BACKEND ÖDEVLERİ VE CEVAPLARI

1. Compiler Nedir? (Derleyici)

Compiler (derleyici), yüksek seviyeli bir dilin girdisini alarak düşük seviyeli bir dilin çıktısını (bir derleme veya makine dili gibi) üreten bir çevirmendir. Temel olarak, bir programlama dilinde yazılmış kodları makine koduna dönüştürmek için kullanılan bir bilgisayar programıdır (bir bilgisayar işlemcisinin anlaması için insan tarafından okunabilen bir ikili 0 ve 1 bit diline kod). Bilgisayar daha sonra ilgili görevleri gerçekleştirmek için makine kodunu işler.

a. Derleyiciler her türlü hatayı, sınırı ve aralığı kontrol eder. Böylece daha akıllıdır.
b. Programının çalışma süresi daha uzundur ve daha fazla bellek kaplar.

2. Interpreter Nedir? (Yorumlayıcı)

Bir programlama dilini anlaşılır bir dile çevirmek için çalışan bir programdır. Üst düzey program ifadelerini makine kodlarına dönüştürmek için kullanılan bir bilgisayar programıdır. Önceden derlenmiş kod, kaynak kodu ve betikleri içerir.

a. Interpreter(Yorumlayıcı), programın bir zamanında yalnızca bir ifadeyi çevirir.
b. Program çalışmadan önce programlama dilinin bir exe'sini oluştururlar.

3. Compiler ve Interpreter arasındaki fark nedir?

Compiler	Interpreter
• Derleyici tüm dil deyimlerini analiz eder ve yanlış bir şey bulduğunda bir hata atar. • Sıfır hata varsa, derleyici kaynak kodunu birinci makineye dönüştürür. • Çeşitli kod dosyalarını çalıştırılabilir bir programa (exe) bağlar. • Programı çalıştırır.	• Dosyaların birbirine bağlanmasını veya makine kodunun oluşturulmasını gerektirmez. • Yürütme sırasında kaynak deyimlerini satır satır yürütür.
Program kodu zaten makine koduna çevrildiği için kod yürütme süresi nispeten daha kısadır.	Yeni başlayanlar için bile kullanımı ve yürütülmesi oldukça kolaydır.
Kaynak koduna dönmeden bir program değiştirilemez.	Yalnızca ilgili Yorumlayıcıya sahip bilgisayarlar, yorumlanan programları çalıştırabilir.
Makine dilini makine kodu biçiminde diskte saklar.	Makine dilini hiç kaydetmiyor.
Derlenen kodlar nispeten daha hızlı çalışır.	Yorumlanan kodlar nispeten daha yavaş çalışır.
Hedef programlar bağımsız olarak yürütülür. Bellekte Derleyici gerektirmezler.	Yorumlayıcı aslen yorumlama anında hafızada bulunur.
Exe formatında bir çıktı programı oluşturur. Bir kullanıcı, orijinal olarak amaçlanan programdan bağımsız olarak çalıştırabilir.	Bir çıktı programı oluşturmaz. Anlamı, bireysel yürütme sırasında her seferinde kaynak programı değerlendirir.
Program yürütmesi derlemeden ayrılabilir. Böylece, yalnızca tüm çıktının derlemesini tamamladıktan sonra gerçekleştirebilirsiniz.	Programın yürütülmesi, Yorumlama sürecinin adımlarından biridir. Yani, satır satır gerçekleştirebilirsiniz.
Java, Scala, C#, C, C++ Derleyicileri kullanır.	Perl, Ruby, PHP Yorumlayıcıları kullanır.

4. **Neden JAVA Öğreniyorum?**
    a. **Açık Kaynak Kodludur**
    Java açık kaynak kodludur. Yani geliştriceğimiz uygulamalar için ayrıca bir ücret ödememize ya da lisans satın almamıza gerek yoktur.
    b. **Nesne Yönelimlidir**
    Java object oriented yani nesne yönelimli bir dildir. Design patternler ve düzenli kod yazmak için oldukça elverişlidir.
    c. **Yüksek Seviyeli Bir Dildir**
    Yüksek seviyeli diller günlük konuşma diline en yakın olan dillerdir. Bunun için öğrenmesi orta ve düşük seviyeli dillere nazaran daha kolaydır.

Java hemen hemen her projede kullanılabilir. Bunlardan bazıları şunlardır.

1. Mobil Uygulama
2. Web Uygulamaları
3. Masaüstü Uygulamaları

Neden Java Öğrenmeliyim?

Java dili tasarımcıları başlangıçta küçük cihazlar ve avuç içi kullanıma yönelik üretilen cihazlar için geliştirilmiş olmakla birlikte, günümüzde tüm dünyada 9 milyon Java geliştiricisine sahip dünya standardı bir dil haline gelmiş bulunmaktadır. Java'yı bu denli popüler kılan dil özellikleri ise;

- Basit oluşu,
- Dağıtık olması,
- Nesne yönelimli oluşu,
- Çoklu iş yeteneğine sahip olması,
- Dinamik olması,
- Mimari yapıdan bağımsız oluşu,
- Taşınabilirlik özelliği,
- Sağlam ve güvenilir oluşu,
- Yüksek performansa sahip olmasıdır.

Basitlik Özelliği: Java tasarımcıları ve geliştiricileri uygulamanın kolay yazılabilmesi, kolayca derlenebilmesi ve kolayca düzeltilmesine yönelik çalışmalar yapmıştır. Java'nın C++ diline oranla daha basit olmasının nedeni, otomatik bellek tahsisi özelliği ve işi biten nesneleri yok ediyor oluşudur.

Dağıtık Olma Özelliği: Birden fazla bilgisayarın bir tek ağ üzerinde bütünleşik bir sistem olarak bir arada çalışmasına olanak sağlaması sebebiyle dağıtık olarak nitelendirilir.

Nesne Yönelimli Olma: Kullanıcısına birçok önemli yetenek sunarak katılım, polimorfizm (çok biçimlilik), hata ayıklama, modular programlama ve kodların yeniden kullanılabilmesi gibi nesnel yönelimli programlama değerlerinin bütün avantajlarını taşımaktadır.

Çoklu İş Yapma Yeteneği: Bu özellik, bilgisayarın aynı anda birden fazla işi yapabilmesi olarak tanımlanabilir. Java'yı popüler kılan özelliklerden biri de, platform bağımsız bir yapıya sahip oluşudur.

5. Java’da JIT (Java-In-Time Compiler) Nedir?

JIT, uygulamanın performansını optimize eden JVM'nin bir parçasıdır. JIT, Java-In-Time Derleyici anlamına gelir. JIT derlemesi, dinamik derleme olarak da bilinir.

JIT veya Just-In-Time derleyicisi, çalışma süresi boyunca java tabanlı uygulamaların performans optimizasyonundan sorumlu olan JRE'nin (Java Runtime Environment) önemli bir parçasıdır. Derleyici, her iki taraf için, yani son kullanıcı ve uygulama geliştiricisi için bir uygulamanın performansına karar vermede kilit unsurlardan biridir.

Bayt kodu, Java'nın platformlar arası yürütmeye yardımcı olan en önemli özelliklerinden biridir. Yürütme için bayt kodunu yerel makine diline dönüştürmenin yolu, hızı üzerinde büyük bir etkiye sahiptir. Bu bayt kodları, komut seti mimarisine bağlı olarak uygun makine talimatlarına göre yorumlanmalı veya derlenmelidir. Ayrıca, talimat mimarisi bytecode tabanlı ise bunlar doğrudan çalıştırılabilir. Bayt kodunun yorumlanması yürütme hızını etkiler. Performansı artırmak için, JIT derleyicileri çalışma zamanında Java Sanal Makinesi (JVM) ile etkileşime girer ve uygun bayt kodu dizilerini yerel makine koduna derler. Bir JIT derleyicisi kullanırken, donanım, JVM'nin aynı bayt kodu dizisini tekrar tekrar yorumlaması ve çeviri işlemi için ek yüke maruz kalmasıyla karşılaştırıldığında yerel kodu yürütebilir. Derlenen yöntemler daha seyrek yürütülmedikçe, bu daha sonra yürütme hızında performans kazanımlarına yol açar.

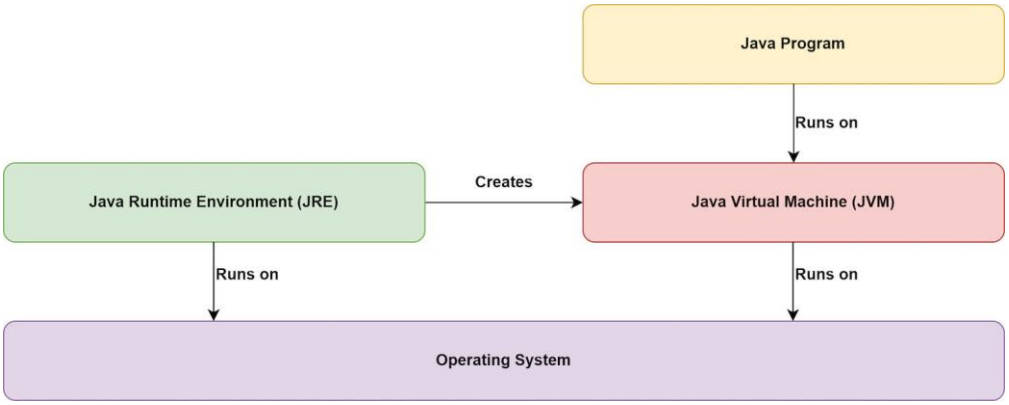
JIT derleyicisi, bir dizi bayt kodunu yerel makine diline derlerken belirli basit optimizasyonları gerçekleştirebilir. JIT derleyicileri tarafından gerçekleştirilen bu optimizasyonlardan bazıları şunlardır: veri analizi, kayıt tahsisi yoluyla bellek erişimlerinin azaltılması, yığın işlemlerinden kayıt işlemlerine çeviri, ortak alt ifadelerin ortadan kaldırılması vb. derleyici yürütme aşamasında harcar. Bu nedenle, yürütme süresine eklenen fazladan ek yük nedeniyle statik bir derleyicinin yapabileceği tüm optimizasyonları yapmayı göze alamaz ve ayrıca programa bakışı da kısıtlanır.

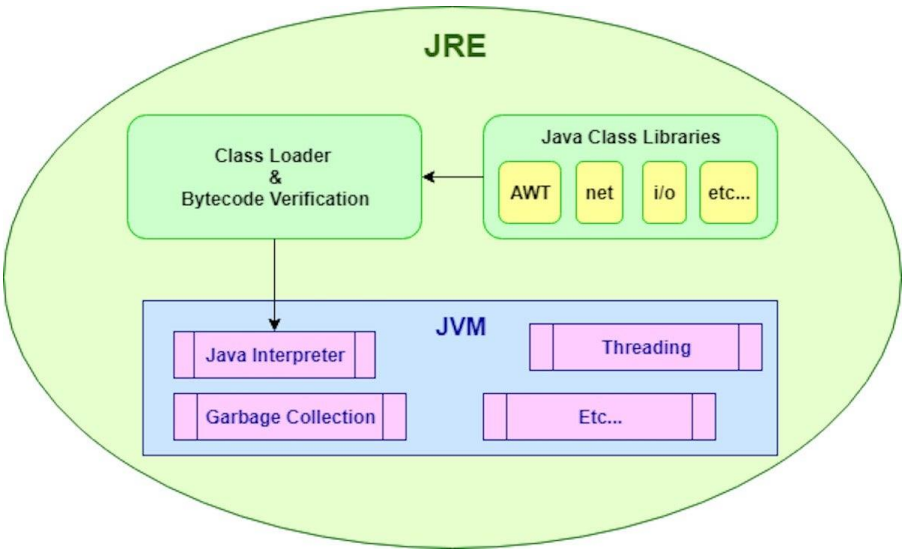
6. JRE (Java Runtime Environment) Nedir?

Java Run-time Environment (JRE), Java Development Kit'in (JDK) bir parçasıdır. Java Sınıf Kitaplığı, özel araçlar ve bağımsız bir JVM içeren, ücretsiz olarak kullanılabilen bir yazılım dağıtımıdır. Java programlarını çalıştırmak için cihazlarda bulunan en yaygın ortamdır. Kaynak Java kodu derlenir ve Java bayt koduna dönüştürülür. Bu bayt kodunu herhangi bir platformda çalıştırmak isterseniz, JRE'ye ihtiyacınız vardır. JRE sınıfları yükler, belleğe erişimi doğrular ve sistem kaynaklarını alır. JRE, işletim sisteminin üstünde bir katman görevi görür.

