می‌توانیم انجام دهیم.

کارهای بسیار فراوانی می‌توان در این حوزه انجام داد که در این مقاله نمی‌توان آن‌ها را گنجاند اما اگر علاقه‌ایی به این موضوعات داشتید خوشحال می‌شویم در این زمینه به شما کمک کنم.

از منابعی که می‌توانید برای یادگیری استفاده کنید می‌توان به

HLSL Development Cookbook, Unity.5.x Shaders and Effect Cookbook اشاره کرد.

مربوط به راس‌هاست درواقع از آنجایی که ورودی این مرحله راس است پس در این مرحله فقط می‌توانیم روی آن‌ها کار کنیم برای مثال انجیمن دادن به راس‌ها در مواردی مانند حرکت آب و درخت در این مرحله می‌توان انجام داد.

مرحله بعد fragment است که در این مرحله ما با پیکسل‌ها کار می‌کنیم درواقع این تابع در هر فریم برای یکسری از پیکسل‌های مشخص اجرا می‌شود. اصل محاسبات پردازش معمولاً در این مرحله است و معمولاً تمامی کارهای نورپردازی در این مرحله اجرا می‌شود.

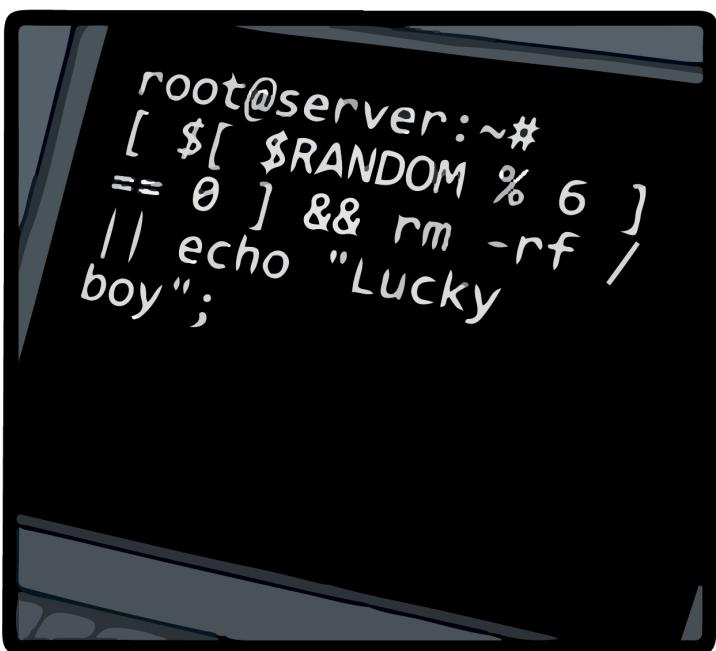
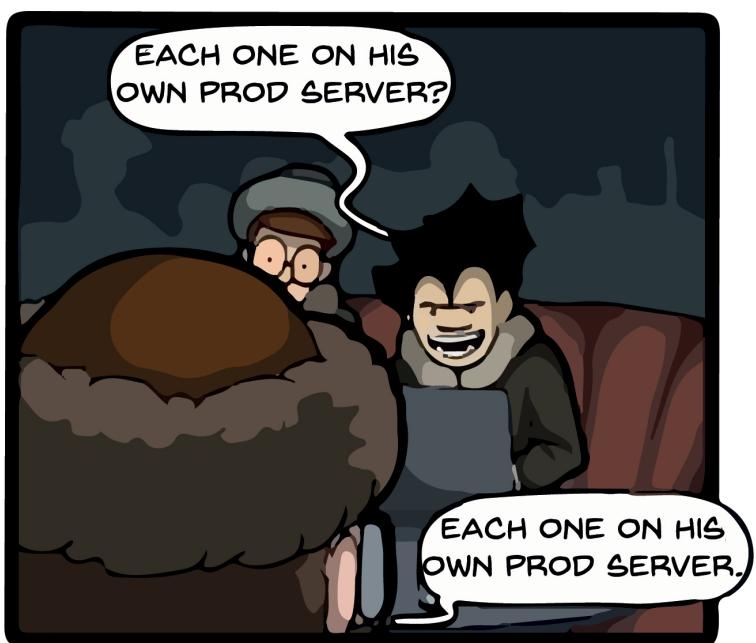
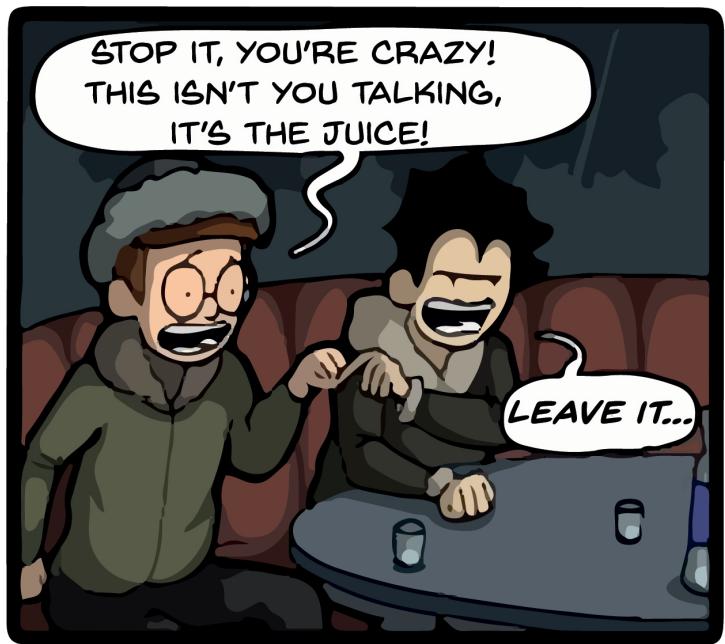
```
Fixed4 frag(v2f i) : SV_TARGET{
    float4 albedo = tex2D(_MainTex, i.uv);
    albedo.rgb *= _Tint.rgb;
    float lightDir = _WorldSpaceLightPos0.xyz;
    float3 viewDir = normalize(_WorldSpaceCameraPos - i.worldPosition);
    float3 reflectionDir = reflect(-lightDir, i.worldNormal);
    float NdotL = DotClamped(lightDir, i.worldNormal);
```

```
    float3 specular = pow(DotClamped(viewDir, reflectionDir), _Smoothness * 100) * _LightColor0.rgb;
```

```
    float3 diffuse = NdotL * albedo * _LightColor0.rgb;
    return float4(diffuse + specular, 1);
}
```

