اصول طراحی در مهندسی نرم افزار:

* قانون Kiss (Keep It Simple, Stuid):
  + کدت رو ساده نگه‌دار و زیاد پیچیدش نکن.
  + کد باید تا حد امکان ساده نوشته بشه تا راحت فهمیده، تست، نگهداری و گسترش داده بشه.
  + توابع کوچک داشته باش. هر تابع باید فقط یک کار انجام بده.
  + از شرط‌های تودرتو پرهیز کن و شرط‌های ساده و واضح بنویس.
  + از متغیر اضافی پرهیز کن. متغیر اضافی پیچیدگی اضافی بهت تحمیل می‌کنه.
  + کد ساده‌تر تست پذیری راحت‌تری هم داره.
  + نام گذاری ها واضح باشن. اسم خوب خودش نصف توضیحاته.
  + در کل بدون هر کسی میتونه کد پیچیده‌ای بنویسه که کار میکنه مهم اینه کدی بنویسی که هم ساده باشه هم کار کنه.
* قانون DRY (Don’t Repeat Yourself):
  + از تکرار کد خودداری کن.
  + تکرار = دردسر در نگهداری
  + در کل بهتره یک تابع یا کلاس مشترک بسازی
* قانون YAGNI (You Are’nt Gonna Need It)":
  + چیزی رو که الان نیاز نداری اضافه نکن.
  + این که بگی شاید بعدا لازم شد = کد اضافی = پیچیدگی بیشتر
  + فقط چیزی رو بساز که واقعا لازم داری
* قانون TDA (Tell, Don’t Ask):
  + به جای این که از شی چیزی بپرسی و بعد تصمیم بگیری، تصمیم رو به خودش بسپار و ازش کاری که میخوای انجام بده رو بخواه.
  + یه مثال:

if (car.getFuelLevel() > 0) {

car.drive();

} //Wrong way

car.drive(); //Correct way

* قانون SOLID (Single responsibility, Opne/Closed, Liskov substitution, Interface Segregation, Dependency Injection):
  + S: Single Responsibility هر کلاس باید فقط یک وظیفه داشته باشد. مثال احراز هویت و ریپازیتوری در یک کلاس
  + O: Open/Closed کلاس‌ها باید برای توسعه باز و برای تغییر بسته باشند. مثال مستطیل و اضافه شدن دایره برای محاسبه مساحت.
  + L: Liskov Substitution اشیاء فرزند باید قابل جایگزینی با والد باشند. مثال پرنده و پنگوئن.
  + I: Interface Segregation نباید مجبور به پیاده‌سازی متدهایی شویم که نیاز نداریم. مثال کارگر که هم ربات پیاده ش کنه هم انسان (کار کردن و خوردن)
  + D: Dependency Inversion وابستگی‌ها باید به abstraction باشد، نه پیاده‌سازی. مثال وابستگی سوییچ به لامپ. باید به دیوایس وابسته باشه.
* قانون GRASP (General Reponsibility Assignment Software Pattern):
  + این قوانین الگوهای کلی برای واگذاری مسئولیت‌ها در طراحی شی گرا هستند.
  + این اصول به ما کمک میکنند تصمیم بگیریم کدام کلاس/شی باید چه مسئولیتی داشته باشد.
  + درواقع SOLID میگه چطور طراحی کنیم و GRASP میگه چه کسی (شی/کلاس) چیکار کنه.
  + شامل موارد زیر هست:
    - Information Expert: کلاسی مسئولیت رو بگیره که بیشترین اطلاعات مرتبط رو داره. مثلا کلاس Invoice باید جمع قیمت آیتم هارو محاسبه کنه چون خودش به آیتم ها دسترسی داره.
    - Creator: کلاسی شیء جدید بسازی که با اون رابطه نزدیک داره. مثلا کلاس Order بیاد OrderItem بسازه چون هر Order شامل OrderItemها هست.
    - Controller: کلاس بین UI و جدول هست که ورودی‌ها رو کنترل میکنه. مثلا در MVC میتونیم OrderController داشته باشیم.
    - Low Coupling: وابستگی بین کلاس‌ها باید تا جای ممکن کم باشه. مثلا به جای این که کلاس A مستقیما کلاس B رو بسازی، از Factory یا Dependenci Injection استفاده کنه.
    - High Cohision: هر کلاسی باید یه کار مشخص تمیز داشته باشه. کلاسی که کارهای پراکنده میکنه قابل نگهداری نیست.
    - Polymorphism: به معنای داشتن رفتار متفاوت با استفاه از Interface هست. مثلا میتونیم از Animal به عنوان اینترفیس استفاده کنیم که تابع Speak داره و بعد سگ و گربه اون رو پیاده میکنند.
    - Pure Fabrication: یک کلاس مصنوعی بساز فقط برای رعایت بقیه اصول. مثلا repository برای جدا کردن دسترسی به دیتابیس در مدل اصلی.
    - Indirection: ایجاد یک واسطه برای کاهش وابستگی سیستم. مثلا استفاده از سرویس یا اینترفیس بین Controller و Data Access
    - Protected Variations: نقاطی که ممکنه تغییر کنند رو با abstraction بپوشون. مثلا استفاده از اینترفیس برای لایه دیتابیس چون ممکنه نوع دیتابیس عوض بشه.پ

