طراحی شی گرا (object Oriented Design): یک شیوه تفکر (paradigm) در میان مدتی برای مدل سازی سیستم هاست.

سیستم شی گرا: مجموعه‌ای از object ها مدلسازی می‌شوند که در ارتباط با یکدیگر هدف تعریف شده مشترکی را برآورده می‌کنند.

مجموعه‌ای از مؤلفه‌های متصل که در ارتباط با یکدیگر هدفی تعریف شده را برآورده می‌کنند.

+ برنامه‌نویسی شی گرا (OOP) روشی برای پیاده‌سازی سیستم طراحی شده مبتنی بر شی گرایی است

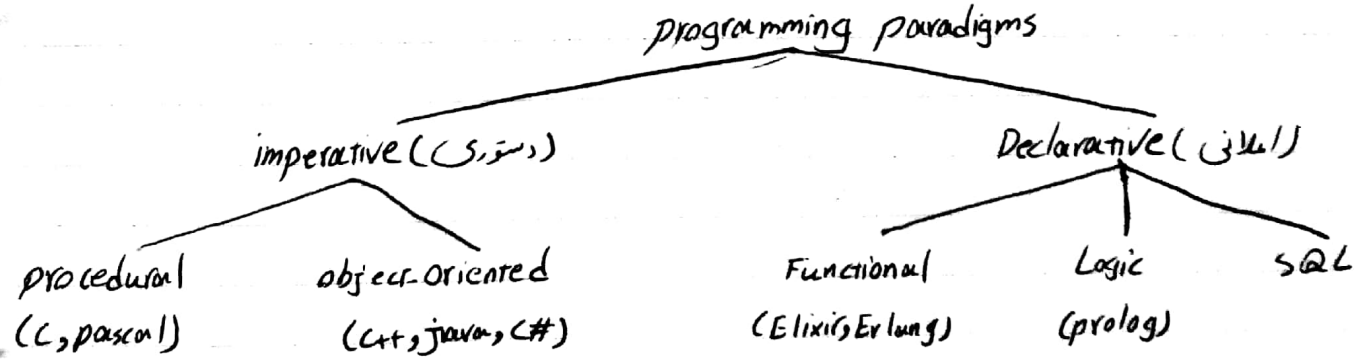
مدل: بیان دیدگاه انتزاعی که یک ناظر خاص نسبت به یک رویداد یا پدیده مشخص دارد.

نکته: مفهوم انتزاعی در تعریف مدل به این نکته اشاره دارد که ناظر براساس دیدگاه خود ممکن است تنها برخی از جزئیات سیستم را ببیند و برخی را نادیده بگیرد

انواع مدل: 1. فیزیکی (مانند ماکت) 2. نرم‌افزاری (مانند شبیه‌سازی ساختار DNA در نرم‌افزار)

3. ریاضی (مانند فرمول‌های ریاضی، روش‌های ریاضی)

شیوه تفکر در پیاده‌سازی:



برنامه‌نویسی دستوری (imperative) & هدف را مشخص می‌کنیم و راه رسیدن به آن را نیز می‌گوییم. از DFA استفاده می‌شود

برنامه‌نویسی اعلانی (Declarative): فقط هدف را برای کامپیوتر مشخص می‌کنیم

چرا نیاز به یک روش (شیوه) تفکر خاص در طراحی نرم‌افزار داریم؟ (چرا نیازمند OOD هستیم؟) 1. پیچیدگی ذاتی فرآیند طراحی

(نیاز به درک بهتر اجزا) 2. نیاز مکرر به تغییرات (تغییر و تکامل نیازهای کاربران) 3. نیاز به روش هماهنگ

در همکاری تیمی

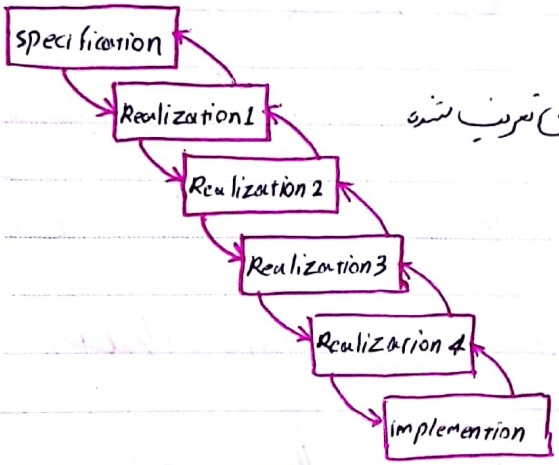
چگونه می‌توان یک سیستم پیچیده را مورد مطالعه و تحلیل قرار داد؟ تجزیه کردن (این کار را آنقدر ادامه می‌دهیم تا کوچکترین

حالت ممکن در بیاید و دید کافی به ما بدهد.