در خط اول این تابع، ما texture اصلی شکل خود را با توجه به مختصات uv می‌خوانیم و سپس در خط بعدی رنگ دلخواهی را که در property مشخص کرده بودیم را به آن اضافه می‌کنیم. حال برای آنکه نور را وارد کار کنیم باید در ابتدا جهت تابش نور را بدست بیاوریم که این بردار جهت توسط خود یونیتی WorldSpaceLight-Pos0 است که همان طور که از اسم آن معلوم است این بردار در دستگاه مختصاتی جهانی است.

برای آنکه بتوانیم نور Diffuse را شبیه سازی کنیم باید در نقطه مورد نظر بردار نرمال(عمود بر صفحه) را با زاویه تابش نور ضرب نقطه‌ایی کنیم که این فرآیند با استفاده از تابع Dot-Clamp در شکل بالا انجام شده است. این مقداری را که بدست می‌آوریم درواقع شدت این نور در نقاط مختلف اشکال است که باید در texture و رنگ نور تابیده شده ضرب شود. اما برای آنکه نور Specular را شبیه سازی کنیم علاوه بر



صرف برای خنده!

الگوهای طراحی در مهندسی نرم افزار

سجاد جعفری

قسمت اول

مقدمه

به حال الگوهای طراحی گسترش پیدا کردند اما همچنان ۲۳ الگوی مهندسی نرم افزار با مشکلاتی مواجه هستیم که اصطلاحاً به آنها recurring problem می‌گویند. یعنی مشکلاتی که هر کس در حوزه مهندسی نرم افزار کار کند با آنها روبرو می‌شود. در طول سال‌ها برای این مشکلات تکرار شونده راه حل‌های موفقی پیدا شده است؛ این راه حل‌ها پس از حذف جزئیات خاص (مثلاً جزئیات مربوط به زبان برنامه نویسی) و abstract شدن ثبت می‌شوند به طوری در موقعیت‌های مشابه بتوان از آنها استفاده کرد.

اولین و مهم‌ترین و مشهورترین الگوها در مهندسی نرم افزار الگوهای طراحی (design pattern) هستند. چهار نفر به نام‌های Erich Gamma, John Vlissides, Ralph Johnson, Richard Helm در سال ۱۹۹۵، ۲۳ تا از این الگوها را جمع‌آوری کردند و در کتاب *Gang of Four* منتشر نمودند. نویسنده‌های کتاب هم صنعتی بودند و هم دانشگاهی و این نشان از این دارد که الگو جزو موضوعاتی هست که در صنعت و دانشگاه مورد توجه است.

الگوهای کتاب *Gang of Four* به الگوهای طراحی اصلی معروف هستند. از آن زمان تا وقتی یک الگو را تعریف می‌کنیم هم ساختار کلاس‌های درگیر در الگو را مشخص نیز الگویی ثبت شده‌اند.