JRE, Java programlarını geliştirmek ve çalıştırmak için birbiriyle ilişkili üç bileşenden biridir. Diğer iki bileşen aşağıdaki gibidir:

- Java Geliştirme Kiti veya JDK, Java uygulamaları geliştirmek için bir dizi araçtır. Geliştiriciler, JDK'ları Java sürümüne ve pakete veya sürüme göre seçer—Java Enterprise Edition (Java EE), Java Special Edition (Java SE) veya Java Mobile Edition (Java ME). Her JDK, her zaman uyumlu bir JRE içerir, çünkü bir Java programı çalıştırmak, Java programı geliştirme sürecinin bir parçasıdır.
- Java Sanal Makinesi veya JVM, canlı Java uygulamalarını yürütür. Her JRE, varsayılan bir JRE içerir, ancak geliştiriciler, uygulamalarının özel kaynak ihtiyaçlarını karşılayan bir başkasını seçmekte özgürdür.



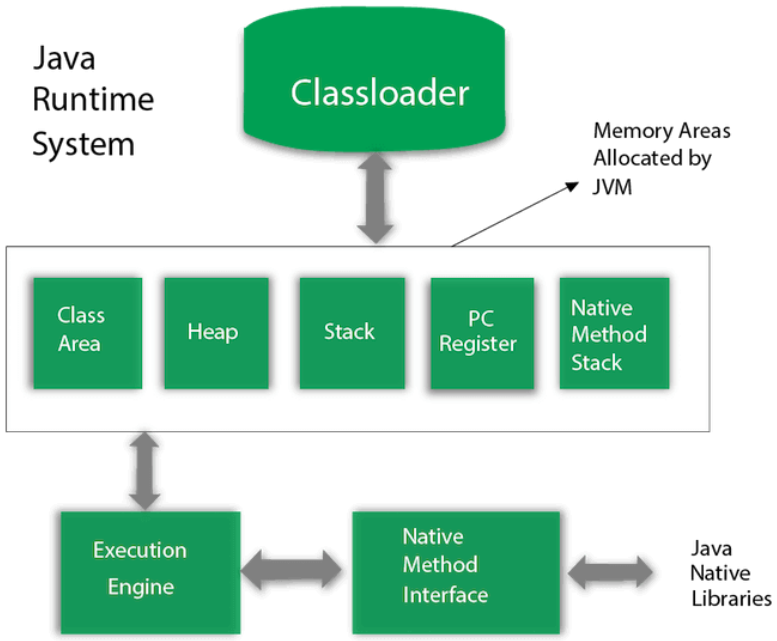


7. JVM (Java Virtual Machine) Nedir?

JVM (Java Virtual Machine) soyut bir makinedir. Java bayt kodunun yürütülebileceği çalışma zamanı ortamını sağlayan bir özelliktir.
JVM'ler birçok donanım ve yazılım platformu için mevcuttur (yani, JVM platforma bağlıdır).

- Java Virtual Machine'in çalışmasının belirtildiği bir belirtim. Ancak uygulama sağlayıcı, algoritmayı seçme konusunda bağımsızdır. Uygulaması Oracle ve diğer şirketler tarafından sağlanmıştır.
- Bir uygulama Uygulaması, JRE (Java Runtime Environment) olarak bilinir.
- Çalışma Zamanı Örneği Java sınıfını çalıştırmak için komut istemine java komutunu her yazdığınızda, bir JVM örneği oluşturulur.

JVM Mimarisi



JVM aşağıdaki işlemi gerçekleştirir:

- Kod yükler
- Kodu doğrular
- Kodu yürütür
- Çalışma zamanı ortamı sağlar

JVM aşağıdakiler için tanımlar sağlar:

- Hafıza alanı
- Sınıf dosyası formatı
- Kayıt seti
- Çöp toplanmış yığın
- Önemli hata bildirimi vb.

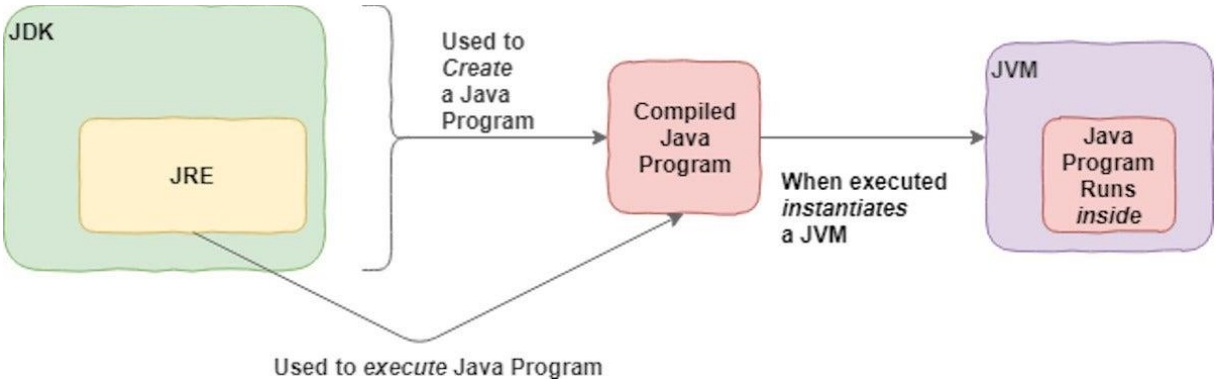
8. JDK: Java Development Kit Nedir?

JDK, Java Development Kit'in kısaltmasıdır. Java Geliştirme Kiti (JDK), java uygulamaları ve uygulamaları geliştirmek için kullanılan bir yazılım geliştirme ortamıdır. Fiziksel olarak var. JRE + geliştirme araçlarını içerir.

JDK, Oracle şirketi tarafından yayınlanan aşağıda verilen Java Platformlarından herhangi birinin bir uygulamasıdır:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

JDK, geliştirmeyi tamamlamak için özel bir Java Sanal Makinesi (JVM) ve bir yorumlayıcı/yükleyici (Java), bir derleyici (javac), bir arşivleyici (jar), bir dokümantasyon oluşturucu (Javadoc) vb. gibi birkaç başka kaynak içerir. bir Java Uygulaması.



9. Debug Nedir? (Hata Ayıklama)

Herhangi bir yazılımın geliştirme sürecinde, hatasız ürünler sunmak adına yazılım programı test edilir, sorun giderilir ve bakımı yapılır. İlk seferde hatasız olan hiçbir şey yoktur.
Bu nedenle, herkesin ilişkilendireceği açık bir şey, yazılım oluşturulduğunda olduğu gibi, birçok hata içerir; kimsenin mükemmel olmaması ve kodda hata almanın nedeni sorun değil, ondan kaçınmak ya da engellememek sorun!
Tüm bu hatalar ve hatalar düzenli olarak atılır, bu nedenle hata ayıklamanın bir yazılım programında bulunan hataları ortadan kaldırma veya düzeltme sürecinden başka bir şey olmadığı sonucuna varabiliriz.

Hata ayıklama, hataları belirleme, analiz etme ve ardından hataları giderme ile başlayarak adım adım çalışır. Bir yazılım sonucu veremediğinde, uygulamayı test etmek ve çözmek için yazılım test uzmanına ihtiyacımız var. Yazılım testinde hata ayıklamanın her adımında hatalar çözüldüğünden, sonuç ne kadar verimli olursa olsun bunun yorucu ve karmaşık bir görev olduğu sonucuna varabiliriz.

Java'da Debugging;
https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html

10. Senkron Programlama Nedir (Sync)?

Senkron programlada her şey belli bir sırayla işlenir ve işlem sırasının bitmesi beklenir. Bu oldukça zaman kaybı olduğu gibi programımızıda yavaşlatır ve hatta işlem bitene kadar durdurabilir.
Senkron programlamadaki her şeyi sırayla işlemesi ve her bir işlemin birbirini beklemesi yeri geldiğinde programımızı çok yavaşlatabilir, hatta işlem bitene kadar durdurabilir.

Örneğin yukarıdaki kodda 5. satır bir önceki satırı yani dosya okuma işlemini beklemek zorundadır. Dosya içeriği çok büyükse bu işlemler dakikalar bile alabilir. Ekrana "Program çalışıyor…" yazdırmak için bir önceki işlemin bitmesini beklemek pek akıllıca değildir. İşte bu tip durumlar için asenkron fonksiyonlar kullanırız. Kod akışının sırayla işlemediği, işlemlerin birbirini beklemediği, kod akışının işlem durumlarına göre devam ettiği programlamaya "Asenkron Programlama" denir.

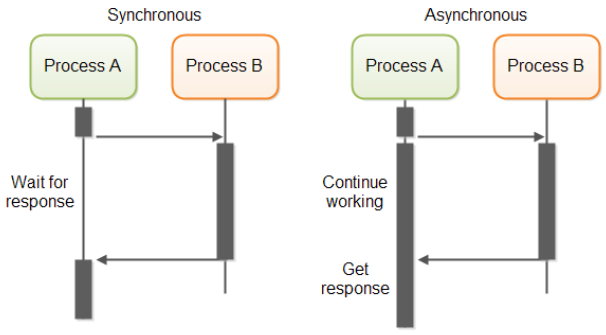
11. Asenkron Programlama Nedir (Async)?

Asenkron(Async) programlamaya neden ihtiyaç duyulmuş ilk olarak bu soruyu cevaplayalım. Kullanmakta olduğumuz programlarda aynı anda birden çok işlem yapılabilmektedir. Örneğin siz kullanıcı arayüzünde bir yazı görüyorken arka planda bir web servisine istek gönderilmiş ve cevabı bekleniyor olabilir. İlk nesil programlarda web servisinden cevap gelene kadar siz kullanıcı arayüzünde herhangi bir tuşa basamazdınız. Eğer basarsanız bir şey olmayacaktır ve peş peşe bir kaç defa basarsanız **program durduruldu** hatası alırdık.

Asenkron fonksiyon tanımlamak özellikler farklı tipte işlemleri bir arada yürütüyorken kullanmak oldukça pratik ve sağlıklıdır. Ancak her fonksiyonu Asenkron olarak yazılmaz. Dataların kontrolünü kaybedebilirsiniz.
Kısacası asenkron programlama programın senkron bir şekilde değil de öncelik verdiğimiz işlemlerin daha önce yapılmasını sağlayan ya da sağladığımız programlama türüdür.

12. Senkron(Sync) ve Asenkron(Async) Programlama Arasındaki Fark Nedir?

- Senkron programlamada kodlar yukarıdan aşağıya doğru hiyerarşik bir biçimde çalışır.
- Asenkron programlamada ise öncelikli olarak hangi fonksiyon çalıştırılmak isteniyorsa o çalışır.



13. Java Socket Programlama

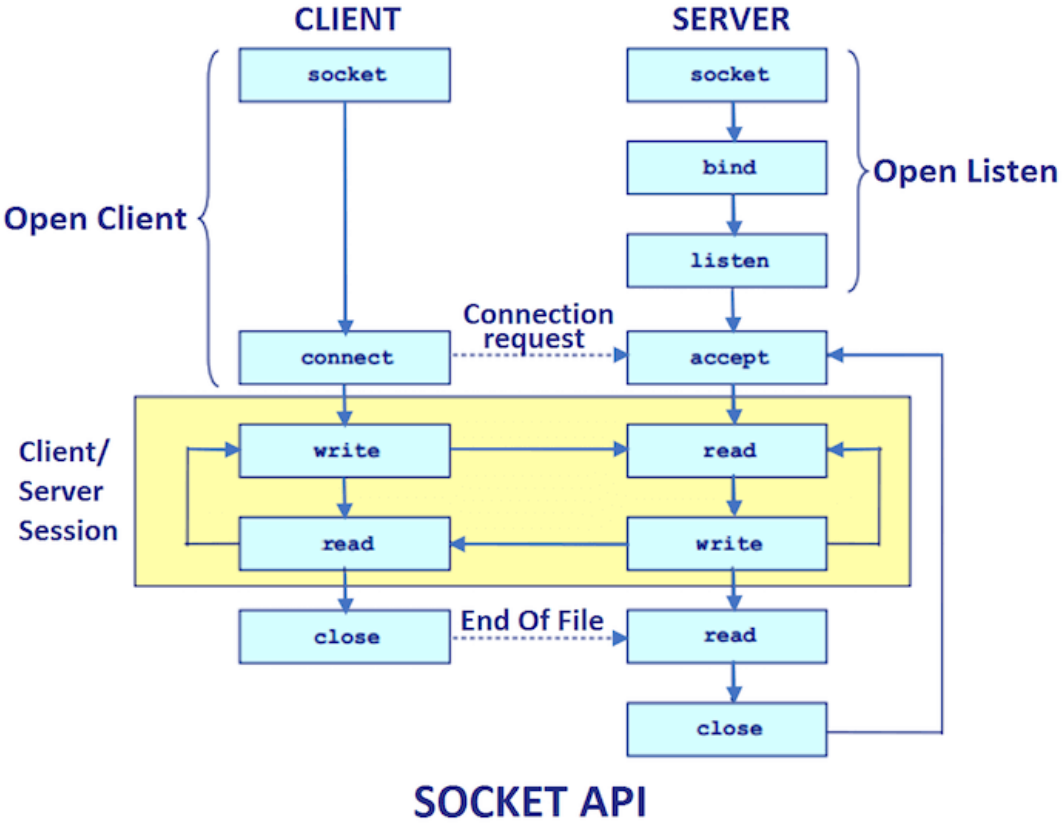
Java Socket programlama, farklı JRE üzerinde çalışan uygulamalar arasındaki iletişim için kullanılır.
Java Socket programlama, bağlantı odaklı veya bağlantısız olabilir.