مراحل مطالعه ساختارهای پیچیده: 1. تجزیه به مؤلفه‌های کوچکتر 2. مطالعه هر مؤلفه به صورت مجزا

3. مطالعه نحوه ارتباطات مؤلفه‌ها با یکدیگر 4. تکرار مراحل بالا برای یک مؤلفه واحد تا رسیدن به درک مطلوب

فرآیند طراحی:



1. طراحی (Design): فرآیند بدست آوردن یک درک عمیق‌تر از نیازمندی‌های تعریف شده

برای یک سیستم (طراحی به ما می‌گوید چگونه اهداف تعیین شده سیستم را برآورده سازیم)

2. پیاده‌سازی (implementation): سطحی از طراحی که بالاترین

درک از نیازمندی‌ها حاصل می‌شود.

3. مشخصات (specification): مشخص کردن این که سیستم چه اهداف و چه نیازمندی‌هایی را باید برآورده سازد

4. وارسی (validation): اطمینان از این که طراحی انجام شده اهداف مشخص شده را برآورده می‌سازد

نکته: بیشترین بخش فرآیند طراحی را Realization تشکیل می‌دهد

MOHASSEL

Subject : نخ گرا

Year. Month. Date. ( )

نیازمندی ها: هر نیازمندی از دیدگاه یک ذی نفع بیان مطرح می شود - نیازمندی های کاربران و نیازمندی توسعه دهنده.

نیازهای مشتریان (کاربری ساده - سریع - ایمن - دسترسی پذیری بالا) - نیازهای توسعه دهندگان (مقیاس پذیری بالا - قابل نگهداری قابل انتقال - قابلیت استفاده مجدد) - نیاز هر گروه پس از طی زمان تغییر می کند

انواع نیازمندی ها:

1. نیازمندی های عملکردی: تعریف آن چه کار سیستم یا یک مؤلفه از سیستم باید انجام دهد - با مشخص کردن رابطه ای بین ورودی و خروجی انجام می شود - هر نیازمندی عملکردی معادل یک use case در نظر گرفته می شود - عموماً با این نیازمندی ها سروکار داریم

2. نیازمندی های غیر عملکردی: معیاری برای کیفیت انجام نیازمندی های عملکردی تعیین می کند - این دسته از نیازمندی ها معمولاً در حوزه معماری سیستم دیده می شود

نکته: سرعت در بازشدن یک سایت غیر عملکردی است ولی سرعت در reaction هوا پیما عملکردی است

معیارهای کیفی: کارایی - امنیت - دسترسی پذیری - مقیاس پذیری - قابلیت استفاده مجدد

نکته: یک نیازمندی خاص مانند امنیت ممکن است از یک دیدگاه یک عملکردی و از دیدگاه دیگر غیر عملکردی باشد

ساختار سلسله مراتبی (Hierachical structure): یک سیستم از چندین زیر سیستم مرتبط با یکدیگر تشکیل شده که هر زیر سیستم خود شامل چندین زیر سیستم است - امکان مشاهده و درک کل سیستم و اجزای کوچکتر سیستم - معماری یک سیستم تابعی از مؤلفه های سازنده آن به همراه روابط میان مؤلفه هاست

تفکیک مسئولیت ها (separation of concerns (SoC)): برای آن که بتوان سیستم را مطالعه کرد باید ماهیت تجزیه پذیری داشته باشد - هر پیمانه مسئولیت واحد دارد که عملکرد آن کمترین وابستگی را به پیمانه های دیگر دارد

واحد تفکیک شده از یکدیگر باید ۲ خاصیت مهم داشته باشند: 1. ارتباطات درون یک پیمانه قوی و زیاد 2. ارتباطات بین پیمانه‌های مختلف کم

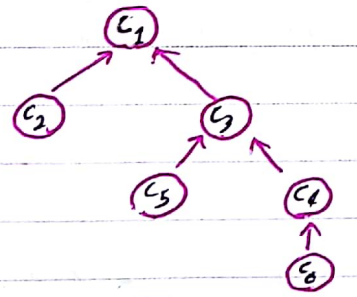
تجزیه‌پذیری: امکان تفکیک سیستم به پیمانه‌ها (Modules) تقریباً مجزا

بسی بودن مؤلفه‌های اولیه (Relative Primitives): تشخیص این که چه مؤلفه‌ای، مؤلفه اولیه برای سیستم است تا حد زیادی بستگی به دیدگاه ناظر به سیستم دارد.

نکته: مؤلفه‌ای که برای یک ناظر مؤلفه اولیه است ممکن است برای ناظر دیگر مؤلفه‌ای پیچیده با سطوح بالاتر از انتزاع باشد.