در شماره‌های قبلی نشریه با تعدادی از الگوهای طراحی در مهندسی نرم افزار آشنا شدیم. در این مقاله نگاهی کلی به مفهوم pattern در مهندسی نرم افزار خواهیم انداخت و راجع به زمینه به وجود آمدن و گسترش و اصولی که الگوهای طراحی مبتنی بر آن هستند خواهیم نوشت. به شدت توصیه می‌کنیم که بعد از خواندن این مقاله نگاه مجددی به pattern‌هایی که تاکنون یاد گرفته‌اید بیندازید. این مقاله به شما کمک می‌کند که حس و فهم بهتری نسبت به الگوهای طراحی و استفاده از آنها داشته باشد. اگر هیچ چیز از pattern نمی‌دانید ۳ پاراگراف بعدی را بخوانید! اما اگر با الگوها آشنایید و حوصله قصه خواندن ندارید ۳ پاراگراف به جلو بروید.

در مهندسی نرم افزار هدف اصلی الگوها استفاده مجدد (reuse) از راه حل‌های موفق است. راه حل‌های موفق برای حل مشکلات ثبت می‌شوند تا بتوان از آنها در وضعیت‌های مشابه استفاده کرد. امروزه تقریباً در تمامی حوزه‌های مهندسی نرم افزار از تحلیل گرفته تا طراحی و پیاده‌سازی و تست و مراقبت و نگهداری الگو وجود دارد. در حوزه‌های معماري و بازنده‌سی و مهندسی معکوس

اولویت بیشتری دارد زیرا flexibility در آن بالاتر است. (هدف الگوهای طراحی). به این اصل در مهندسی نرم‌افزار اصطلاحا Composite reuse principle می‌گویند.

Protected Variation : این اصل می‌گوید باید مراقب

قسمت‌هایی از سیستم که تغییرات در آن زیاد است بود و جلوی انتشار تغییرات به قسمت‌های دیگر سیستم را گرفت. چطور؟ باید آن قسمت را encapsulate کرد. اصطلاحا می‌گویند که باید یک wrapper دور این قسمت از سیستم کشید تا قسمت‌های دیگر سیستم از طریق یک wrapper (مثلاً یک interface) با آن در ارتباط باشند و از تغییرات آن مطلع نشوند.

تمامی ۳ اصل فوق برای این می‌باشد که وابستگی و flexibility را بین اجزا تشکیل دهنده سیستم کم کنند تا افزایش پیدا کند.

دسته بندی الگوهای طراحی:

Purpose			
	Creational	Structural	Behavioral
Class	Factory Method	Adapter (class)	Interpreter
			Template Method
Scope	Abstract Factory	Adapter (object)	Chain of Responsibility
Object	Builder	Bridge	Command
	Prototype	Composite	Iterator
	Singleton	Decorator	Mediator
		Facade	Memento
		Flyweight	Observer
		Proxy	State
			Strategy
			Visitor

در کتاب GoF، ۲۳ الگوی طراحی معرفی شده که این‌ها الگوهای طراحی اصلی هستند. این الگوها قابل تقسیم شدن به ۳ دسته اصلی هستند که هر دسته را به اختصار توضیح می‌دهیم.

الگوهای طراحی از نوع Creational : هدف این الگوها نمونه‌سازی از object ها می‌باشد به نحوی که reuse بالا برود تعداد این الگوها ۵ عدد است.

الگوهای طراحی از نوع Structural : هدف این الگوها جداسازی پیاده‌سازی از توصیف می‌باشد (implementation) از Interface (به طوری که هر کدام به صورت جداگانه قابل گسترش باشند).

الگوهای طراحی از نوع Behavioral : هدف این دسته از الگوها مدیریت رفتار object ها در زمان اجرا می‌باشد. همانطور که در شکل بالا هم ملاحظه می‌شود تعداد الگوهای این دسته بیشتر از دو دسته دیگر است.

می‌کنیم و هم رفتاری که object ها در زمان اجرا از خود نشان می‌دهند. عبارت های زیر هم مفهوم جمله قبل است.

- هم structure را مشخص می‌کنیم و هم collaboration بین object

- هم وجه استاتیک را مشخص می‌کنیم و هم وجه داینامیک.

- هم class diagram می‌کشیم و هم sequence diagram

- هم وجه ساختاری و هم وجه رفتاری.

بیایید یک بار دیگر الگو را تعریف کنیم تا به مفاهیم جدیدتری دست پیدا کنیم:

یک solution برای یک problem در یک context مشخص یا اگر بخواهیم دقیق‌تر تعریف کنیم یک راه حلی برای یک مساله تکرارشونده که جریات از آن حذف شده است و در حوزه ای خاص و مشخص از آن به دفعات فراوان استفاده شده است.