Bağlantı yönelimli soket programlama için Socket ve ServerSocket sınıfları, bağlantısız soket programlama için DatagramSocket ve DatagramPacket sınıfları kullanılır.

Soket programlamadaki istemci iki bilgiyi bilmelidir:

1. Sunucunun IP Adresi ve
2. Port Numarası

Burada tek yönlü client ve server iletişimi yapacağız. Bu uygulamada istemci sunucuya bir mesaj gönderir, sunucu mesajı okur ve yazdırır. Burada iki sınıf kullanılıyor: Socket ve ServerSocket. Socket sınıfı, istemci ve sunucu arasında iletişim kurmak için kullanılır. Bu sınıf sayesinde mesaj okuyabilir ve yazabiliriz. ServerSocket sınıfı, sunucu tarafında kullanılır. ServerSocket sınıfının accept() yöntemi, istemci bağlanana kadar konsolu engeller. İstemcinin başarılı bağlantısından sonra, sunucu tarafında Socket örneğini döndürür.

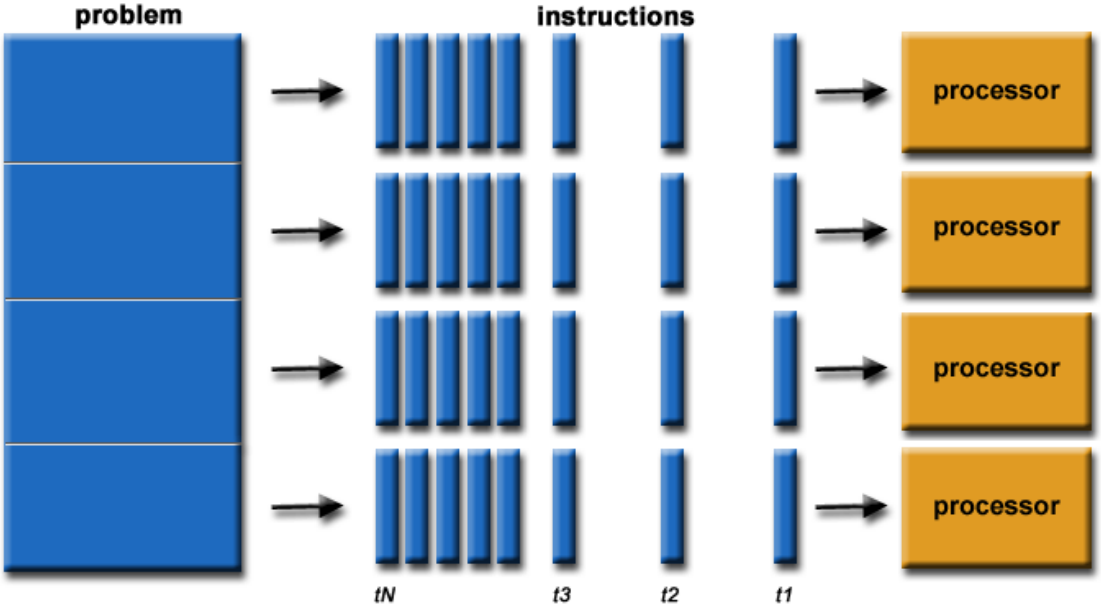


Kaynakça:
https://www.javatpoint.com/socket-programming

14. Java’da Paralel Programlama

Paralel Programlama, karmaşık bir sorunu, birkaç bilgisayar kaynağı kullanılarak aynı anda yürütülebilen daha küçük ve daha basit görevlere ayırma işlemidir. Paralel Programlama sürecinde, birbirinden bağımsız görevler, farklı bilgisayarlar veya bir bilgisayarın CPU'sunda bulunan birden fazla bilgisayar kullanılarak paralel olarak yürütülür. Paralel Programlama, firmaların ekonomik standartlarını korumaları gerektiğinden, ağır ve büyük ölçekli projeleri ele almaları için vazgeçilmez bir gerekliliktir. Paralel Programlama sayesinde projenin hızı artar, hata olasılığı azalır.

Paralel Programlama, görevlerinin bir yürütme sırasını takip etmesi gerekmediği için çoklu kullanımdan oldukça farklıdır. Paralel Programlamanın her görevi, gerçekleştirmesi gereken işleve göre tasarlanır. Bu nedenle, yaygın olarak İşlevsel Paralellik veya Veri Paralelliği olarak bilinir.



Paralel programlama, **haberleşebilen ve iş birliği yapabilen**, **birden fazla** işlem biriminin **eş zamanlı** çalışarak tek bir işlem adımında birden fazla işlemi gerçekleştirebilmesidir. Belirli bir zaman zarfında birden fazla işlem yapılır. İşlemler birden çok bilgisayarda gerçekleştirilebildiği gibi, bir bilgisayardaki çok çekirdekli bir işlemcinin üzerinde de gerçekleştirilebilir. Problemin her bir parçası farklı bir bilgisayarda çözümlenir.

Kaynakça:
https://blog.gutfsozluk.com/paralel-programlama-nedir/
https://www.javatpoint.com/parallel-programming-in-java

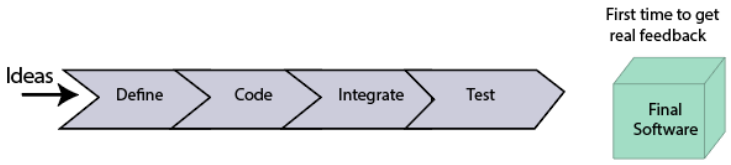
15. Agile Nedir?

Agile (çevik) yöntem, yazılım geliştirmede kullanılan proje yönetimine özel bir yaklaşımdır. Bu yöntem ekiplerin yazılım geliştirme süreçlerinin öngörülemezliğine cevap vermesine yardımcı olur. Genellikle sprint olarak bilinen artımlı, yinelemeli iş dizilerini kullanır.

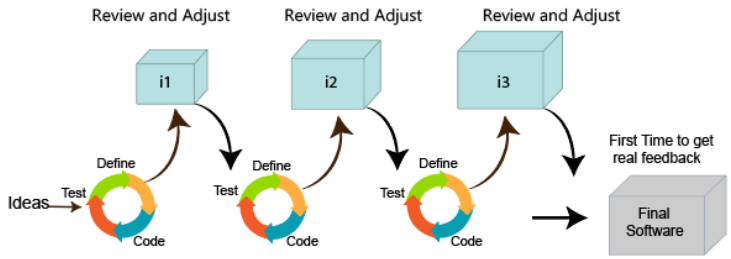
Agile metodunun tanımı:
Sprint, bir projenin belirli bir aşaması için ayrılan süredir. Sprintlerin süresi dolduğunda proje tamamlanmış sayılır. Ekip üyeleri arasında, gelişimin tatmin edici olup olmadığı konusunda anlaşmazlıklar olabilir; ancak, projenin belirli bir aşamasında bu konuda daha fazla çalışma yapılmayacaktır. Projenin kalan aşamaları, kendi zaman dilimleri içinde geliştirilmeye devam edecektir.

Agile metodunun genel prensipleri:

- Müşteriyi memnun etmek ve sürekli yazılım geliştirmek önemlidir.
- Müşterinin rekabet avantajı için değişen gereksinimler benimsenmelidir.
- Sık sık çalışan yazılımlar sunmaya odaklanılmalıdır. Teslimat, mümkün olan en kısa sürede yapılmalıdır.
- Geliştiriciler ve iş adamları tüm proje boyunca birlikte çalışmalıdır.
- Projeler motive olmuş insanlarla devam etmelidir. Onlara uygun ortam ve ihtiyaç duydukları destek sağlanmalıdır. İşlerini yapmak için güvende olmalıdırlar.
- Yüz yüze iletişim, bir takıma bilgi aktarmanın en iyi yoludur.
- Çalışan yazılım, ilerlemenin birincil ölçümüdür.
- Çevik süreçler sürdürülebilir kalkınmayı teşvik ederler. Sponsorlar, geliştiriciler ve kullanıcılar belirsiz ve sürekli bir tempoyu koruyabilmelidir.
- Teknik mükemmellik ve iyi tasarıma sürekli dikkat etmek çevikliği artıracaktır.
- Sadelik, yapılmayan işi en üst düzeye çıkarma sanatı olarak kabul edilir ve esastır.
- Kendi kendine organize ekipler genellikle en iyi tasarımları oluşturur.
- Düzenli aralıklarla, takımın nasıl daha etkili olacağına dair düşünülmeli ve davranışlar buna göre düzenlemelidir.



Traditional Method



Agile Method

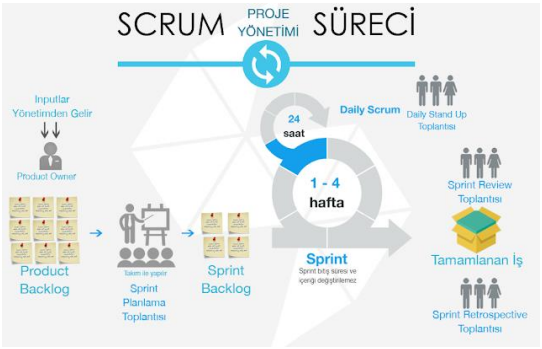
16. Scrum Nedir?

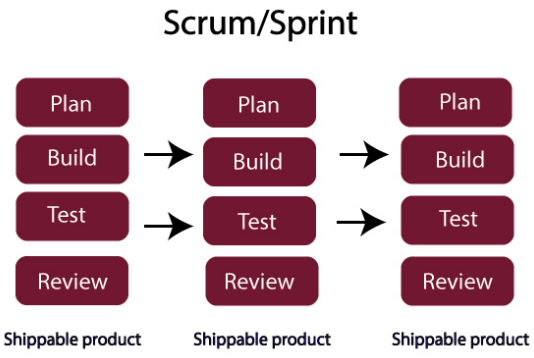
Scrum, Agile(çevik) ekiplerin birlikte çalışmasına yardımcı olan bir framework. Bunu kullanarak, ekip üyeleri karmaşık ürünü teslim edebilir ve sürdürebilir. Takımı uygulama yoluyla öğrenmeye, problem üzerinde çalışırken kendi kendini organize etmeye teşvik eder. Pislik, çerçeve aracılığıyla yapılan ve müşterilere sürekli olarak değer gönderen bir çalışmadır.

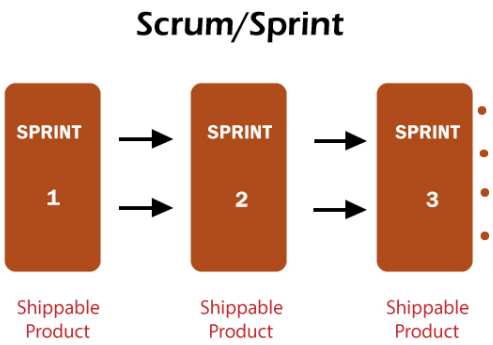
Geliştirme ekibi tarafından en sık kullanılan yazılımdır. İlkesi ve dersleri her türlü ekip çalışmasına uygulanabilir. Politikası ve deneyimleri, Scrum çerçevesinin popülaritesinin bir nedenidir. Scrum, ekiplerin yapılanmasına yardımcı olan bir dizi aracı, toplantıyı ve rolü tanımlar. Aynı zamanda ekibin yaptığı işleri de yönetir.