الگوهای مشترک: بسیاری از سیستم‌های پیچیده از مؤلفه‌های مشترک قابل استفاده مجدد تشکیل شده‌اند و در زیرسیستم‌ها شبیه به هم هستند

فرم‌های مشترک پایدار: (stable Intermediate Forms): با تکامل یک سیستم، کل یا بخشی از آن که پایدار است می‌تواند به عنوان مؤلفه‌ای اولیه سیستم دیگر تلقی شود. اگر بخواهیم از مؤلفه‌های اولیه استفاده نکنیم با مشکلات زیادی روبه‌رو می‌شویم



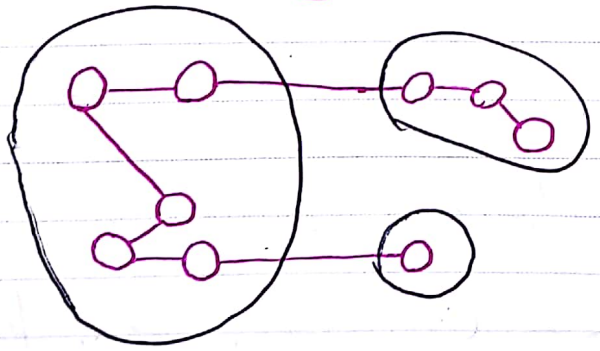
دو فرم کلی از تجزیه سیستم وجود دارد:

1. Is A (class structure): وراثتی - مثلاً

انسان نوعی از موجوده

2. Part Of (Object Structure): بخشی از

مثلاً کلیه بخشی از انسان است



Subject :
Year. Month. Date. ( )

قوانین تجزیه سیستم:

1. تجزیه الگوریتمی: سیستم براساس کارهایی که باید انجام شوند تجزیه می شوند - از خود می پرسیم چه کارهایی باید انجام شوند - هر مؤلفه سیستم یک قدم عمده در پیشبرد فرآیندهای سیستم را نشان می دهد - ترتیبی از انجام رویدادها را مشخص می کند

2. روش شیءگرا: سیستم براساس کارگزاران سیستم تجزیه می شود - از خود می پرسیم چه کسانی باید کارها را انجام دهند.

مفهوم طراحی: مدلی برای رسیدن از نیازمندی ها به پیاده سازی تعریف می شود.

هدف از طراحی، ساخت سیستمی است که: یک مجموعه نیازمندی های عملکردی تعیین شده را برآورده می سازد - با محدودیت های رسانه هدف تطابق دارد - نیازمندی های ضمنی و صریح در مورد کارایی و استفاده از منابع را برآورده می سازد - معیارهای فنی و صریح طراحی را در قالب مصنوعات (Artifacts) برآورده می سازد - محدودیت های فرآیند طراحی مانند هزینه و زمان یا ابزار انتخابی برای انجام طراحی را برآورده می سازد

متد: یک روال منظم برای تولید مجموعه ای از مدل ها که جنبه های مختلف یک سیستم در دست توسعه را با استفاده از یک روش نشانه گذاری خوش تعریف بیان می کند.

متدولوژی: مجموعه ای از متدها که در چرخه حیات فرآیند طراحی مورد استفاده قرار گرفته اند و فلسفه و فرآیند یکپارچه دارند

فصل 2

+ هر نسل از زبان های برنامه نویسی یک سطح انتزاع از زبان ماشین ارائه نموده اند

نسل اول: سطح انتزاع فرمول های ریاضی و محاسباتی بود مثل Algol58, Forman I

Subject : شیءگرا

Year. Month. Date. ( )

نسل دوم: ساختار انتزاعی الگوریتم‌ها بودند - کاربرد در تجارت - مثل Forman II و COBOL و LISP

نسل سوم: انتزاع داده - معرفی انواع داده - مثل PL/1 و Pascal و C

platform (سکو): مجموعه ابزارهای نرم‌افزاری و سخت‌افزاری که بستری برای اجرای برنامه فراهم می‌کند (اندروید، ویندوز، NET Framework و ...)

Framework (چارچوب): مجموعه‌ای از کتابخانه‌ها و ابزارها که بر بستر یک زبان برنامه‌نویسی ساختاری شکل‌گرفته و یک روش خاص در حل مسئله می‌دهد و با کم کردن استقلال ما حل مسئله را راحت‌تر می‌کند. (J2EE و AnguLarJS و Laravel و ASP.NET)

SDK (Software Development Kit): ابزارهایی که ساخت برنامه‌های Platform را تسهیل می‌کنند

تولد شیءگرایی نسل اول و اوایل نسل دوم: داده‌های عمومی - زیربرنامه‌های محلی - خطاهای منتشرشونده - خطاهای سخت

تولد شیءگرایی نسل دوم و اوایل نسل سوم: امکان ساخت زیربرنامه‌های تودرتو - پیدایش الگوهای طراحی

تولد شیءگرایی نسل سوم: نیاز به تولید نرم‌افزارهای بزرگتر و پیچیده‌تر - امکان توسعه مستقل ماژول‌ها - جداسازی داده و الگوریتم - عدم امکان تشخیص خطاهای نوع داده در زمان کامپایل

تولد شیءگرایی: ظهور زبان‌های StrongType - ظهور روش‌های طراحی داده‌محور - واحدسازنده کلاس‌ها و اشیاء به‌جای ماژول‌ها - شیء = داده + الگوریتم - حذف داده‌های عمومی (Global) - متغیر پذیری بالا

★ برنامه‌نویسی شیءگرا (OOP): روشی برای پیاده‌سازی نرم‌افزار که در آن برنامه‌ها به شکل مجموعه‌ای از اشیاء مرتبط مدل‌سازی می‌شوند - هر شیء نمونه‌ای متعلق به یک یا چند کلاس است و هر کلاس عضو یک رده از سلسله مراتب است

Subject :
Year. Month. Date. ( )

که مکانیزم رابطه وراثت تعریف می‌شوند.