منظور از context ???

منظور از context محل یا فعالیتی در مهندسی نرم‌افزار context است که عمل مشخصی در آن انجام می‌شود. برای مثال الگویی که در طراحی تعریف می‌شود با context الگویی که در تحلیل یا تست یا بازاریابی انجام می‌شود متفاوت است. اگر یک الگو (pattern) را در غیر از حوزه (context) مشخص که در آن تعریف شده است به کار ببریم تبدیل به پادالگو (anti pattern) می‌شود. پس یک راه حل رایج! برای یک مساله تکرارشونده خاص که در عمل موفقیت‌آمیز نیست و به جای حل مساله آن را پیچیده‌تر می‌کند. (چون در حوزه‌ای غیر از حوزه اصلی خود به کار رفته است).

هدف الگوهای طراحی (مخصوصاً الگوهای کتاب GoF) بالا بردن flexibility است. آن‌ها این کار را از طریق کم کردن coupling در روابط بین موجودیت‌ها انجام می‌دهند. یک متريک نرم‌افزاری است که مشخص می‌کند چقدر دو مأژول یا دو تابع به هم وابسته هستند.

۳ اصل مهم در الگوهای طراحی

- تعریف Interface بین وابستگی‌ها (to an interface concrete)؛ این اصل می‌گوید که کلاس‌های (کلاس‌های دارای پیاده‌سازی) را به هم وصل نکنیم. آن‌ها را از طریق interface به هم وصل کنیم. به این اصل در مهندسی نرم‌افزار اصطلاحا Dependency Inversion Principle می‌گویند که به DIP معروف است.

- اولویت delegation به inheritance: این اصل می‌گوید delegation نسبت به inheritance که در حالت کلی استفاده از

همانطور که در شکل بالا هم ملاحظه می‌کنید برای الگوهای طراحی دسته بندی دیگری در نظر گرفته شده است که الگوها را به دو نوع object scope و class scope دسته بندی کرده است. الگوهای class scope که تعداد انها بسیار کم هست (چرا؟!) در زمان کامپایل محقق می‌شوند و by inheritance در زمان اجرا محقق می‌شوند و by delegation هستند. اما الگوهای object scope در زمان اجرا محقق می‌شوند و هستند. تا زمانی که ساختار object‌ها در زمان اجرا ساخته نشده است، الگو محقق نشده است.

الگوهای class scope علی رغم تعداد کمیان مفید و کاربردی می‌باشند اما همانطور که پیشتر اشاره شد هدف الگوهای طراحی GoF روی اصل Composite Resuse Principle می‌باشد و الگوهایی که by delegation پیاده‌سازی می‌شوند (الگوهای object scope) در اولویت هستند و تعدادشان هم بیشتر است.

الگوهای

طراحی در

مهندسی

نرم افزار

در این قسمت به بررسی دو الگوی طراحی دیگر می پردازیم: الگوهای Builder و Adapter هستند. الگوی Builder از گروه الگوهای Creational می باشد و الگوی Adapter زیرمجموعه الگوهای Structural قرار دارد.

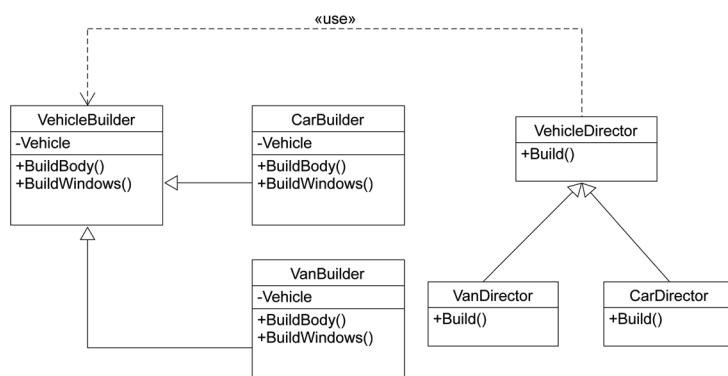
:Builder الگوی

هدف این الگو جدا کردن مسیر ساخت یک شی پیچیده از نحوه نمایش و ساخت آن است، به صورتی که با یک پروسه مشابه بتوان اشیای مختلفی ساخت.

برای مثال یک کارخانه ماشین سازی را در نظر بگیرید که هم ون و هم ماشین می سازد. اما مشخصاً روند ساخت هر دوی این ماشین آلات متفاوت هستند. برای مثال داخل ون ممکن است فضای زیادی برای بخش راننده و مسافر کنار آن باشد و همچنین یک بخش بزرگی برای ذخیره و کنارگذاشتن اشیا باشد، در حالی که یک ماشین سدان داری یک فضا برای نشستن مسافران و همچنین یک صندوق عقب باشد. به طور قطع نیز در فرآیند و جزئیات ساخت آنها تفاوت‌های زیادی وجود دارد.