Scrum Metodu nedir?
Scrum; Agile proje yönetim metotlarından biridir. Scrum Metodu yazılım süreçlerinin yönetimi için kullanılmaktadır. Ayrıca düzenli geri bildirim ve planlamalar ile hedefe ulaşmayı sağlar. Bu açıdan ihtiyaca yönelik ve esnek bir yapıya sahiptir. Müşteri ihtiyacına yönelik olduğu için geri bildirimlere göre yapılanmayı sağlar. Scrum Metodu sürecinde iletişim ve takım çalışması oldukça önemlidir. Bu metot 3 temel prensip üzerine kurulmuştur. Bunlar aşağıdaki gibidir:

Şeffaflık: Projenin ilerlemesi ve problemler, gelişmeler herkes tarafından görülebilir olmalıdır.
Gözlemleme: Projenin ilerlemesi düzenli olarak kontrol edilmelidir.
Uyarlama: Proje, uygulanacak değişiklikler ile uyumlu olmalıdır.







https://btm.istanbul/blog/scrum-metodu-nedir-nasil-uygulanir
https://www.javatpoint.com/agile-scrum

17. Swagger ve API Document

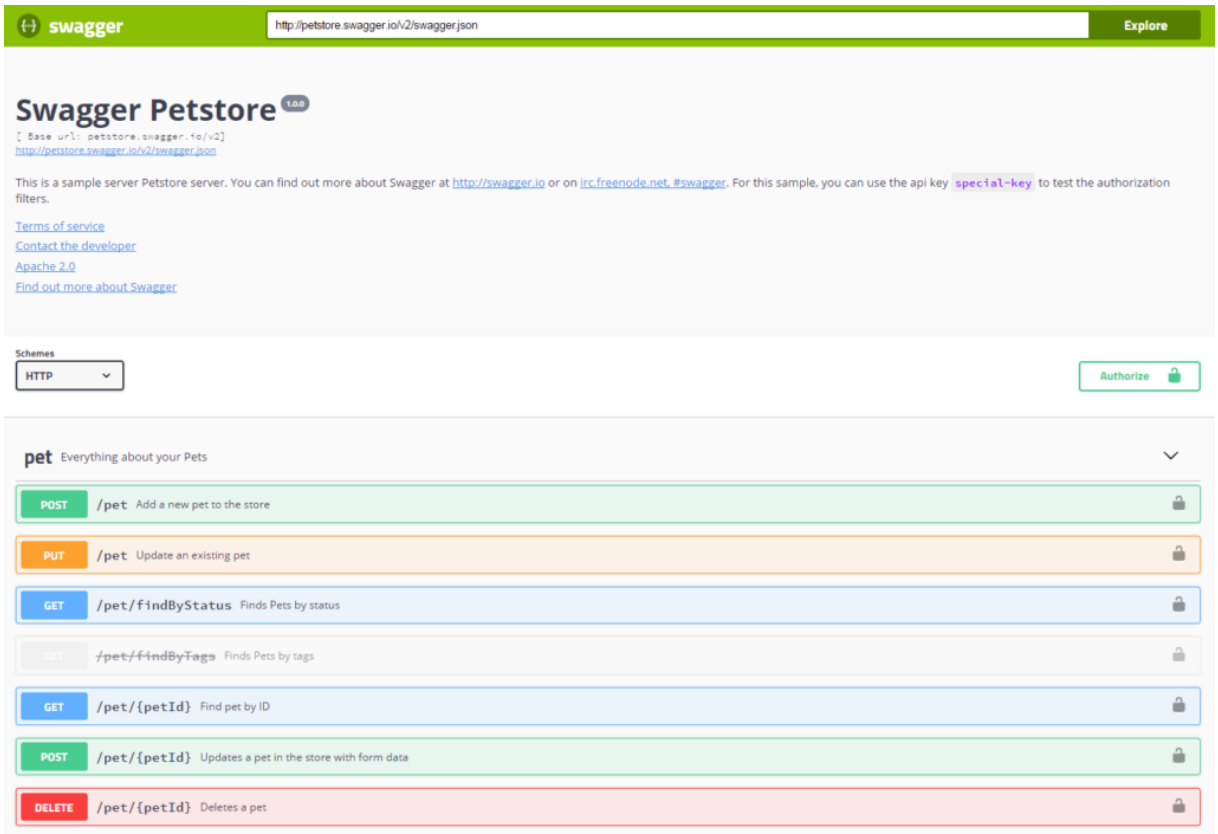
Swagger

Swagger, API'lerinizin yapısını makinelerin okuyabilmesi için tanımlamanıza olanak tanır. API'lerin kendi yapılarını tanımlama yeteneği, Swagger'daki tüm mükemmelliğin köküdür. Neden bu kadar harika? API'nizin yapısını okuyarak, otomatik olarak güzel ve etkileşimli API belgeleri oluşturabiliriz. API'niz için birçok dilde otomatik olarak istemci kitaplıkları oluşturabilir ve otomatik test etme gibi diğer olasılıkları keşfedebiliriz. Swagger bunu, API'nizden tüm API'nizin ayrıntılı bir açıklamasını içeren bir YAML veya JSON döndürmesini isteyerek yapar. Bu dosya, esasen API'nizin OpenAPI Spesifikasyonuna uyan bir kaynak listesidir.

API Documentation

API dokümantasyonu, bir API'nin etkin bir şekilde nasıl kullanılacağına ve API ile entegre edileceğine ilişkin talimatları içeren, teslim edilebilir bir teknik içeriktir. Öğreticiler ve örneklerle desteklenen, işlevler, sınıflar, dönüş türleri, bağımsız değişkenler ve daha fazlasıyla ilgili ayrıntılarla birlikte API ile çalışmak için gereken tüm bilgileri içeren özlü bir başvuru kılavuzudur. API Dokümantasyonu, geleneksel olarak düzenli içerik oluşturma ve bakım araçları ve metin editörleri kullanılarak yapılır.

OpenAPI/Swagger Spesifikasyonu gibi API açıklama formatları, dokümantasyon sürecini otomatikleştirerek ekiplerin bunları oluşturmasını ve sürdürmesini kolaylaştırdı.



https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/

18. ModalMapper

ModelMapper, nesneleri bir modelden diğerine eşlemeye yardımcı olan ve manuel eşleme kodu ihtiyacını azaltan bir Java kütüphanesidir. Nesneleri otomatik olarak eşlemek için kuralları kullanan, ancak özel eşleme kuralları sağlayacak şekilde de yapılandırılabilen, yüksek düzeyde yapılandırılabilir bir kütüphanedir.

https://zubairehman.medium.com/how-to-use-modelmapper-in-spring-boot-8a494e316840#:~:text=ModelMapper%20is%20a%20Java%20library,to%20provide%20custom%20mapping%20rules.
https://modelmapper.org/
https://modelmapper.org/getting-started/
https://www.baeldung.com/java-modelmapper

19. BCrypt

BCrypt, Niels Provos ve David Mazieres tarafından "Geleceğe Uyarlanabilir Bir Parola Şeması" bölümünde açıklanan şemayı kullanarak OpenBSD tarzı Blowfish parola karma işlemini uygular.
Bu parola karma sistemi, Bruce Schneier'in Blowfish şifresine dayalı, hesaplama açısından yoğun bir karma algoritması kullanarak çevrimdışı parola kırılmasını engellemeye çalışır. Algoritmanın iş faktörü parametrelendirilir, böylece bilgisayarlar hızlandıkça artırılabilir.

Kullanımı:

```
String pw_hash = BCrypt.hashpw(plain_password, BCrypt.gensalt());
```

Bir düz metin parolasının daha önce karma oluşturulmuş bir parolayla eşleşip eşleşmediğini kontrol etmek için checkpw yöntemini kullanın:

```
if (BCrypt.checkpw(candidate_password, stored_hash))
    System.out.println("It matches");
else
    System.out.println("It does not match");
```

BCrypt, Blowfish şifresine dayalı tek yönlü bir salted hash işlevidir. Düz metin parolalar (maalesef bu hala oldukça sık oluyor) ve geleneksel karma algoritmalar (md5) üzerinde çeşitli geliştirmeler sağlar. BCrypt'in şifreleri saklamanın en iyi yolu olduğunu söylemek doğru olmaz ama yeterince iyi olmalı. PBKDF2 gibi algoritmalar, daha kapsamlı bir şekilde test edilmiş bir algoritma olarak kullanılabilir, ancak BCrypt de yaygın olarak kullanılır. jBCrypt, BCrypt'in bir Java uygulamasıdır.

Kaynakça:
https://dzone.com/articles/hashing-passwords-in-java-with-bcrypt
https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html

20. SPRING FRAMEWORK

Spring Framework, her türlü dağıtım platformunda modern Java tabanlı kurumsal uygulamalar için kapsamlı bir programlama ve yapılandırma modeli sağlar.

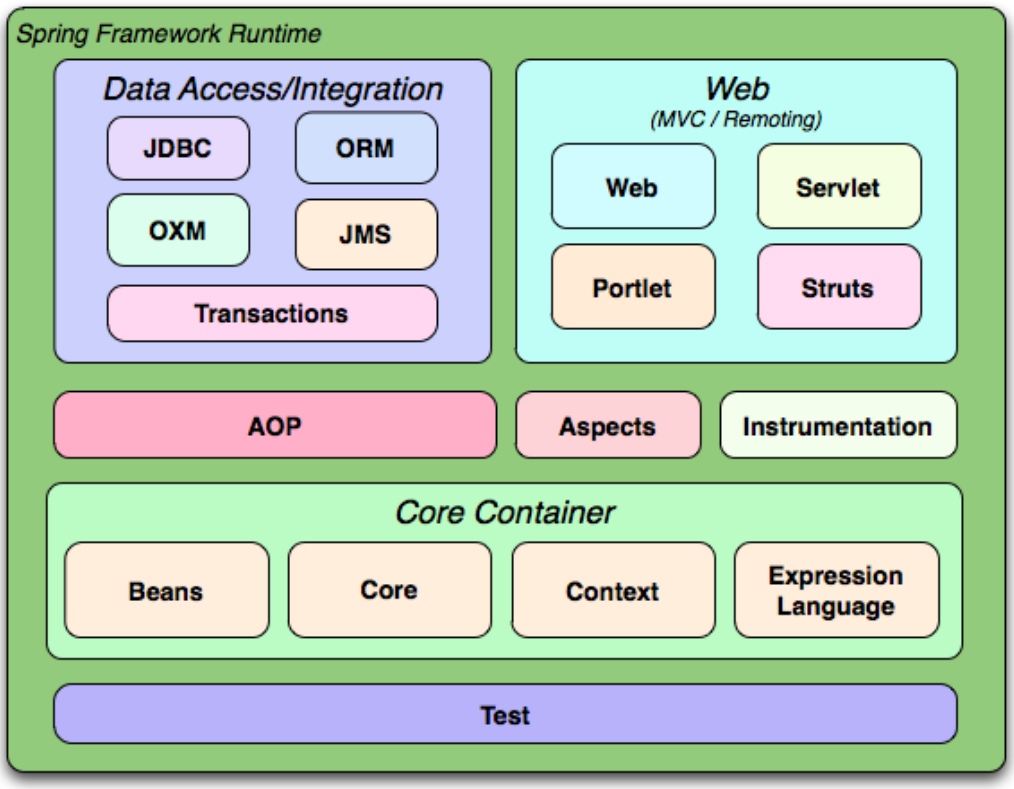
Spring'in temel unsurlarından biri, uygulama düzeyinde altyapısal destektir: Spring, kurumsal uygulamaların "tesisatına" odaklanır, böylece ekipler, belirli dağıtım ortamlarıyla gereksiz bağlar olmadan uygulama düzeyinde iş mantığına odaklanabilir.

Spring Framework, Java uygulamaları geliştirmek için kapsamlı altyapı desteği sağlayan bir Java platformudur. Spring, uygulamanıza odaklanabilmeniz için altyapıyı yönetir.

Spring, "düz eski Java nesnelerinden" (POJO'lar) uygulamalar oluşturmanıza ve kurumsal hizmetleri müdahaleci olmayan bir şekilde POJO'lara uygulamanıza olanak tanır. Bu yetenek, Java SE programlama modeli ve tam ve kısmi Java EE için geçerlidir.

Spring platformu avantajını nasıl kullanabileceğinize dair örnekler:

- İşlem API'leriyle uğraşmak zorunda kalmadan bir veritabanı işleminde bir Java yönteminin yürütülmesini sağlayın.
- Uzak API'lerle uğraşmak zorunda kalmadan yerel bir Java yöntemini uzak bir prosedür haline getirin.
- JMX API'leriyle uğraşmak zorunda kalmadan yerel bir Java yöntemini bir yönetim işlemi haline getirin.
- JMS API'leriyle uğraşmak zorunda kalmadan yerel bir Java yöntemini bir mesaj işleyici yapın.