+ هر شیء نمونه‌ای از یک کلاس است

رابطه‌ی Part Of : ساختار سازنده یا اشیاء هستند نه الگوریتم‌ها

رابطه‌ی IS A : کلاس‌ها طبق رابطه وراثت با یکدیگر ارتباط دارند

یک زبان شیءگرا است اگر و تنها اگر ۳ مشخصه زیر را برآورده سازد :

1. پشتیبانی از تعریف اشیاء (داده + الگوریتم) به طوری که داده از دید اشیاء دیگر مخفی بوده ولی الگوریتم‌ها قابلیت فراخوانی توسط اشیاء دیگر را داشته باشند (متدهای public و مشخصه‌های private)

2. هر شیء یک کلاس متناظر به خود داشته باشد

3. کلاس‌ها مشخصه‌های خود را طبق رابطه وراثت از کلاس والد (superclasses) به ارث می‌برند

* طراحی شیءگرا (OOD) : روش برای تجزیه سیستم‌های پیچیده بر مبنای اشیاء و یک سیستم نشانه‌گذاری برای نمایش مدل‌های فیزیکی و منطقی، ایستا و پویا برای طراحی - هر روشی که مبتنی بر تجزیه شیءگرا نبود، طراحی شیءگرا است.

برخلاف OOP واسطه یک زبان خاص نیست و نتیجه OOD را می‌توان در هر زبان OOP پیاده‌سازی کرد

تحلیل شیءگرا OOA : روش برای تحلیل کدنیازمندی‌ها از دیدگاه اشیاء و کلاس‌های قابل کشف در دامنه تعریف مسئله بررسی می‌کند

عناصر مدل شیءگرا :

۴ عنصر اصلی :

Subject :
Year. Month. Date. ( )

1. Abstraction

یک انتزاع مشخصه‌های اساسی یک شیء از دیدگاه یک ناظر مشخص را نشان می‌دهد که آن را از سایر اشیاء دیگر متمایز می‌سازد

دیدگاه ناظر: دیدگاهی که دیگران نسبت به یک چیز دارند

انواع انتزاع:

1.1. Entity Abstraction (انتزاع موجودیت): هر موجودیت در دامنه مسأله به یک موجودیت در فرم انتزاع نگاشت می‌شود

2.1. Action Abstraction (انتزاع عملیات): یک شیء حاوی مجموعه‌ای از عملیات مرتبط با یکدیگر و مناسب نوشتن در دنیای واقعی نیست مثل رمزنگاری

3.1. VM Abstraction (انتزاع ماشین مجازی): مجموعه‌ای از عملیات که تحت کنترل یک کاربر در نظر گرفته می‌شوند چند هدف ولی یکی آن را کنترل می‌کند

4.1 Coincidental Abstraction (انتزاع تصادفی): یک شیء حاوی مجموعه‌ای از عملیات که ارتباط مشخصی با یکدیگر ندارند.

client (مشتری، سرویس گیرنده): شیء‌ای که از منابع فراهم شده توسط شیء دیگر استفاده می‌کند

server (خدمت‌گذار، سرویس دهنده): شیء‌ای که منابع را برای استفاده client‌ها فراهم می‌کند

contract (قرارداد): مجموعه قوانینی که یک مشتری برای دریافت سرویس صحیح از خدمت‌گزار باید رعایت کند و هر قرارداد در ارتباط با یک سطح انتزاع تعریف می‌شود

Invariant (ثابت): یک عبارت منطقی (T و F) که T بودن آن در زمان مشخص باید حفظ شود

Precondition (پیش شرط): یک invariant که فرض یا مفروضات ابتدایی برای کار یا یک server را در نظر می‌گیرد

postcondition (پس شرط): یک invariant که فرض یا مفروضاتی در مورد صحت کار server در نظر می‌گیرد

Subject :
Year. Month. Date. ( )

signature (امضای عملیات): مشخصه نام، پارامترهای ورودی و مقدار بازگشتی یک عمل یا امضای آن عمل می‌گویند

```
int div (int a , int b)
    return a/b

int add (int a , int b)
    return a+b

float div (float a , float b)
    return a/b

string add (string a , string b)
    return a+b
```

مثال اسلاید 23

2. Encapsulation:

پنهان سازی جزئیات پیاده‌سازی یک شیء از دید کاربران

+ کپسوله‌سازی یک مفهوم کاملاً مرتبط با زمان پیاده‌سازی است در حالی که انتزاع هر ۳ مرحله تحلیل، طراحی و پیاده‌سازی کاربرد دارد.