الگوی Builder برای ساخت اشیای پیچیده، متدهای مورد نیاز برای ساخت آن را در کلاس‌های تحت عنوان Builder فراهم می‌کند و سپس با استفاده از یک کلاس Director ترتیب درست برای ساخت آن اشیا که متدهای آن در کلاس Builder است را فراهم می‌کند.

دیگر اگر زیر ساختار کلی این الگو را نشان می‌دهد.



برای شروع ابتدا به تعریف کلاس انتزاعی VehicleBuilder می‌پردازیم:

```

public abstract class VehicleBuilder{
    public virtual void BuildBody() {}
    public virtual void BuildBoot() {}
    public virtual void BuildChassis() {}
    public virtual void BuildPassengerArea() {}
    public virtual void
        BuildReinforcedStorageArea(){}
    public virtual void BuildWindows() {}
    public abstract IVehicle Vehicle {get;}
}
  
```

قسمت دوم - امیرراد کیمیابی

توجه کنید که متدهایی در کلاس CarBuilder وجود دارند (override شده‌اند) که در کلاس VanBuilder نیستند. زیرا مثلاً ون مدنظر ما بخش مسافر ندارد که برای آن تابع BuildPassengerArea قابل تعریف باشد، البته این موضوع بالعکس نیز صادق است. (مثلاً برای

تابع BuildReinforcedStorageArea

حال با استفاده از این دو تابع builder کلاس director می‌تواند شی مورد نظر خود را با فراخوانی درست و با ترتیب مشخص خود ایجاد کند.

VehicleDirector کلاس

```
public abstract class VehicleDirector{
    public abstract IVehicle Build(VehicleBuilder vc);
}
```

حال با توجه به این کلاس VanDirector و CarDirector، خاصی که مدنظرمان است می‌سازیم:

```
public abstract class VehicleDirector{
    public abstract IVehicle Build(VehicleBuilder vc);
}

public class CarDirector : VehicleDirector{
    public override IVehicle Build(VehicleBuilder builder){
        builder.BuildChassis();
        builder.BuildBody();
        builder.BuildPassengerArea();
        builder.BuildBoot();
        builder.BuildWindows();
        return builder.Vehicle;
    }
}

public class VanDirector : VehicleDirector{
    public override IVehicle Build(VehicleBuilder builder){
        builder.BuildChassis();
        builder.BuildBody();
        builder.BuildReinforcedStorageArea();
        builder.BuildWindows();
        return builder.Vehicle;
    }
}
```

حال با استفاده از کلاس director و builder می‌توانیم در روند اصلی برنامه هر نوع ماشین یا ون خاصی که خواستیم را بسازیم. فرض کنیم کلاسی با نام Saloon داریم که از کلاس AbstractCar ارث بری می‌کند. آنگاه:

```
AbstractCar car = new Saloon(new StandardEngine(1300));
VehicleBuilder builder = new CarBuilder(car);
VehicleDirector director = new CarDirector();
Vehicle v = director.Build(builder);
```

روند مدنظر الگوی builder در ساخت یک Saloon است.

Adapter الگوی

هدف این الگو انطباق کلاس با interface مختلف با یک دیگر است. با استفاده از این مسئله چندین کلاس می‌توانند

به این نکته حتماً توجه شود که تمامی متدهای ساخت ون و ماشین در همین کلاس تعریف می‌شود. و اینکه فیلد انتزاعی Vehicle هم برای دریافت ماشین مدنظر است.

حال کلاسهای Builder اصلی را بر اساس این کلاس انتزاعی به صورت زیر

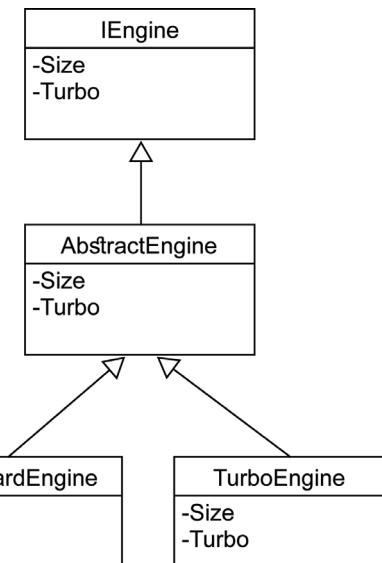
CarBuilder برای کلاس خواهیم داشت.

```
public class CarBuilder:VehicleBuilder{
    public AbstractCar carInProgress;
    public CarBuilder(AbstractCar car) {
        carInProgress=car;
    }
    public override void BuildBody(){
        Console.WriteLine("Building the body of the
car");
    }
    public override void BuildWindows(){
        Console.WriteLine("Building windows of the
car");
    }
    public override void BuildPassengerArea(){
        Console.WriteLine("Building Passenger area of the
car");
    }
    public override void BuildChassis(){
        Console.WriteLine("Building Chassis of the
car");
    }
    public override void BuildBoot(){
        Console.WriteLine("Building car boot");
    }
    public override IVehicle Vehicle{
        get{return carInProgress;}
    }
}
```