21. SPRING BOOT

Spring Boot, "sadece çalıştırabileceğiniz", bağımsız, üretim sınıfı Spring tabanlı Uygulamalar oluşturmayı kolaylaştırır.
Spring platformuna ve üçüncü taraf kitaplıklarına ilişkin kararlı bir görüş alıyoruz, böylece en az zahmetle başlayabilirsiniz. Spring Boot uygulamalarının çoğu minimum Spring konfigürasyonuna ihtiyaç duyar.

Java Spring Framework (Spring Framework), Java Virtual Machine (JVM) üzerinde çalışan bağımsız, üretim düzeyinde uygulamalar oluşturmak için popüler, açık kaynaklı, kurumsal düzeyde bir çerçevedir.

Java Spring Boot (Spring Boot), üç temel yetenek aracılığıyla Spring Framework ile web uygulaması ve mikro hizmetler geliştirmeyi daha hızlı ve kolay hale getiren bir araçtır:

- Autoconfiguration (otomatik yapılandırma)
- Konfigürasyona düşünceli bir yaklaşım
- Konfigürasyona düşünceli bir yaklaşım

Bu özellikler, minimum yapılandırma ve kurulumla Spring tabanlı bir uygulama kurmanıza izin veren bir araç sağlamak için birlikte çalışır.

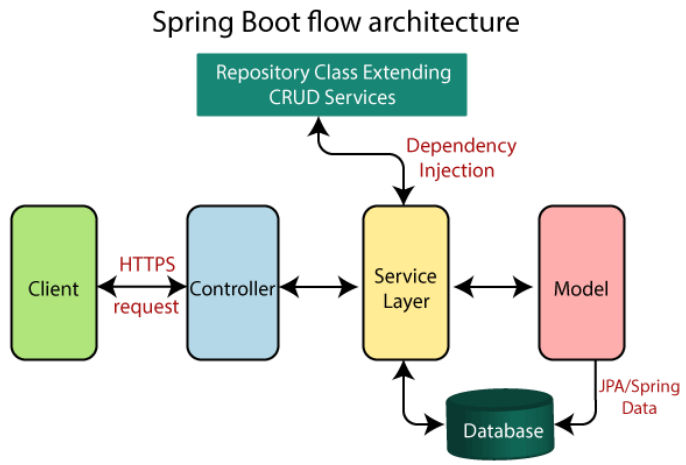
Avantajlar

- Anlaşılması ve geliştirilmesi kolay yay uygulamaları
- Verimliliği artırır
- Geliştirme süresini azaltır

Hedefler / Amaçlar

- Spring'deki karmaşık XML yapılandırmasından kaçınmak için
- Üretime hazır Spring uygulamalarını daha kolay bir şekilde geliştirmek
- Geliştirme süresini azaltmak ve uygulamayı bağımsız olarak çalıştırmak için
- Uygulamaya başlamanın daha kolay bir yolunu sunmak

Spring Boot Akış Mimarisi



22. Servlet Nedir? (Sunucu Uygulaması)

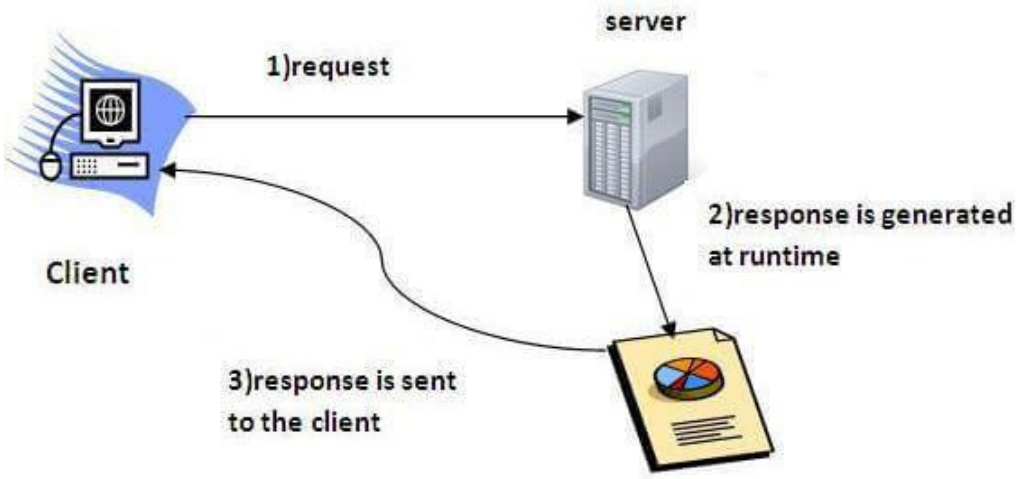
Servlet teknolojisi, bir web uygulaması oluşturmak için kullanılır (sunucu tarafında bulunur ve dinamik bir web sayfası oluşturur).

Servlet, istek-yanıt programlama modeli aracılığıyla erişilen uygulamaları barındıran sunucuların yeteneklerini genişletmek için kullanılan bir Java programlama dili sınıfıdır. Servlet'ler her türlü talebe cevap verebilmelerine rağmen, genellikle web sunucuları tarafından barındırılan uygulamaları genişletmek için kullanılırlar.
Bu tür uygulamalar için Java Servlet teknolojisi, HTTP'ye özgü servlet sınıflarını tanımlar.

javax.servlet ve javax.servlet.http paketleri, servlet yazmak için arayüzler ve sınıflar sağlar. Tüm sunucu uygulamaları, yaşam döngüsü yöntemlerini tanımlayan Servlet arabirimini uygulamalıdır. Genel bir hizmeti uygularken, Java Servlet API ile sağlanan GenericServlet sınıfını kullanabilir veya genişletebilirsiniz. HttpServlet sınıfı, HTTP'ye özgü hizmetleri işlemek için doGet ve doPost gibi yöntemler sağlar.

Servlet, bağlama bağlı olarak birçok şekilde tanımlanabilir:

- Servlet, bir web uygulaması oluşturmak için kullanılan bir teknolojidir.
- Servlet, dokümantasyon da dahil olmak üzere birçok arayüz ve sınıf sağlayan bir API'dir.
- Servlet, herhangi bir Servlet oluşturmak için uygulanması gereken bir arayüzdür.
- Servlet, sunucuların yeteneklerini genişleten ve gelen isteklere cevap veren bir sınıftır. Her türlü talebe cevap verebilir.
- Servlet, dinamik bir web sayfası oluşturmak için sunucuda dağıtılan bir web bileşenidir.



Kaynakça:
https://www.javatpoint.com/servlet-tutorial
https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html
https://www.geeksforgeeks.org/introduction-java-servlets/

23. HTTP (Hypertext Transfer Protocol)

Köprü Metni Aktarım Protokolü (HTTP), World Wide Web'in temelidir ve hiper metin bağlantılarını kullanarak web sayfalarını yüklemek için kullanılır. HTTP, ağa bağlı cihazlar arasında bilgi aktarmak için tasarlanmış bir uygulama katmanı protokolüdür ve ağ protokolü yığınının diğer katmanlarının üzerinde çalışır. HTTP üzerinden tipik bir akış, bir istemci makinenin bir sunucuya istekte bulunmasını ve ardından bir yanıt mesajı göndermesini içerir.

- HTTP (Hypertext Transfer Protocol), Hiper Metin Aktarım Protokolü anlamına gelir.
- İstemci bilgisayarlar ve web sunucuları arasındaki iletişim, HTTP İstekleri göndererek ve HTTP yanıtları alarak yapılır.

World Wide Web İletişimi

WWW, web istemcileri ve sunucular arasındaki iletişimle ilgilidir.

İstemciler genellikle tarayıcılardır (Chrome, Edge, Safari), ancak herhangi bir tür program veya cihaz olabilirler.

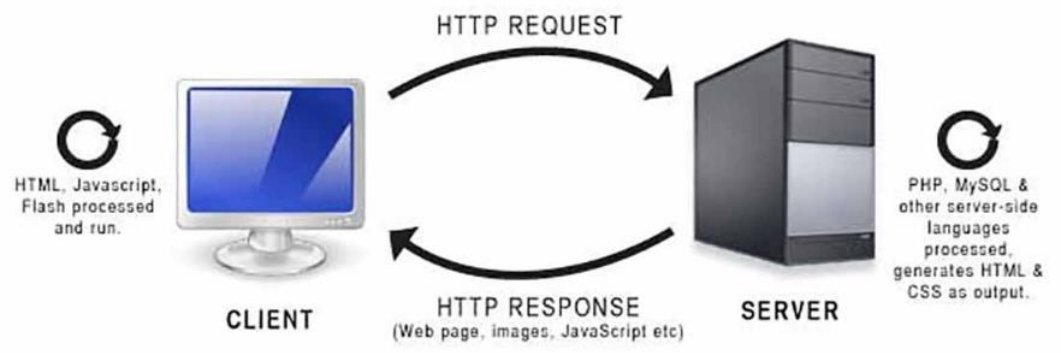
Sunucular genellikle buluttaki bilgisayarlardır.

HTTP Request / Response (İstek / Yanıt)
İstemciler ve sunucular arasındaki iletişim, istekler ve yanıtlarla yapılır:

- Bir client/istemci (bir tarayıcı) web'e bir HTTP isteği gönderir.
- Bir web server (web sunucusu) isteği alır.
- Server (Sunucu), isteği işlemek için bir uygulama çalıştırır.
- Server (Sunucu), tarayıcıya bir HTTP yanıtı (çıktı) döndürür.
- Client/istemci (tarayıcı) yanıtı alır

HTTP Request Circle
Tipik bir HTTP isteği/yanıt döngüsü:

- Tarayıcı bir HTML sayfası ister. Sunucu bir HTML dosyası döndürür.
- Tarayıcı bir stil sayfası ister. Sunucu bir CSS dosyası döndürür.
- Tarayıcı bir JPG görüntüsü ister. Sunucu bir JPG dosyası döndürür.
- Tarayıcı, JavaScript kodunu ister. Sunucu bir JS dosyası döndürür
- Tarayıcı veri ister. Sunucu verileri döndürür (XML veya JSON'da).



24. URL ve URI

URL (Uniform Resource Locator)
URL genellikle bir adrese yönlendirilen bir karakter dizisi olarak tanımlanır. Web üzerindeki kaynakları bulmak için çok yaygın olarak kullanılan bir yoldur. Ağ konumunu veya birincil erişim mekanizmasını tanımlayarak fiziksel konumun sunumunu almak için bir yol sağlar.

Protokol, kaynak ve kaynak adını almak için kullanılan URL'de açıklanmıştır. Kaynak web tipi bir kaynak olabilirse, URL başlangıçta http/https içerir. Benzer şekilde, kaynak bir dosya ise ftp ile, e-posta adresi ise mailto ile başlar. Bir URL'nin sözdizimi, birincil bölümün protokol için kullanıldığı ve bölümün geri kalanının bir web sitesi adı veya program adından oluşan kaynak için kullanıldığı aşağıda gösterilmiştir.

```
https://www.geeksforgeeks.org/minimum-cost-graph
```

Burada alan adı, sunucuyu (web hizmeti) ve program adını (sunucudaki dizine ve dosyaya giden yol) tanımlar.

URI (Uniform Resource Identifier)
URL'ye benzer şekilde, URI da konum, ad veya her ikisini birden kullanarak web'deki bir kaynağı tanımlayan bir karakter dizisidir. Kaynakların tek tip tanımlanmasına izin verir. Bir URI ayrıca bir yer bulucu, bir ad veya her ikisi olarak gruplandırılır, bu da onun bir URL'yi, URN'yi veya her ikisini birden tanımlayabileceğini gösterir. URI içindeki tanımlayıcı terimi, kullanılan tekniğe rağmen kaynakların önemini ifade eder.

URI'deki önceki kategori URL'dir; bu sırada, kaynağa erişim yöntemini belirtmek için bir protokol kullanılır ve kaynak adı ek olarak URL'de düzenlenir. Bir URL, kalıcı olmayan bir URI türü olabilir. Bir URN'nin küresel olarak benzersiz olması gerekir ve küresel bir kapsama sahiptir.

URI	URL
URI, adı söyleyen URN ve konumu bildiren URL olmak üzere iki alt küme içerir.	URL, kaynağın tek konumunu söyleyen URI'nin alt kümesidir.
Adı veya konumu söyleyebildikleri için tüm URI'ler URL olamaz.	Her URL yalnızca konumu içerebileceğinden, tüm URL'ler URI'lerdir.
Bir URI, kaynağın adını veya kaynağın konumunu kullanarak bir kaynağı tanımlamayı ve onu diğer kaynaklardan ayırmayı amaçlar.	Bir URL, web üzerindeki bir kaynağın konumunu veya adresini bulmayı amaçlar.
Bir URI örneği, ISBN 0-486-35557-4 olabilir.	Bir URL örneği https://www.javatpoint.com'dur.
Kaynakları ve ikili dosyaları tanımlamak için JSTL ve XSTL gibi XML ve etiket kitaplığı dosyalarında yaygın olarak kullanılır.	Genelde internetteki web sayfalarında arama yapmak için kullanılır.
URI şeması protokol, atama, belirtim veya herhangi bir şey olabilir.	URL şeması genellikle HTTP, HTTPS, FTP vb. gibi bir protokoldür.