ORMها یا Object Relational Mapping:

* تبدیل خودکار بین اشیاء برنامه (Objects) و جداول پایگاه داده (Relational Table) را انجام می‌دهد.
* یک ابزار یا به عبارتی یک کتابخانه است که کمک می‌کند به جای نوشتن مستقیم SQL خام، از طریق کلاس‌ها و اشیاء جاوا با دیتابیس کار کنیم.
* معروف تریم ORM در جاوا Hibernate است.
* JPA هم یک استاندارد جاوا برای ORM است و هایبرنیت به عبارتی یکی از پیاده سازی های JPA است.
* Spring Data JPA هم یک ابسترکت بالاتر روی JPA است و برای راحتی بیشتر است. کار راه راحت تر و امن تر کرده و قابلیت نگهداری را افزایش می دهد اما برای کوئری های پیچیده بهتر است از SQL استفاده شود.
* انواع کارکردها و نیازهاشون:
  + برای CRUD ساده و معمولی: ORM عالیه
  + برای پروژه های بزرگ و تیمی: ORM برای نگهداری بهتره
  + در صورت نیاز به پرفورمنس بالا: SQL خام در برخی بخش‌های خاص بهتره
  + برای گزارش‌گیری سنگین و پیچیده: SQL یا Stored Procedure مفید تره
    - نمونه SP:

@NamedStoredProcedureQuery(

name = "getEmployeesByDepartment",

procedureName = "GET\_EMPLOYEES\_BY\_DEPT",

parameters = {

@StoredProcedureParameter(mode = ParameterMode.IN, name = "deptId", type = Integer.class)

},

resultClasses = Employee.class

)

@Entity

public class Employee {

// ...

}

و سپس در کد:

StoredProcedureQuery query = entityManager

.createNamedStoredProcedureQuery("getEmployeesByDepartment")

.setParameter("deptId", 10);

List<Employee> employees = query.getResultList();

* Annotationهای پایه در هایبرنیت برای نگاشت کلاس به جدول:
  + @Entity: کلاس نماینده یک جدول است.
  + @Table: نام جدول رو در دیتابیس مشخص میکند.
  + @Id: مشخص کننده کلید اصلی یا همان Primary Key است.
  + @GeneratedValue: نحوه تولید مقدار کلید اصلی را مشخص می‌کند.
  + @Column: نام یک ستون خاص در جدول را مشخص میکند.
* Annotationهای پایه برای نگاشت روابط بین جداول (ارتباط بین کلاس‌ها):
  + برای روابط یک به یک، یک به چند و چند به یک @JoinColumn که ستون کلید خارجی را مشخص میکند.
  + برای روابط چند به چنتد @JoinTable که جدول واسط را مشخص میکند.
* Annotationهای مربوط به ویژگی ستون‌ها:
  + @Column مشخص می‌کند که یک فیلد به یک ستون در جدول پایگاه داده نگاشت شود. قابلیت تنظیم ویژگی‌هایی مثل name, nullable, length, unique و columnDefinition را دارد.

@Column(name = "first\_name", nullable = false, length = 50)

private String firstName;

* + @Temporal فقط برای فیلدهای نوع java.util.Date یا Calendar استفاده می‌شود. تعیین می‌کند که کدام بخش از زمان (تاریخ/ساعت/هر دو) در پایگاه داده ذخیره شود. مقادیر ممکن: TemporalType.DATE, TemporalType.TIME, TemporalType.TIMESTAMP.

@Temporal(TemporalType.DATE)

private Date birthDate;

* + @Lob برای فیلدهایی با حجم بالا استفاده می‌شود (مثل متن‌های طولانی یا داده‌های باینری). متن بزرگ (CLOB) یا داده باینری بزرگ (BLOB) را مشخص می‌کند.