VanBuilder برای کلاس

```
public class VanBuilder:VehicleBuilder{
    public AbstractVan vanInProgress;
    public VanBuilder(AbstractVan van){
        vanInProgress=van;
    }
    public override void BuildBody(){
        Console.WriteLine("Building the body of the van");
    }
    public override void BuildWindows(){
        Console.WriteLine("Building windows of the van");
    }
    public override void BuildReinforcedStorageArea(){
        Console.WriteLine("Building the storage for the
van");
    }
    public override void BuildChassis(){
        Console.WriteLine("Building Chassis of the van");
    }
    public override IVehicle Vehicle{
        get{return vanInProgress;}
    }
}
```

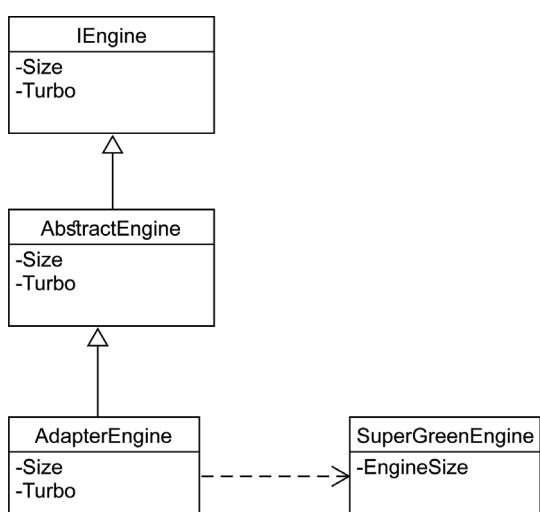
با هم کار کنند که در غیر این صورت به خاطر تفاوت interface که از `interface` بـلا استفاده نمی‌کند. (مثلـاً کلاسـی به نام `SuperGreenEngine`) و همچنین ما به کـد آن کلاس دسترسـی نداریـم کـه با ایجاد تغیـیرات در آـن، اـز `interface` مـورد نـظر استـفاده کـنیـم. در اـین صـورـت دیـگـر هـمـهـی مـوـتـورـهـای ماـشـینـی دـارـیـم. باـ فـرضـ کـنـیدـ کـهـ `interface` خـاصـیـ برـایـ مـوـتـورـهـایـ ماـشـینـیـ دـارـیـمـ. اـسـتـفـادـهـ اـزـ اـیـنـ `interface` شـرـکـتـ خـاصـیـ دـوـ نـوعـ مـدلـ مـوـتـورـ رـاـ دـرـ دـسـتـ تـولـیـدـ قـرـارـ دـادـهـ اـسـتـ. دیـاـگـرـامـ مقـابـلـ رـاـ درـ نـظـرـ بـگـیرـیدـ:



اینجـاستـ کـهـ الـگـوـیـ `Adapter` کـارـایـ خـودـ رـاـ نـشـانـ مـیـ دـهـدـ. اـیـنـ الـگـوـ سـعـیـ مـیـ کـنـدـ کـهـ بـاـ سـاخـتـ کـلاـسـیـ کـهـ بـاـ نـامـ `AbstractEngine` شـناـختـهـ مـیـ شـودـ بـاـ `interface` کـهـ شـناـختـهـ شـدـهـ اـسـتـ تـطـابـقـ پـیـداـ کـنـدـ. اـمـاـ اـیـنـ کـلاـسـ چـگـونـهـ عـمـلـ مـیـ کـنـدـ؟

کـلاـسـ `Adapter` اـزـ کـلاـسـ یـاـ آـنـ `interface` خـاصـیـ کـهـ مـیـ خـواـهـیـمـ بـاـ آـنـ منـطـبـقـ شـودـ اـرـثـ بـرـیـ مـیـ کـنـدـ (کـهـ بـهـ آـنـ `Adaptee` مـیـ گـوـینـدـ) وـ بـرـایـ اـنـطـبـاقـ کـلاـسـ جـدـیدـ کـهـ دـادـهـ شـدـهـ اـسـتـ (درـ اـینـجاـ منـظـورـ هـمـاـنـ مـوـتـورـ جـدـیدـ یـاـ `SuperGreenEngine` یـاـ `instance` یـکـ `reference` اـزـ آـنـ رـاـ درـیـافـتـ مـیـ کـنـدـ) کـهـ مـعـمـوـلـاـ اـیـنـ کـارـ درـ سـازـنـدـهـیـ کـلاـسـ `Adapter` اـنـجـامـ مـیـ شـودـ).