تفاوت سطوح انتزاع و کپسوله‌سازی: سطوح انتزاع معمولاً نسبت به اشیاء مختلف است اما کپسوله‌سازی نسبت به رفتار پیاده‌سازی است

+ هر کلاس مطابق تعریف کپسوله‌سازی باید دارای دو بخش باشد:

1. یک رابط interface: روشی برای تعامل سایر اشیاء و کاربران با شیء فعلی فراهم کند

2. پیاده‌سازی Implementation: جزئیات نحوه فراهم سازی یک سرویس توسط یک شیء

مثال اسلاید 34

3. Modularity:

ماژول‌بندی عبارت است از تقسیم یک برنامه به بخش‌های کوچک‌تر خوش تعریف با مرزهای معین که می‌توانند به صورت مجزا کامپایل شوند.

Subject :
Year. Month. Date. ( )

خاصیت پیمانه‌ای خاصیتی از یک سیستم است که به مجموعه‌ای از مؤلفه‌های مستقل یا وابستگی کم تجزیه شده است.

ماژول‌ها حاوی مجموعه‌ای از کلاس‌ها هستند که یک ارتباط منطقی و معنایی با یکدیگر دارند.

مثال: یک گیت منطقی (AND و ...) یک واحد عملیاتی برای انجام محاسبات است. گیت‌ها معمولاً در قالب یک تراشه بسته‌بندی می‌شوند

مزایای پیمانه‌بندی: 1. امکان استفاده مجدد از ماژول‌ها 2. امکان تقسیم کار میان گروه‌های مختلف 3. کاهش زمان و هزینه توسعه 4. تست و نگهداری ساده‌تر

قواعد پیمانه‌بندی: تغییرات در پیاده‌سازی یک ماژول نباید مستلزم تغییر در پیاده‌سازی ماژول‌های دیگر، منجر به تغییر رفتار ماژول‌های دیگر و نیاز به دانش مشخص در مورد پیاده‌سازی سایر ماژول‌ها باشد.

+ با اعمال کپسوله‌سازی صحیح هزینه‌ی تغییرات در پیاده‌سازی یک ماژول به حداقل می‌رسد.

+ هزینه تغییرات در ماژول‌ها زیاد است

4. Hierarchy:

رده‌بندی سطوح مختلف انتزاع اطلاق می‌شود. که شامل ۲ نوع Is A و Part Of است

وراثت (Inheritance): یک نوع مهم از سلسله مراتب Is A است. در رابطه وراثت یک زیرکلاس subclass رفتار و ساختار خود را از یک یا چند سوپرکلاس Superclass به ارث می‌برد که ممکن است مواردی را به آن‌ها اضافه کند یا برخی را تغییر دهد

+ اگر کلاس B از نوع کلاس A نباشد نباید از آن ارث‌بری کند

Subject :
Year. Month. Date. ( )

+ با کشف کلاس های جدیدتر در حین طراحی می توان رفتارهای مشترک میان آن ها را به یک سوپر کلاس بالاتر منتقل نمود

انواع وراثت :

4-1. وراثت واحد (Single Inheritance) : هر زیرکلاس فقط و فقط از یک سوپرکلاس ارث بری می کند

مثال اسلاید 45

4-2. وراثت چندگانه (Multiple Inheritance) : هر زیرکلاس از بیش از یک کلاس ارث بری دارد.

مثال اسلاید 46

4-3. تجمع (Aggregation) : بیان گر رابطه ی Part Of است

مثال اسلاید 47

+ در سلسله مراتب IS A هر کلاس که جزئیات کمتری داشت انتزاع سطح بالا و هر کلاس که جزئیات بیشتری داشت انتزاع سطح پایین در نظر گرفته می شد

مسأله الماس اسلاید 48

کلاس های Mixin : کلاس است که متدهایی برای استفاده سایر کلاس ها فراهم می کند بدون اینکه نیاز باشد به عنوان سوپرکلاس معرفی شود.

غامض فرعی

5. Typing :

Subject :
Year. Month. Date. ( )

عموماً اشاره به انواع داده Data type دارد

نوع یک توصیف دقیق از مشخصات رفتاری و ساختاری مشترک میان اشیاء یک مجموعه است

یک زبان ممکن است نوع قوی (Strongly Typed)، نوع ضعیف (Weakly Typed) یا بدون نوع (UnTyped) باشند.

زبان های نوع قوی: تطابق اسامی در زبان فرآیندی یکنواخت الزامی است (نام، نوع داده، مقدار بازگشتی) -

بررسی نوع (Type checking) عموماً در زمان کامپایل انجام می شود

مشکل عمده ی زبان های قوی این است که وابستگی معنایی در روابط اشیاء ایجاد می کنند و حتی کمترین تغییرات در رابط اشیاء هزینه بر است.