@Lob

private String longText;

@Lob

private byte[] imageData;

* + @Transient این فیلد در پایگاه داده ذخیره نمی‌شود. برای داده‌هایی استفاده می‌شود که فقط در حافظه نگهداری می‌شوند (مثلاً برای محاسبات موقت). مشابه transient در جاوا ولی مخصوص JPA است.

@Transient

private String tempDisplayName;

* نحوه کار به این صورته که زمانی که هایبرنیت شروع به کار میکنه، کلاس هایی که انوتیشن @Entity دارند رو شناسایی میکنه . براشون جدول معادل در دیتابیس ایجاد میکنه (در صورت استفاده از auto-schema-generator) اطلاعات موجود در جدول رو به شیء جاوا و برعکس نگاشت میکنه.

مقایسه کلید اولیه یا Primary Key با کلید خارجی یا Foreign Key:

| ویژگی | PK | FK |
| --- | --- | --- |
| تعریف | شناسه یکتای ردیف جدول | ایجاد ارتباط بین جداول |
| یکتا بودن | باید یکتا باشد | می تواند تکراری باشد |
| Nullable بودن | خیر | ممکن است (بسته به نیاز) |
| تعداد در جدول | فقط یکی | می تواند چندتا باشد |
| وابستگی | مستقل است | به جدول دیگر وابسته است |

احراز هویت یا Authentication و سطح دسترسی یا Authorization:

| مفاهیم | Authentication | Authorization |
| --- | --- | --- |
| معنی | احراز هویت | مجوز دسترسی |
| سوال اصلی | تو کی هستی؟ | اجازه دسترسی به چه چیزایی رو داری؟ |
| هدف | تأیید هویت کاربر | تعیین سطح دسترسی |
| ترتیب | اول انجام میشه | بعد از احراز هویت انجام میشه |
| مثال | وارد کردن نام کاربری و گذرواژه | اجازه دیدن، ویرایش یا حذف اطلاعات |

* روش های رایج احراز هویت
  + User/Pass
  + ایمیل یا پیامک (OTP)
  + اثر انگشت یا تشخیص چهره
  + Token (مثل JWT)
* یک روال معمول در احراز هویت و تعیین دسترسی:
  + APIGateway 🡪 Auth Service (login/jwt/token validation) 🡪 Policy Service (roles/permissions/rules) 🡪 Other microservice check access
* دو نوع کنترل دسترسی:
  + RBAC یا Role-Based Access Control: کنترل دسترسی براساس نقش اتفاق می افتد. مثلا نقش Admin اجازه حذف یک داده را دارد ولی نقش User ندارد.
  + ABAC یا Attribute-Based Access Control: کنترل دسترسی براساس ویژگی اتفاق می افتد. مثلا اگر دپارتمان یک یوزر منابع انسانی بود، می تواند در بازه ساعت 8 تا 10 تغییراتی را اعمال کند.
* Json Web Token یا JWT
  + یک توکن سبک، رمزنگاری شده و قابل حمل که اطلاعات عضویت و دسترسی کاربران را در خود نگهداری میکند. کاربرد اصلی JWT در احراز هویت در اپلیکیشنهای مختلف مانند وب، موبایل و APIهای Restful است.
  + سه بخش دارد:
    - Header که نوع توکن و الگوریتم های رمزنگاری را مشخص میکند.
    - Payload
    - Signiture
  + برخی از مزایای JWT:
    - بی نیاز اس سشن است. به این معنا که اطلاعات داخل خود توکن ذخیره می شود و نیازی به نگهداری سشن در سمت سرور نیست و اصطلاحا Stateless است (در مقابل Stateful)
    - مقیاس‌پذیری را بیشتر میکند چون سرورها سشن ذخیره نمیکنند و به راحتی میشه چند سرور همزمان داشت.
    - امن است. اگر به درستی و با استفاده از HTTPS و کلید محرمانه قوی پیاده شود، امنیت بسیار بالایی دارد.
    - در APIها چون Stateless هستند به راحتی قابل استفاده هستند
  + نکات امنیتی:
    - همیشه از HTTPs استفاده کن
    - توکنها رو در HTTP Only Cookie یا Secure Storage قرار بده
    - زمان انقضا یا Expiration کوتاه تنظیم کن و از رفرش توکن برای تجدید توکن استفاده کن