دیـاـگـرـامـ مقـابـلـ اـیـدـهـ رـاـ بـهـترـ بـرـرسـیـ مـیـ کـنـدـ:



با توضیـحـاتـ گـفـتهـ شـدـهـ،ـ یـکـ نـمـوـنـهـ کـوـتـاهـ اـزـ کـلاـسـ `Adapter` مـدـ نـظـرـ خـودـ رـاـ

بهـ صـورـتـ زـيرـ مـيـتوـانـيمـ بـنوـيـسيـمـ:

```

public class AdapterEngine:AbstractEngine{
    public AdapterEngine(SuperGreenEngine engine)
        :base(greenEngine.EngineSize, false)
    {
    }
}
  
```

با استـفادـهـ اـزـ هـمـيـنـ کـلاـسـ حـالـ مـيـتوـانـ يـكـ لـيـسـتـ اـيـجادـ کـردـ وـ عـلـاـوهـ بـرـ اـضـافـهـ کـرـدنـ مـوـتـورـهـایـ قـبـلـیـ،ـ مـوـتـورـ `SuperGreenEngine` رـاـ باـ اـسـتـفـادـهـ اـزـ يـكـ `AdapterEngine` instance اـضـافـهـ کـردـ.

با هـمـ کـارـ کـنـنـدـ کـهـ درـ غـيرـ اـينـ صـورـتـ بـهـ خـاطـرـ تـفـاوـتـ `interface` نـمـيـ توـانـسـتـندـ.

فـرضـ کـنـیدـ کـهـ `interface` خـاصـیـ بـرـایـ مـوـتـورـهـایـ ماـشـینـیـ دـارـیـمـ. اـسـتـفـادـهـ اـزـ اـیـنـ `interface` شـرـکـتـ خـاصـیـ دـوـ نـوعـ مـدلـ مـوـتـورـ رـاـ دـرـ دـسـتـ تـولـیـدـ قـرـارـ دـادـهـ اـسـتـ.

دـیـاـگـرـامـ مقـابـلـ رـاـ درـ نـظـرـ بـگـیرـیدـ:

اـگـرـ فـرضـ کـنـیدـ کـهـ کـلاـسـ `AbstractEngine` بـهـ صـورـتـ زـيرـ پـيـاـدهـ سـازـيـ شـدـهـ اـسـتـ:

```

public class AbstractEngine : IEngine{
    private int size;
    private bool turbo;
    public AbstractEngine(int size, bool turbo){
        this.size = size;
        this.turbo = turbo;
    }
    public virtual int Size{
        get{
            return size;
        }
    }
    public virtual bool Turbo{
        get{
            return turbo;
        }
    }
    public override ToString(){
        return this.GetType().Name+"("+size+")";
    }
}
  
```

درـ اـینـ صـورـتـ مـیـ تـوـانـ بـاـ اـیـجادـ يـكـ لـيـسـتـ اـزـ آـبـجـكـتـهـایـیـ اـزـ نـوعـ `AbstractEngine` وـ کـلاـسـهـایـیـ کـهـ اـزـ آـنـ اـرـثـ مـیـ بـرـزـدـ بـهـ تـمـامـیـ اـنـوـاعـ مـوـتـورـهـایـ دـاشـتـهـ بـاـشـیـمـ وـ مـثـلـاـ اـطـلـاعـاتـ آـنـهـاـ رـاـ بـاـ دـسـتـرـسـیـ بـهـ يـكـ مـقـدـیـکـسانـ چـاـپـ کـنـیـمـ.

اماـ فـرضـ کـنـیدـ کـهـ کـلاـسـیـ بـرـایـ مـوـتـورـ دـیـگـرـیـ سـاختـهـ شـدـهـ

Character classes

Useful Java classes & methods

Quantifiers

<code>[abc]</code>	matches a or b , or c .
<code>[^abc]</code>	negation, matches everything except a , b , or c .
<code>[a-c]</code>	range, matches a or b , or c .
<code>[a-c[f-h]]</code>	union, matches a , b , c , f , g , h .
<code>[a-c&&[b-c]]</code>	intersection, matches b or c .
<code>[a-c&&[^b-c]]</code>	subtraction, matches a .