نوع ایستا و پویا: در نوع ایستا مقدیرسازی در زمان کامپایل (Early Binding) و در انواع پویا مقدیرسازی تا زمان اجرا به تاخیر می افتد (Late Binding)

+ مفاهیم نوع قوی و ضعیف با ایستا و پویا متفاوت است ← مفاهیم نوع قوی و ضعیف اشاره به مقید ساختن یک نام (داده یا متد) به نوع مشخص دارد ولی مفاهیم ایستا و پویا به زمان ایجاد این مقدیرسازی اشاره دارد.

6. Concurrency:

همروندی: قابلیت تجزیه پذیری یک برنامه به زیر برنامه های مجزا و ترتیب اجرای آن ها برای رسیدن به هدف نهایی برنامه اشاره دارد

سؤال: آیا ترتیب اجرای این اجزای کوچکتر در نتیجه ی نهایی تاثیرگذار است؟ بله غیر (همروندی بی قید Unbound concurrency)

Subject :
Year. Month. Date. ( )

هم بلد (شرایط رقابتی (Race Condition))

سؤال: آیا می‌توان کاری کرد که در صورت اجرای با ترتیب دلخواه اجرای کدهایتان همواره یک نتیجه به دست آید؟ بله.

همگام‌سازی (Synchronization) ← باعث می‌شود وقتی داده به آن اضافه می‌شود کسی دیگری به آن دسترسی نداشته باشد

سؤال: دلیل به وجود آمدن شرایط رقابتی چیست؟

+ شرایط رقابتی ← تغییر نتیجه نهایی در صورت تغییر ترتیب اجرا است

جواب: وجود یک متغیر مشترک میان واحدهای اجرایی مختلف که هر واحد اجرایی برای تغییر آن متغیر مشترک (Mutable State) با دیگر واحدها رقابت می‌کند و از این می‌توان برنده نهایی این رقابت را تعیین نمود.

سؤال: چگونه شکل شرایط رقابتی را حل کنیم؟ با ۲ راهکار زیر

فقط راهکار ۱ مهم است

راهکار ۱: از متغیر مشترک به نحوی حفاظت کنیم که هر لحظه بیش از ۱ موجودیت به آن دسترسی نداشته باشد (انحصار متقابل (Mutual Exclusion)) - روش ← مکانیزم‌های قفل مانند سمافورها، مانیتورها، Mutex و ... - از این روش برای حل مسأله ناحیه بحرانی استفاده می‌شود

راهکار ۲: متغیر مشترک حذف شود - روش ← برنامه‌نویسی تابعی (Functional Programming) ← در این روش تابع نیز مانند انواع داده (int و ...) جزو انواع داده باید هستند و می‌توانیم یک تابع را به عنوان پارامتر به تابعی دیگر ارسال کنیم که در برنامه‌نویسی تابعی هیچ تعریف متغیری انجام نمی‌شود و همه مقادیر ثابت یا constant هستند

راهکار ۳: متغیر مشترک را حفظ کنیم اما آن را با اشیاء دیگر به اشتراک نگذاریم (Actor Model)

راهکار ۴: فرآیندها بدون اینکه اطلاعی از وجود یکدیگر داشته باشند از طریق کانال‌های عمومی به یکدیگر پیام بفرستند (Communicating Sequential Processes)

Subject :
Year. Month. Date. ( )

8 Persistence.6

هر شی بخشی از فضا را اشغال کرده و یک طول عمر مشخص دارد

طیف طول عمر اشیاء را میتوان در 6 مورد در نظر گرفت : 1. اشیاء موقتی حاصل از ارزیابی نتایج میانی عبارات (2+(6*3)) 2. اشیاء محلی فعال درون یک تابع 3. اشیاء عمومی معتبر در سراسر برنامه (متغیرهای global) 4. اشیاء معتبر میان اجراهای مختلف یک برنامه 5. اشیاء معتبر بین نسخه‌های مختلف یک برنامه 6. اشیاء معتبر خارج از چرخه حیات برنامه

1 تا 3 ← توسط برنامه‌نویس

4 تا 6 ← توسط پشتیبان پایگاه داده

+ به ابزارهایی برای تبدیل اشیای مورد نظر به عناصر قابل ضبط برای پایگاه داده داریم مانند ADO.NET و Nhibernate و Entity Framework و ... به عنوان پل میان زبان های برنامه‌نویسی و پایگاه داده هستند

فصل ۳

اشیاء (Objects) موجوداتی دارای حالت (State) و رفتار (Behavior) و هویت (Identity) است.