25. Docker

Docker, uygulamalarınızı hızla derlemenize, test etmenize ve dağıtmanıza imkan tanıyan bir yazılım platformudur. Docker, yazılımları kitaplıklar, sistem araçları, kod ve çalışma zamanı dahil olmak üzere yazılımların çalışması için gerekli her şeyi içeren container adlı standartlaştırılmış birimler halinde paketler. Docker'ı kullanarak her ortama hızla uygulama dağıtıp uygulamaları ölçeklendirebilir ve kodunuzun çalışacağından emin olabilirsiniz.

Docker en net tanımlamayla open source bir 'container' teknolojisidir. Docker, aynı işletim sistemi üzerinde, yüzlerce hatta binlerce birbirinden izole ve bağımsız containerlar sayesinde sanallaştırma sağlayan bir teknolojidir. Web uygulamalarımızın kolayca kurulumunu, testini, çalışmasını ve deploymentını sağlar. Bunun yanında sunucu maliyetlerini önemli ölçüde azaltır.

Docker, tekrarlayan, sıradan yapılandırma görevlerini ortadan kaldırır ve geliştirme yaşam döngüsü boyunca masaüstü ve bulut gibi hızlı, kolay ve taşınabilir uygulama geliştirme için kullanılır. Docker'ın kapsamlı uçtan uca platformu, tüm uygulama teslimi yaşam döngüsü boyunca birlikte çalışacak şekilde tasarlanmış kullanıcı arabirimlerini, CLI'leri, API'leri ve güvenliği içerir.

Docker'ın Avantajları Nelerdir?

- **Docker saniyeler içerisinde başlar**, çünkü içerisinde barındırdığı her bir container sadece birer processtir. Böylece karşımıza lightweight bir yapı karşımıza çıkar. Bu da bizi sanal makinelerin hantallığından kurtarmış oluyor aslında.
- **Daha hızlı deployment süreci:** İşte bence en önemli avantajı diyebilirim docker için. Dockerı kullanmak için yeni bir environment kurmaya gerek yoktur. Farklı sunucularda çalışmak isteyen developerlar sadece docker imageleri indirip o sunucuda imageleri çalıştırmaları yeterlidir.
- **Daha Kolay Yönetim ve Ölçeklendirme:** Bir sanal makineye göre docker üzerindeki containerleri çok daha kolay bir şekilde çalıştırabiliriz veya istediğimiz zaman yok edebiliriz. Containerleri manage etmek için farklı toollar mevcut. En çok ta Orchestrator diye nitelendirdiğimiz Kubernetes teknolojisi daha popüler olarak kullanılıyor. Kubernetes, kısaca container kullanan uygulamaların dağıtımını, ölçeklendirmesini ve yönetilmesini otomatik hale getiren açık kaynak kodlu bir sistem.
- **Daha İyi Kaynak Kullanımı** Sanal makinelere göre tek bir sunucu üzerindeki kaynak tüketimi dockerda çok daha verimlidir. Daha az kaynak tüketimi ile daha fazla containeri çalıştırabiliriz.
- **Deployment Verimliliği:** Dockerın en güzel yanlarında birisi de şu: Siz localinizde test ettiniz, uygulamanızı Test veya Live ortamınıza attınız. (veya daha çeşitli ortamlar (dev, staging, pre-prod vs..) Localde çalıştırdığınız her şey burada da aynı şekilde çalışacak.
- **Farklı İşletim Sistemlerine Destek Vermesi** Docker Windows, Linux, MacOs gibi farklı işletim sistemlerine destek verir.
- **Popüler Cloud Servislerle Entegre Edilebilir**. Docker , AWS, Microsoft Azure, Ansible, Kubernetes, Istio ve daha fazla tool ve cloud hizmetlerle entegre şekilde çalışabilir.

https://medium.com/batech/docker-nedir-docker-kavramlar%C4%B1-avantajlar%C4%B1-901b37742ee0

26. Kubernetes Nedir?

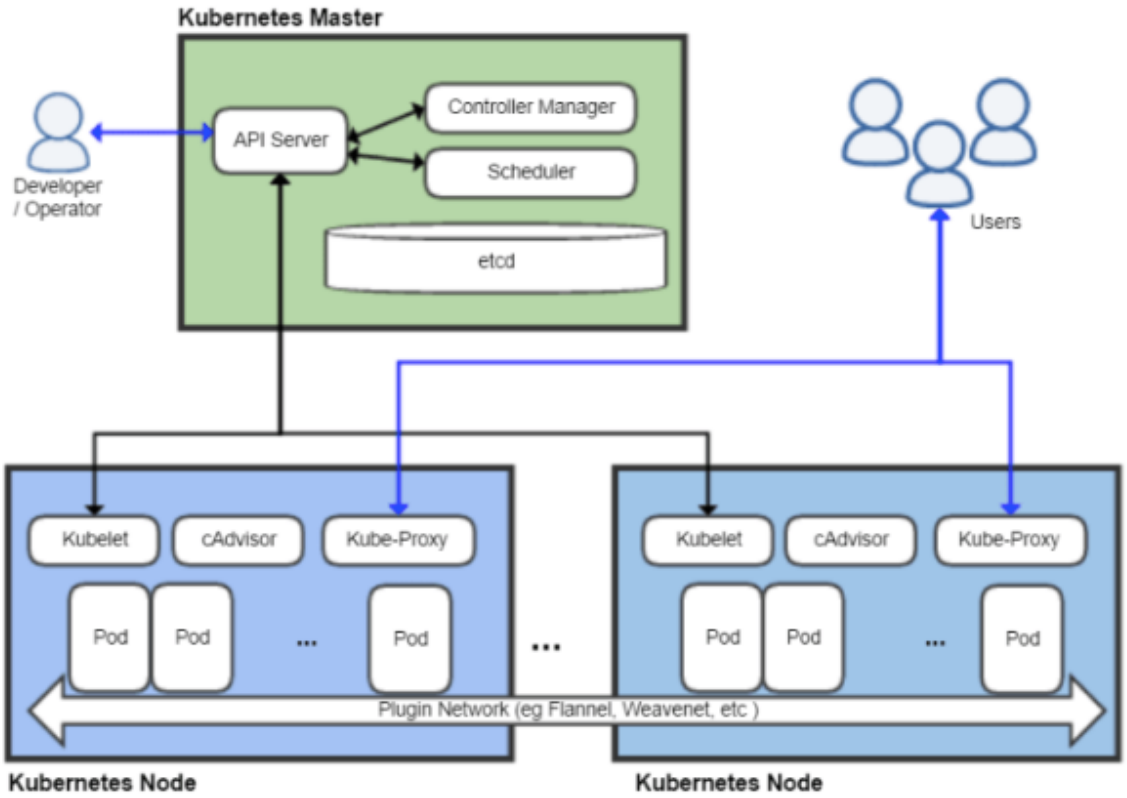
K8s olarak da bilinen Kubernetes, kapsayıcılı uygulamaların dağıtımını, ölçeklendirmesini ve yönetimini otomatikleştirmek için açık kaynaklı bir sistemdir. Kolay yönetim ve keşif için bir uygulamayı oluşturan containerları mantıksal birimler halinde gruplandırır.

Kubernetes Google tarafından GO dilinde geliştirilmiş Cloud Native Computing Foundation tarafından desteklenen mevcut konteyner haline getirilmiş uygulamalarınızı otomatik deploy etmek, sayılarını arttırıp azaltmak gibi işlemler ile birlikte yönetmenizi sağlayan bir Konteyner kümeleme (container cluster) aracıdır.

Kubernetes, kullanıcıları için aşağıdakiler dahil birçok özellik sunar:

- Otomatik kutu paketleme
- IPv4/IPv6 dual-stack
- Toplu yürütme
- Yük dengeleyici
- Zamanlayıcı
- Hizmet keşfi

Kubernetes ayrıca, uygulamalarınızın sağlıklı bir şekilde çalıştığını kontrol etme (ve hatta dağıtım sırasında herhangi bir şeyi olumsuz etkiliyorsa değişikliği tersine çevirme), tercih ettiğiniz depolama sistemini kurma, uygulamalarınızı ölçeklendirme, kendi kendini iyileştirme (container'ları otomatik olarak değiştirme) gibi bir dizi başka otomatik işlevi yerine getirir. Gerektiğinde yanıt vermeyenleri etkisiz hale getir, otomatik ölçeklendirme gerektiğinde, başarısız kapsayıcıları yeniden başlatabilir veya yeniden zamanlayabilir.



27. Bir Uygulamayı Dockerize Etme

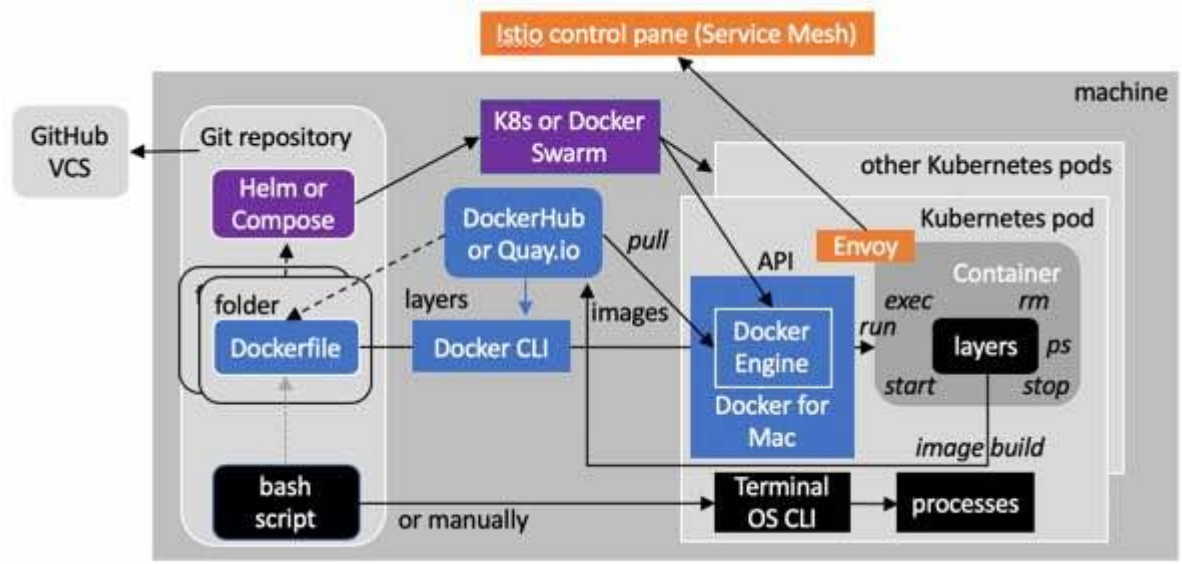
Bir uygulamayı "dockerize etme", bir uygulamayı bir Docker Konteyneri içinde çalışacak şekilde dönüştürme ve bunun için Dockerfile oluşturma işlemidir.

Dockerizing, Docker kapsayıcılarını kullanarak uygulamaları paketleme, dağıtma ve çalıştırma işlemidir. Docker, uygulamanızı gerekli tüm işlevlerle tek bir paket olarak gönderen açık kaynaklı bir araçtır. Docker'ı uygulamanızı, uygulamayı çalıştırmak için ihtiyaç duyduğunuz her şeyle (kütüphaneler gibi) paketlemek ve tek bir paket (bir kapsayıcı) olarak göndermek için kullanabilirsiniz.

Docker'ın iki bölümü vardır:

- **Docker Engine** - taşınabilir bir paketleme aracı
- **Docker Hub** - uygulamaları paylaşmak için bulut hizmet

Docker kapsayıcısı, başka bir kullanıcının bilgisayar ortamını hızlı bir şekilde yeniden oluşturmasına olanak tanır. Kapsayıcı, "kullanıcı alanını" soyutlayarak işletim sanallaştırması sağlar. Bu teknoloji, iş arkadaşlarınızın ürünü sizinle aynı ortamı kullanarak geliştirmesini veya test etmesini sağlar ve bu da daha az hatayla sonuçlanır.



Dockerizing'i Neden Kullanmak İsteyebilirsiniz?