انواع مدیریت پروژه های نرم افزاری

* Waterfall: فرایند آبشاری یک فرایند خطی و مرحله ای است. هر مرحله (تحلیل، پیاده سازی و آزمون) باید به طور کامل تمام شود تا به مرحله بعدی برود. در این روش تغییر در مرحله قبل سخت و هزینه بر است.
* Agile: فرایند چابک فرایندی تدریجی و تکرارشونده است. پروژه به بخش های کوچکی به نام اسپرینت تقسیم می شود که مثلا به طول دو یا سه هفته هستند. در این روش، در هر چرخه بازخورد مشتری دریافت شده و قابلیت تغییر سریع در پروژه وجود دارد.

Scrum:

* نقش‌های اصلی در اسکرام:
  + Product Owner یا PO: مالک محصول است و نقش تجاری یا Business Role محسوب می شود. PO درواقع نماینده مشتری و ذینفعان است و مشخص می‌کند چه چیزی ساخته شود (What to build). مالک محصول مدیریت و اولویت بندی وظایف مربوط به پروژه را انجام می دهد و Backlog محصول را به روز نگه میدارد. بر ارزش محصول و نیاز بازار تمرکز دارد. مثلا ممکن است تصمیم بگیرد که اول سیستم پرداخت آنلاین توسعه داده شود چون ارزش تجاری بیشتری دارد.
  + در کنار PO تیم فنی را داریم که شامل نقش فنی یا Technical Role می شود. این تیم تیم توسعه است. یک تیم خودگردان که وظیفه اش ساختن محصول است. این تیم شامل برنامه نویس، طراح، تستر و DevOps است. این تیم تصمیم می‌گیرد محصول چگونه ساخته شود (How to build). در هر اسپرینت، راهبر فنی و تیم فنی آیتم‌هایی رو از Backlog انتخاب می کنند و توسعه می ‌دهند. مثلا این تیم در بخش پرداخت تصمیم می‌گیره پروژه با استفاده از Spring Boot و به صورت NoSQL نوشته بشه.
  + در کنار این ها نقش Scrum Master رو داریم که نقش خدمت گذار یا تسهیل گرد رو داره. این نقش محافظ چارچوب اسکرام هست و کمک میکند فرآیندها کاملا روان و طبق اصول اسکرام پیش بروند و موانع را برای تیم برطرف میکند. این فرد تسهیل کننده جلسات اسکرام مانند Dayli, Planning, Review, Grooming, Retro است. همچنین مسول افزایش بهره وری، سلامت تیم و همکاری با افراد است و تحلیل می‌کند عملکرد تیم چگونه بوده و با شناسایی مشکلات، فرایند‌ها را بهبود می دهد.
  + یک تیم DevOps هم داریم که در پروژه‌های بزرگ باعث کاهش زمان انتشار، افزایش کیفیت، کاهش ریسک و افزایش بهره وری تیم ها میشود. این تیم با کاهش شکاف بین تیم توسعه (Dev) و تیم عملیات (Ops) باعث این دو تیم را باهم یکپارچخ کرده و کندی مربوط به فاصله آن‌ها را از بین می برد.

CI/CD:

* فرض کنیم یک تیم برنامه نویسی دارند روی یک پروژه کار میکنند. ممکنه یک توسعه دهنده یک باگ در کدش داشته باشد. ممکن است کسی فایلی را به اشتباه پاک کرده باشد. حال در صورتی که این موارد دستی مرج بشوند یا دستی تست بشوند و بعد روی سرور کپی بشوند، احتمال خطا بالا می رود. در این نقطه CI/CD وارد می شود.
* در CI/CD وقتی یک توسعه دهنده کد خود را روی گیت قرار می دهد، یک سیستم خودکار این کد رو بیلد می کند، تست های آن را اجرا می کند و در صورت نیاز آن را روی محیط آزمایشی یا حتی نهایی مستقر می کند (Deploy). حال در ادامه هر بخش مجزا توضیح داده می شود.
* CI به معنای Continuous Integration یا یکپارچگی مستمر است. یعنی هربار کسی کد جدیدی زد و برای مثال روی برنچ دولوپ خورد، چک می کند آیا این کد سالم و قابل اجرا هست یا نه. از فواید آن می توان به شناسایی مشکلات و ترکیب امن تر کدهای مختلف اشاره کرد.
* CD به معناری Contiuous Delivery/Deployment یا تحویل/استقرار مستمر است. در تحویل مستمر، بعد از تست موفق سیستم آماده است تا استقرار یابد و این کار با تأیید دستی انجام می شود. اما در استقرار مستمر، بعد از تست موفق، سیستم به صورت خودکار مستقر می شود و روی سرور واقعی (Production Server) قرار می گیرد.
* با این تفاسیر، مراحل مختلف CI/CD رو میشه به این ترتیب در نظر گرفت: برنامه نویسی، کامیت، بیلد، تست، استقرار. تمامی مراحل کامیت، بیلد، تست و استقرار می توانند خودکار باشند و از طریق ابزارهایی مثل Jenkins این خودکار بودن محقق می شود.
* مزایای استفاده از CI/CD شامل موارد زیر است:
  + افزایش سرعت انتشار: به جای انتشار ماهانه، می توان روزانه یا حتی ساعتی نسخه زد.
  + کاهش خطای انسانی: از آنجایی که همه چیز توسط ماشین انجام می شود، خطای انسانی به شدت کاهش می یابد.
  + اطمینان از کیفیت: از طریق تست خودکار، باگ ها سریعتر کشف و رفع می شوند.
  + فرآیند تکرارپذیر و پایدار: با هربار تغییر در کد، یک فرآیند یکسان با همان ترتیب و روند قبلی اجرا می شود.
  + واکنش سریع به خطاها: اگر مشکلی پیش بیاید و خطایی کشف شود، از طریق همین فرآیند با سرعت کشف و رفع می شود.

Docker:

* یک پلتفرم یا سکوی سریع و سبک برای بسته بندی، اجرا و انتقال نرم‌افزار در یک محیط ایزوله به نام Container است. درواقع نرم افزار ما تبدیل به یک بسته به نام Docker Image می شود که می توان از این بسته بی نهایت Docker Container ایجاد کرد که در زمان اجرا می توانند با یک آیپی و پورت در دسترس باشند.
* دارای ویژگی‌ها و مزایای زیر است:
  + حل مشکل «تو سیستم من که کار می‌کنه!»: گاهی ممکن است شرایطی باشد که یک نرم افزار روی سیستم توسعه دهنده کار کند اما روی سیستم همکارش نه. داکر کل محیط اجرا (کد+lib+تنظیمات) را در یک بسته قابل حمل تجمیع می‌کند و این بسته روی هر سیستمی به یک شکل کار خواهد کرد.
  + محیط توسعه یکسان برای همه: هر توسعه دهنده با اجرای یک دستور مثل docker compose up دقیقا همان محیطی را خواهد داشته که سایر اعضای تیم هم دارند (مثلا با همون نسخه جاوا و دیتابیس).
  + استفاده آسان از سرویس‌های جانبی: همانطور که نرم افزار ما در قالب یک Image بیلد شده و سپس در قالب Container قابل اجرا است، تمامی سرویس های جانبی هم نسخه داکرایز شده دارند. مثلا استفاده از Redis یا Kafka دیگر پیچیده نیست و می توان با یک خط کامند مانند docker run postgres یک دیتابیس Postgres بالا آورد.
  + استقرار آسان: می توان هر اپلیکیشن را به صورت یک docker image درآورد و در هر سرور یا حتی روی کلاد اجرا کرد (در قالب docker container). تنظیمات پیچیده ای هم نخواهد داشت.
  + اجرای تست‌های خودکار: در Docker، می توان از CI/CD برای اجرای Integration/Unit Test در محیط‌های ایزوله و کنترل شده استفاده کرد.
  + ساخت میکروسرویس‌ها: در محیط‌های میکروسرویسی، هر سرویس با کانتینر خود (شامل زبان، پورت، پلتفرم اجرا و دیتابیس) اجرا می شود.
  + نسخه بندی محیط ها: در داکر می توان نسخه ای خاص از یک نرم افزار خاص را در تنظیمات مشخص کرد (مثلا فقط جاوا 8) تا هر بار که پروژه اجرا می شود، دقیقا با همان نسخه ها اجرا شود.
* در کنار Docker Image که در دیسک ذخیره میشه و از بین نمیره و Docker Container که در رم قرار داره و در صورت توقف دیتای اون از بین میره (مربوط به زمان اجرا است) یک Docker Volume هم داریم که می توان برخی اطلاعات مربوط به یک کانتینر را در آن ذخیره کرد تا در صورت توقف یا حذف آن، بتوان با بالا آمدن دوباره همان کانتینر یا یک مورد جدید، آن دیتا را داشت.