Predefined character classes

<code>\d</code>	Any character.
<code>\d</code>	A digit: 0-9
<code>\D</code>	A non-digit: [^0-9]
<code>\s</code>	A whitespace character: [\t\n\x0B\f\r]
<code>\S</code>	A non-whitespace character: [^\s]
<code>\w</code>	A word character: [a-zA-Z_0-9]
<code>\W</code>	A non-word character: [^\w]

Boundary matches

<code>\A</code>	The beginning of a line.
<code>\G</code>	The end of the previous match.
<code>\Z</code>	The end of the input but for the final terminator, if any.
<code>\z</code>	The end of the input.

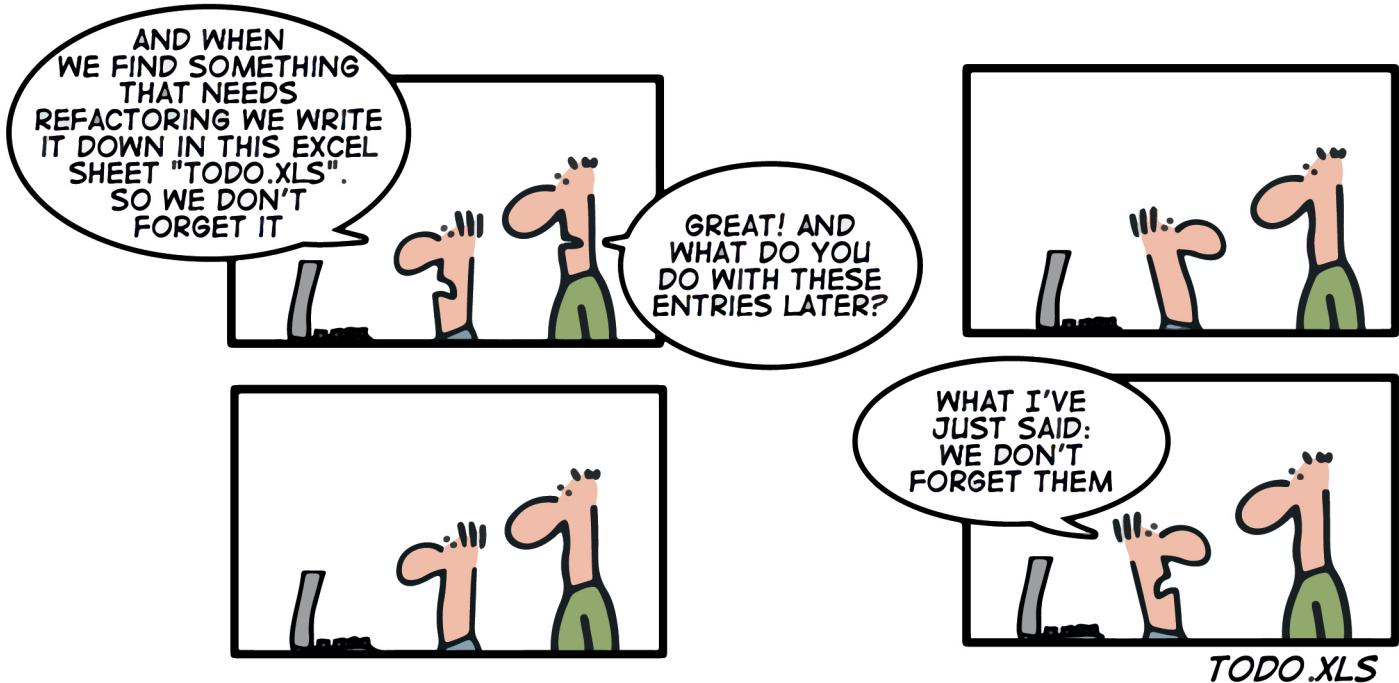
Pattern flags

<code>Pattern.CASE_INSENSITIVE</code>	- enables case-insensitive matching.
<code>Pattern.COMMENTS</code>	- whitespace and comments starting with <code>#</code> are ignored until the end of a line.
<code>Pattern.MULTILINE</code>	- one expression can match multiple lines.
<code>Pattern.UNIX_LINES</code>	- only the <code>\n</code> line terminator is recognized in the behavior of <code>,</code> , <code>&</code> , and <code>\$</code> .

Logical operations

<code>XY</code>	X then Y .
<code>X Y</code>	X or Y .

SIMPLY EXPLAINED



“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

– Martin Fowler

GRAPHIC DESIGN BY
AMIRHOSSEIN SAFARI

DESIGNED, PROGRAMMED AND PRODUCED BY
ALI OLIAYI AND MOHAMMADSINA KIAROSTAMI

MUSIC BY
ARON FARD

SAY HELLO TO MADNESS

MAD BALL



GAME CORNER
Find Your Dreams

 MADNESS
GAME STUDIO