حالت یک شی (State) حالت یک شی توسط تمامی مشخصات آن شی به علاوه مقادیر فعلی این مشخصات تعریف می‌شود

مثال های اسلاید 3 و 4 و 5

+ یک تکنیک سودمند این است که حالت یک شی درون خودش کپسوله شود و به طور مستقیم توسط اشیاء دیگر قابل خواندن و نوشتن نباشد

+ خودش تصمیم می‌گیرد که چه نمایشی از حالت خود به اشیاء خارجی داشته باشد (متدهای get و set)

+ درصورت نیاز به ایجاد تغییرات در پیاده‌سازی عموماً فقط همان شیء نیاز به کامپایل مجدد دارد نه سایر اشیایی که از آن استفاده می‌کنند

رفتار Behavior: نحوه کنش و واکنش آن با اشیای دیگر از طریق تغییر حالت (state) و ارسال پیام را تعریف می‌کند - رفتار یک شیء فعالیت های بیرونی قابل مشاهده یک شیء است - تأثیر این رفتار دو طرفه است -

رفتار اشیاء درون توابعی داخل بدنه کلاس آن شیء تعریف می‌شود (عموما public) - نشانگر تابع تعیین رفتار آن شیء است

-- اشیاء با فراخوانی متدها (توابع) عضو یکدیگر با هم ارتباط برقرار می‌کنند message = operation

+ برخی از عملیات انجام شده روی شیء تأثیرات جانبی بر روی آن شیء داشته و موجب تغییر حالت آن می‌شوند

عملیات Operations: یک عملیات نشانگر سرویس ارائه شده از جانب یک سرویس دهنده (server) به یک مشتری (client) است

پنج نوع عملیات می‌باشد که توسط مشتری انجام می‌شود

اصلاح کننده (modifier): تغییر حالت یک شیء

انتخاب کننده (selector): دسترسی به حالت یک شیء بدون تغییر آن

تکرار کننده (Iterator): امکان پیمایش یک مجموعه از اعضای داخلی یک شیء را با ترتیب مشخص فراهم می‌کند

سازنده (constructor): عملیاتی که یک شیء جدید ایجاد کرده و حالت آن را مقداردهی اولیه می‌کند

مخرب (Destructor): عملیاتی که حالات یک شیء و خود شیء را از بین می‌برد

نقش ها: اگر مجموعه رفتارهای یک شیء را به دسته‌بندی های منطقی کوچکتری تقسیم کنیم، به هر شیء متناسب با تعداد این دسته‌بندی ها یک یا چند نقش Role تعلق می‌گیرد.

Subject : شیءگرا

Year. Month. Date. ( )

مسئولیت ها (Responsibility): تمامی سرویس هایی که یک شیء تحت قراردادهای معین به مشتریان خود ارائه می کند. — مجموعه رفتار و حالات یک شیء نقش آن شیء در سیستم را تعریف می کند که هر شیء متناسب با نقش خود مسئولیت هایی را به عهده دارد. مثال ۱۵

+ برخی نقش ها ممکن است با یکدیگر همپوشانی داشته باشند و برخی دیگر ناسازگار باشند.

+ هر شیء می تواند یک ماشین فعال یا غیرفعال باشد

+ اشیاء فعال نخ کنترلی مختص خود را داشته و به صورت خودمختار عمل می کند

سیستم مرکزی: یک سیستم هر روند سیستمی است که در آن واحد بیش از یک شیء فعال دارد

هویت (Identity): مشخصه ای از یک شیء که آن را از سایر اشیاء متمایز می کند. شامل اسامی متغیرها، توابع، اشیاء کلاس ها، کلیدهای اصلی در پایگاه داده و... - اگر هویت یک شیء این بود دیگر قابل بازیابی نیست. مثال ۱۹

روابط بین اشیاء: روابط بین اشیاء منطقی است که اشیاء مورد یکدیگر دارند شامل عملیاتی که هر یک می تواند بر روی دیگری انجام دهد. انواع روابط:

1. پیوند (Link): یک کانال اتصال معنوی یا فیزیکی بین اشیاء که از طریق آن به تبادل پیام میپردازند. تبادل پیام بسته به نقش اشیاء نسبت به یکدیگر ممکن است یک یا دو طرفه باشد

+ بسته به عملکرد یک شیء در یک پیوند هر شیء می تواند یکی از سه نقش زیر را داشته باشد

controller: عملیاتی را بر روی اشیای دیگر انجام می دهد اما اشیای دیگر قادر به انجام هیچ گونه عملیاتی روی آن نیستند

server: توسط اشیای دیگر مورد استفاده قرار می گیرد اما هیچ عملی را بر روی اشیای دیگر انجام نمی دهد

proxy (نماینده): هم عملیاتی برای استفاده توسط اشیای دیگر دارد و هم عملیاتی را خود بر روی اشیای دیگر انجام می‌دهد. اشیای

مثال ۲۵

استخراج شده از دامنه حقیقی مسأله معمولاً از این دست اشیا هستند

2. تجمع (Aggregate): پیوندها یک رابطه نظیر به نظیر یا server / client را میان اشیا تعریف می‌کردند - تجمع یک

رابطه کل / جزء که می‌توان از کل به جزء رسید - از یک شیء جزء تنها در صورتی می‌توان به شیء کل رسید که شیء جزء

ارجاعی به شیء کل را به عنوان یک حالت داخلی خود نگهداری کند

همکاری (synchronization): هرگاه دو شیء از طریق یک پیوند به تبادل پیام با یکدیگر بپردازند می‌گوییم اشیا با یکدیگر

همکار هستند - اشیای فعال هر یک نخ کنترلی مختص خود را دارند و صحت آن‌ها در حضور اشیای فعال

دیگر در سیستم تضمین می‌شود

اگر یک شیء فعال در پیوند با یک شیء غیرفعال باشد یکی از راهکارهای زیر برای همکاری نیاز است

1. ترتیبی (sequential): صحت معنایی شیء غیرفعال تنها در صورتی قابل تضمین است که

تنها یک شیء فعال (نخ اصلی) در سیستم وجود داشته باشد

2. محافظت شده (Guarded): صحت معنایی شیء غیرفعال در حضور چندین شیء فعال قابل تضمین

است اما اشیای فعال مشتری (client) باید خود با یکدیگر برای حل مسأله انحصار متقابل همکاری کنند.