- Kullanımı kolay
- Hızlı
- Tekrarlanabilir bir ortam yaratabilme

Kaynakça :
https://developerexperience.io/articles/dockerizing
https://wilsonmar.github.io/dockerize/

28. JAR Dosyası Nedir?

JAR (Java Arşivi), Java platformunda uygulama yazılımını veya kitaplıklarını dağıtmak için genellikle birçok Java sınıfı dosyasını ve ilişkili meta verileri ve kaynakları (metin, resimler vb.) tek bir dosyada birleştirmek için kullanılan bir paket dosya biçimidir.

Basit bir ifadeyle, bir JAR dosyası, .class dosyalarının, ses dosyalarının, görüntü dosyalarının veya dizinlerin sıkıştırılmış bir sürümünü içeren bir dosyadır. Bir .jar dosyasını, WinZip yazılımı kullanılarak oluşturulan sıkıştırılmış bir dosya (.zip) olarak hayal edebiliriz. WinZip yazılımı bile bir .jar dosyasının içeriğini çıkarmak için kullanılabilir. Böylece bunları kayıpsız veri sıkıştırma, arşivleme, açma ve arşiv paketinden çıkarma gibi görevler için kullanabilirsiniz.

JAR dosyası ne işe yarar?

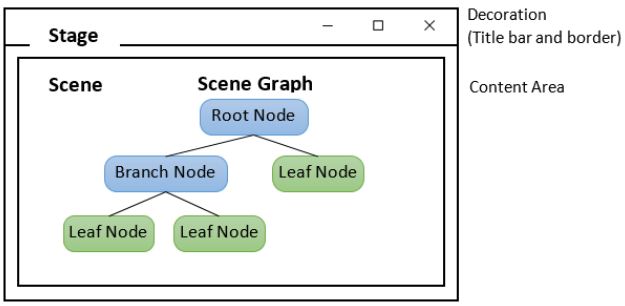
En basit tanımıyla JAR dosyası, her şeyin derli toplu bir şekilde saklanmasını sağlar. İçinde Java ile hazırlanmış bir uygulamanın onlarca farklı bileşenini saklayan bir JAR dosyası sayesinde tüm uygulama, tek bir istek ile verimli bir şekilde dağıtılabilir. Sıkıştırıldıkları için indirme süresini kısaltırlar. JAR dosyası içinde bulunan bildirim dosyaları, bu dosyanın hangi amaçla kullanılacağını açıklar.

29. JavaFX Nedir?

JavaFX, Masaüstü uygulamalarının yanı sıra Zengin İnternet Uygulamaları (RIA) geliştirmek için kullanılan bir Java kitaplığıdır. JavaFX'te yerleşik uygulamalar, Web, Mobil ve Masaüstü Bilgisayarlar dahil olmak üzere birden çok platformda çalışabilir.

JavaFX, Java uygulamalarında bir GUI çerçevesi olarak salınımın yerini alacak şekilde tasarlanmıştır. Ancak, salıncaktan daha fazla işlevsellik sağlar. Swing gibi, JavaFX de kendi bileşenlerini sağlar ve işletim sistemine bağlı değildir. Hafif ve donanım hızlandırmalı. Windows, Linux ve Mac OS dahil olmak üzere çeşitli işletim sistemlerini destekler.

- Kolay ve güvenilir bir şekilde java kütüphanelerinin tamamını kullanmamıza imkan sağlayan zengin görsel, internet uygulama oluşturmamıza imkan sağlar.
- Uygulama içerisinde CSS denetimlerine imkan sağlayarak, görsel açıdan ayarlamalar yapmamıza yardımcı olur.
- Masaüstü ve mobil cihaz servisleri için farklı maskelemeler oluşturmamıza ve optimize çalışması yapmamıza imkan sağlar.
- Ek platformlar tarafından desteklenme imkanı sağlar.

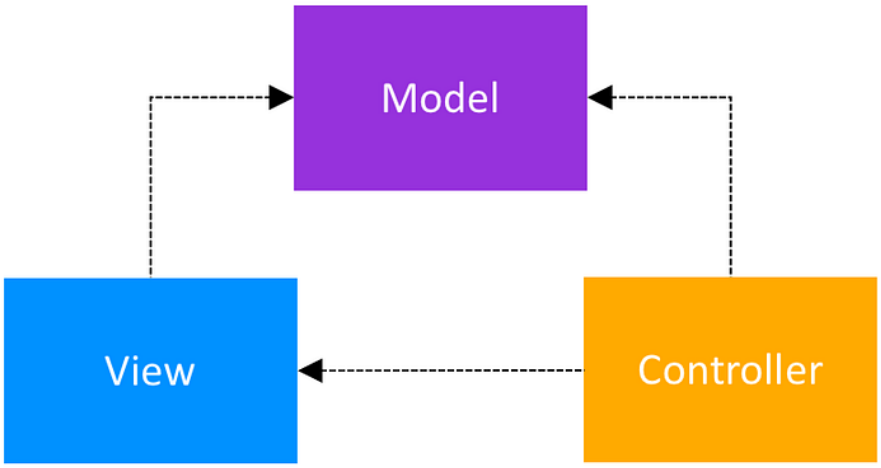


30. MVC

MVC, Yazılım Mühendisliği'nde önemli bir yere sahip architectural patterns (yazılım mimari desenleri)'ın bir parçasıdır. Model, View ve Controller kelimelerinin baş harflerinden oluşan MVC (Model-View-Controller), 1979 yılında Tygve Reeskaug tarafından oluşturulmuş ve yazılım gelişmede bir çok projede kullanılmıştır. Son dönemlerde Microsoft'un MVC desenini Asp.Net teknolojisi ile birleştirmesi ile popülaritesi daha da artmıştır.

MVC ile ilgili en yanlış bilgi, MVC'nin Microsoft tarafından çıkartıldığı düşüncesidir. Yukarıdaki paragrafta da bahsedildiği üzere, MVC'nin Asp.Net'e entegre edilmesinden önce bu deseni bir çok (.Net) yazılım geliştiricisi bilmemekteydi, bilse de kullanmıyorlardı. Asp.Net MVC'nin gelişiminin ardından MVC'ye ilgi oldukça artmış görünüyor.

MVC deseni, 3 katmandan oluşmaktadır ve katmanları birbirinden bağımsız (birbirini etkilemeden) olarak çalışmaktadır. Bu sebeple çoğunlukla büyük çaplı projelerde projelerin yönetiminin ve kontrolünün daha rahat sağlanabilmesi için tercih edilmektedir. MVC ile geliştirilen projelerde projenin detaylarına göre bir çok kişi eş zamanlı olarak kolaylıkla çalışabilmektedir.



Model Nedir?

Model, MVC'de projenin iş mantığının (business logic) oluşturulduğu bölümdür. İş mantığıyla beraber doğrulama (validation) ve veri erişim (data access) işlemleri de bu bölümde gerçekleştirilmektedir.

Model tek katmandan oluşabileceği gibi kendi içinde birden fazla katmandan da oluşabilir. İç yapılandırma projenin büyüklüğü ile yazılım geliştiricinin planlamasına kalmış bir durumdur. Eğer proje büyük çaplı ise modeli birden çok katmana ayırmak projenin yönetimi açısından faydalı olacaktır.

View Nedir?

View, MVC'de projenin arayüzlerinin oluşturulduğu bölümdür. Bu bölümde projenin kullanıcılara sunulacak olan HTML dosyaları yer almaktadır. Projenin geliştirildiği yazılım dillerine göre dosya uzantıları da değişebilmektedir. Projelerin büyüklüğüne göre dikkat edilmesi gereken bir nokta ise, klasörlemedir.

Eğer bir web projesi geliştiriyorsanız, projenin View'larının yer aldığı klasörlerinin hiyerarşisi, ilerleyen dönemlerde karmaşıklığa sebep olmaması için dikkatli yapılmalıdır. Kimi yazılım geliştiriciler web projelerinde HTML dosyaları ile Javascript, CSS ve resim dosyalarını aynı klasör içinde barındırmaktadır. Ufak bir ayrıntı gibi görünse de projenin ilerleyen dönemlerinde ciddi problemler oluşturmaktadır.

View'ın bir görevi de, kullanıcılardan alınan istekleri controller'a iletmektir.

Controller Nedir?

Controller, MVC'de projenin iç süreçlerini kontrol eden bölümdür. Bu bölümde View ile Model arasındaki bağlantı kurulur. Kullanıcılardan gelen istekler (request) Controller'larda değerlendirilir, isteğin detayına göre hangi işlemlerin yapılacağı ve kullanıcıya hangi View'ın döneceği (response) belirtilir.

MVC'nin Yaşam Döngüsü(Life Cycle)

MVC'nin parçaları olan Model, View ve Controller'ın ne olduğu yukarıdaki bölümde anlatıldı. Şimdi bu bilgileri toparlayıp MVC'nin yaşam döngüsünü (çalışma prensibini) detaylıca inceleyelim.

1. **HTTP Request:** Sizin her ASP.NET MVC uygulamasını görüntülemek istemeniz bir request(istek) tir. Bu istediğinizi HTTP üzerinden IIS tarafından alınır. Her yaptığınız istek Server tarafından bir yanıtla son bulması gerekir.
2. **Routing:** ASP.NET MVC uygulamasını her istek yaptığınızda, yaptığınız yanıt UrlRoutingModule HTTP Module tarafından durdurulur. UrlRoutingModule bir isteği durdurduğu zaman, gelen istek RouteTable'dan hangi Controller tarafından üstleneceğine karar verilir.
3. **Controller:** RouteTable'dan gelen route bilgisine göre Controller hangi Action'ı çalıştıracaksa o View çalıştırılır. View, Controller tarafından render edilmez. Controller tarafından geriye ViewResult döndürülür.
4. **ViewResult:** ViewResult, View'i render etmek için aktif View Engine'i çağırır.
5. **ViewEngine**: Bir CSHTML dosyayı oluşturduğunuzda içerisindeki script ve markuplar, Razor View Engin tarafından bazı ASP.NET API'lerini sayfalarınızı HTML'e çevirmek için kullanır.
6. **View:** View Engine tarafından HTML'e çevirilen kodlar kullanıcıya sunulur.
7. **Response:** HTTP üzerinden View kullanıcıya gösterilir.

https://medium.com/@kdrcandogan/mvc-nedir-mvc-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-life-cycle-8e124f24650c

https://www.geeksforgeeks.org/mvc-framework-introduction/

31. Encapsulation

Kapsülleme, nesne yönelimli programlamanın (OOP) temel kavramlarından biridir. Encapsulation, nesnelerin sahip olduğu özellik ve davranışların gizlenmesidir. Türkçe karşılıklarına baktığımızda Sarmalama, Kapsülleme gibi anlamlara gelir. Bu işlem “private” ve “protected” anahtarları ile gerçekleştirilir.

Özellikler eğer “private” olarak tanımlanırlarsa, bu özelliklere yalnızca nesnenin üretildiği sınıf içerisinden erişim mümkün hale gelir ve diğer sınıflardan nesne özelliklerine (doğrudan) erişilemez. Böylece hata yapma riski en aza indirilmiş olur.

Peki nesne özelliklerine nasıl erişeceğiz ?

Nesne özelliklerini “private” olarak tanımladığımızda başka sınıflar içerisinden bu özelliklere erişimleri kısıtlamış olduk. Elbette bu durum asla nesne özelliklerine erişemeyeceğimiz anlamına gelmiyor. Tıpkı arabaya binmek için ihtiyaç duyulan anahtar veya telefonu açmak için girilmesi gereken şifre gibi biz de bir aracı yardımı ile bu özelliklere erişebiliriz. Bu noktada Getter ve Setter Metotlar yardımımıza koşuyor.

Getter & Setter Metodları

Getter ve Setter metotlar, kapsüllenmiş nesne özelliklerine erişmek için kullanılan metotlardır. Bu metotlar sayesinde nesne özellikleri üzerinde işlemler gerçekleştirilebilir.

32. Primitive Types and Wrapper Class Arasındaki Farklar Nelerdir?

Wrapper Class

Java'daki bir Wrapper sınıfı, ilkel bir veri türünü bir nesneye ve nesneyi de ilkel bir türe dönüştürmek için kullanılır. İlkel veri türleri birincil veri türlerini depolamak için kullanılsa bile, Dizi Listeleri ve Vektörler gibi veri yapıları nesneleri depolar. Bu nedenle, dönüştürme için sarmalayıcı sınıfların kullanılması gerekir. char, byte, short ve int ilkel türleri için karşılık gelen sarmalayıcı sınıfları, Character, Byte, Short ve Integer'dır. Long, float, double ve boolean için karşılık gelen sarma sınıfları Long, Float, Double ve Boolean'dır.