3. همروند (concurrent): صحت معنایی شیء فعال در حضور چندین شیء فعال قابل تضمین است و شیء

سرور، خود انحصار متقابل را تضمین می‌کند

کلاس (class): مجموعه از اشیا که ساختار رفتار و معنای مشترکی با یکدیگر دارند - یک گروه، مجموعه یا گروه که

خواص مشترک دارند

رابط (Interface): رفتار بیرونی اشیای تعریف شده از یک کلاس به علاوه قراردادهای نحوه استفاده از آن کلاس را تعریف می‌کند. از این طریق می‌توان مجموعه رفتارها و خواص مشخص را برای یک کلاس اجبار نمود

+ یک interface شامل موارد زیر است: 1. اعلان متدهای انتزاعی (Abstract Methods) 2. تعریف سایر کلاس‌ها 3. ثوابت (constants) 4. استثناهای ممکن (Throws)

+ از یک interface نمی‌توان نمونه‌سازی انجام داد ⟸ interface هیچگونه متد constructor ندارد

کلاس‌های مجرد (Abstract): کلاس‌هایی که شامل متدها و مشخصه‌های عادی مانند سایر کلاس‌ها هستند اما از آن‌ها نمی‌توان به عنوان نمونه ساخته شود.

از این کلاس‌ها به عنوان کلاس‌های پایه در سلسله مراتب کلاس‌ها برای تعریف خصوصیات مشترک و بنیادی میان کلاس‌های مختلف استفاده می‌شود مثال ۳۲

متدهای مجرد (Abstract Methods): متدهایی هستند دارای نام، پارامترهای ورودی و خروجی بدون پیاده‌سازی

+ درون interface عموماً متدهای مجرد تعریف می‌شود مثال ۳۳

انواع سطوح دسترسی:

1. عمومی (public): متدها و متغیرهایی که برای تمامی مشتریان شامل دسترسی هستند.

2. خصوصی (private): متدها و متغیرهایی که تنها توسط اعضای داخلی خود کلاس شامل دسترسی هستند.

3. محافظت شده (protected): متدها و متغیرهایی که تنها توسط اعضای داخلی کلاس و زیرکلاس‌های این کلاس

MOHASSEL

Subject :
Year. Month. Date. ( )

قابل دسترسی هستند.

4. Package: متدها و مقادیری که تنها توسط کلاس های موجود در یک پکیج (ماژول) قابل دسترسی هستند.

روابط بین کلاس ها:

1. Generalization / specialization (IS A Relationship)

2. whole / part (part of Relationship)

3. Association: یک رابطه منطقی بین کلاس های غیرمرتبط - ضعیف ترین رابطه معنایی -

+ روابط انجمنی معمولاً دو طرفه در نظر گرفته می شوند مگر اینکه صریحاً قید شود. مثال ۳۷

درجات ← یک به یک - یک به چند - چند به چند

چند ریختی (polymorphism): استفاده از یک نام واحد برای اشاره به چندین رفتار یا عملیات مختلف

تفاوت بین عملیات مختلف با نام یکسان تنها بر اساس انواع ورودی و خروجی آن ها تعیین می شود

۳ نوع چند ریختی:

1. چند ریختی موردی (Ad hoc Polymorphism): اشاره به توابع هم نام اما با نوع پارامترهای دریافتی متفاوت دارد که هر تابع بر اساس نوع پارامتر دریافتی رفتار متفاوتی از خود نشان می دهد

2. چند ریختی پارامتری (Parametric Polymorphism): این امکان را فراهم می کند که بتوانیم برای عملیات بر روی انواع داده مختلف یک تابع واحد بنویسیم - مفهوم چند ریختی پارامتری هم به انواع داده و هم به متدها اعمال می شود

مثال ۴۱

Subject :

Year. Month. Date. ( )

Generics: یک متد Generic در زبان جاوا می‌تواند بسته به نوع داده پارامترهای ورودی، خروجی مناسب تولید کند.

3. زیرنوع‌ها (subtyping): در این حالت یک تابع به گونه‌ای نوشته می‌شود که یک نوع پایه T به عنوان ورودی دریافت می‌کند و علاوه بر آن زیرنوع‌های آن مانند S را نیز پذیرفته و به آن‌ها پاسخ مناسب می‌دهد