Primitive Types

İlkel veri türleri, Java programlama dili tarafından sağlanan önceden tanımlanmış veri türleridir. Sekiz ilkel tip vardır. Bunlar byte, short, int, long, float, double, boolean ve char'dır. Bayt veri türü, 8 bitlik işaretli ikinin tümleyen tamsayısını depolamak için kullanılır. Kısa veri türü, 16 bit işaretli ikinin tümleyen tamsayısını depolamak için kullanılır. 32-bit işaretli ikinin tümleyen tamsayısını saklamak için bir int veri türü kullanılırken, uzun veri türü 64-bit ikinin tümleyen tamsayısını depolamak için kullanılır. Float, tek duyarlıklı 32-bit kayan noktalı değeri depolamak için kullanılır ve double, çift duyarlıklı 64-bit kayan noktalı değeri depolamak için kullanılır. Boole, doğru veya yanlışı temsil etmek için kullanılır. Char, tek bir karakteri depolamak için kullanılır. Bunlar Java'daki sekiz ilkel türdür.

Farklar

Java'daki Wrapper sınıfı ve Primitive Type, programlamada veri depolamak için kullanılabilir.

Wrapper Class vs Primitive Type	
Wrapper sınıfı, ilkel türü nesneye ve nesneyi primitive (ilkel) türe dönüştürmek için bir mekanizma sağlar.	Primitive bir tür, Java tarafından sağlanan önceden tanımlanmış bir veri türüdür.
İlişkili Sınıf	
Bir nesne oluşturmak için bir Wrapper sınıfı kullanılır; bu nedenle, karşılık gelen bir sınıfa sahiptir.	İlkel tür bir nesne değildir, dolayısıyla bir sınıfa ait değildir.
Null Değerler	
Wrapper sınıf nesneleri boş değerlere izin verir.	İlkel bir veri türü boş değerlere izin vermez.
Memory Gereksinimi	
Gerekli hafıza primitif tiplere göre daha fazladır. Kümelenmiş İndeks ek bir boşluk gerektirmez.	Sarmalayıcı sınıflara kıyasla gerekli bellek daha düşüktür.
Collections	
Bir Wrapper sınıfı, ArrayList vb. gibi bir koleksiyonla kullanılabilir.	Koleksiyonlarla birlikte ilkel bir tür kullanılmaz.

33. Heap Memory ve Stack Memory

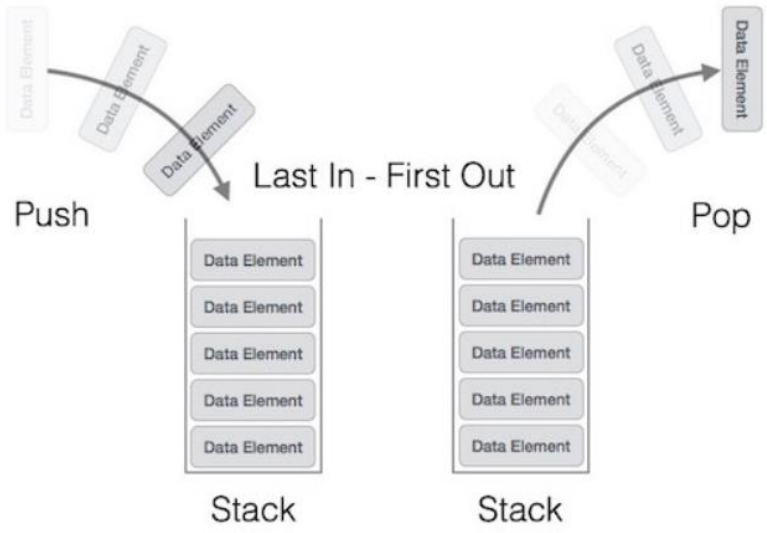
Stack Memory

İşlemcilerin register bilgilerinin tutulduğu yerdir. Burada programınızla ilgili bilgiler (örneğin; lokal değişkenler, referans değişkenler vs) yer almaktadır. Bu memory, geliştirici tarafından değil, compiler tarafından yönetilir. Stack'teki bilgiler kodunuzun derleme aşamasında, direk bellek içine yerleştirildiği için erişimi oldukça hızlıdır.

Programımız işlemci üzerinde çalışabilmek için sürekli stack belleğini arar. Her lokal değişken ve çağrılan her fonksiyon buraya gider. Genellikle istemeden de olsa stack üzerinde birçok hatalarla karşılaşabiliriz. Sıklıkla karşılaştığımız stack buffer overflow ve incorrect memory'ye erişmeye çalışmak bu hatalardan bazılarıdır.

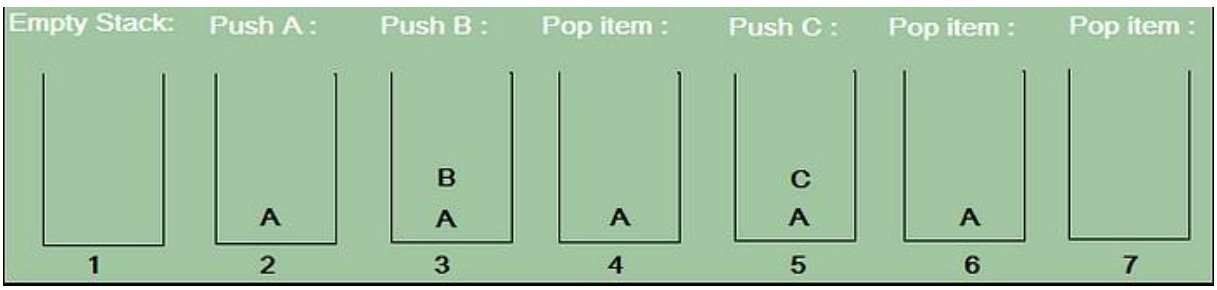
Öncelikle stack bir LIFO (Last In First Out) data structures'dır.

İçine kitapların yalnızca üst üste yerleşebildiği kapağı açık boş bir kutu düşünün. Örnek olarak, kutuya üst üste 5 kitap koydunuz. Kutudan çıkaracağınız ilk kitap 5. koyduğunuz kitap olacaktır. Yani son giren 5. kitap ilk çıkacak kitaptır. LIFO dediğimiz olay da tam olarak budur. Aşağıdaki görselle de bunu somutlaştırabilirsiniz.



Push & Pop

Bu iki işlem birbirlerinin tam olarak tersidir. Push, programdan gelen bir değeri stack'in üzerine eklerken, pop is en yüksek değeri stack'ten çıkarır, boşaltır.



Heap Memory

Bellek üzerinde yer tahsisi yapılan belli bir bölümdür. Bu yer, bellek üzerinde "malloc" fonksiyonu aracılığıyla tahsis edilir ve heap üzerinde allocate edilen(yer tahsisi yapılan) bellek "free" lenerek tekrar kullanım için serbest bırakılır. Heap'teki bellek kullanımı compiler tarafından değil, geliştiriciler tarafından kontrol edilir. Karmaşık programlar oluştururken, genellikle büyük bir bellek alanına ihtiyaç duyarız. Bu durumda Heap Memory kullanırız. Heap üzerinde allocate ettiğimiz bellek operasyonuna "dynamic memory allocation" adı verilir.

Stack'in tersine, geliştiriciler tarafından yönetilen RAM'deki memory bölgesidir. Heap üzerinde bellek ayırabilmek için C dilinin "stdlib" kütüphanesiyle birlikte yerleşik olarak gelen "malloc", "calloc" gibi fonksiyonlar kullanılır. Ayrılan bellek, scope dışına çıkıldığı zaman kullanılmaya devam eder. Bunu önlemek için de yine aynı kütüphane ile birlikte gelen "free" fonksiyonu kullanılarak ayrılan hafıza, heap memory tarafından yeniden kullanılabilmek için serbest bırakılır. Heap memory'nin okunması ve yazılması stack memory'ye göre daha yavaştır, çünkü heap üzerindeki belleğe erişmek için pointer kullanılmak zorundadır.

Stack ve Heap Arasındaki Temel Farklar

Öncelikle ikisi arasındaki en temel fark; Stack Memory'deki değerler son giren ilk çıkar mantığına göre tutulurken, Heap Memory'de bu durum rastegeledir(random). Programlarımızda bu iki belleği birbirinden olabildiğince ayırırız.

- Stack doğrusal bir veri yapısı iken Heap hiyerarşik bir veri yapısıdır.
- Stack bellek asla parçalanmaz, oysa Heap bellek, bellek blokları önce tahsis edilip sonra serbest bırakıldığı için parçalanabilir.
- Stack yalnızca yerel değişkenlere erişirken, Heap değişkenlere genel olarak erişmenize izin verir.
- Stack değişkenleri yeniden boyutlandırılamazken Heap değişkenleri yeniden boyutlandırılabilir.
- Stack bellek bitişik bir blokta tahsis edilirken, Heap bellek herhangi bir rastgele sırada tahsis edilir.
- Stack, değişkenlerin tahsisinin kaldırılmasını gerektirmez, oysa Heap tahsisinin kaldırılması gerekir.
- Stack tahsisi ve serbest bırakma, derleyici talimatları tarafından yapılırken, Heap tahsisi ve serbest bırakma, programcı tarafından yapılır.

https://www.digitalocean.com/community/tutorials/java-heap-space-vs-stack-memory

https://medium.com/@memrekaraaslan/nedir-bu-memory-stack-heap-memory-leak-memory-management-c3c14d1c3e6e

https://www.guru99.com/stack-vs-heap.html

34. Access Modifiers

Java'da iki tür değiştirici vardır: erişim değiştiriciler ve erişim dışı değiştiriciler.

Java'daki erişim değiştiricileri, bir alanın, yöntemin, yapıcının veya sınıfın erişilebilirliğini veya kapsamını belirtir. Üzerine erişim değiştiricisini uygulayarak alanların, yapıcıların, yöntemlerin ve sınıfın erişim düzeyini değiştirebiliriz.

Bir java metodu, değişkeni ya da sınıf oluşturulurken bu öğelere kimlerin erişebileceğini belirtme olanağımız vardır. Bu eylemi gerçekleştirecek olan anahtar kelimelere Erişim Belirleyiciler(Access Modifiers) adını veririz.

Dört tür Java erişim değiştiricisi vardır:

1. **Private:** Bir özel değiştiricinin erişim düzeyi yalnızca sınıf içindedir. Sınıf dışından erişilemez.
2. **Default:** Bir varsayılan değiştiricinin erişim düzeyi yalnızca paket içindedir. Paketin dışından erişilemez. Herhangi bir erişim düzeyi belirtmezseniz, varsayılan olacaktır.
3. **Protected:** Korumalı bir değiştiricinin erişim düzeyi paketin içinde ve paketin dışında alt sınıf aracılığıyladır. Alt sınıfı yapmazsanız, pakete dışarıdan erişilemez.
4. **Public:** Bir genel değiştiricinin erişim düzeyi her yerdedir. Sınıf içinden, sınıf dışından, paket içinden ve paket dışından erişilebilir.

Private

Değişkenler ve yöntemler özel olarak bildirildiğinde, bunlara sınıfın dışından erişilemez. Örneğin,

```
class Data {
    // private variable
    private String name;
}

public class Main {
    public static void main(String[] main){

        // create an object of Data
        Data d = new Data();

        // access private variable and field from another class
        d.name = "Programiz";
    }
}
```

Yukarıdaki örnekte, name adlı özel bir değişken bildirdik. Programı çalıştırdığımızda aşağıdaki hatayı alacağız:

```
Main.java:18: error: name has private access in Data
        d.name = "Programiz";
         ^
```

Data sınıfının private değişkenine Main sınıfından erişmeye çalıştığımız için hata oluşur.

Bu özel değişkenlere erişmemiz gerekip gerekmediğini merak ediyor olabilirsiniz. Bu durumda getters and setters yöntemini kullanabiliriz. Örneğin,

```
class Data {
    private String name;

    // getter method
    public String getName() {
        return this.name;
    }
    // setter method
    public void setName(String name) {
        this.name= name;
    }
}
public class Main {
    public static void main(String[] main){
        Data d = new Data();

        // access the private variable using the getter and setter
        d.setName("Programiz");
        System.out.println(d.getName());
    }
}
```

Yukarıdaki örnekte, name adında özel bir değişkenimiz var. Değişkene dış sınıftan erişmek için getName() ve setName() metotlarını kullandık. Java'da bu yöntemlere getter ve setter denir.

Burada, değişkene değer atamak için ayarlayıcı yöntemini (setName()) ve değişkene erişmek için alıcı yöntemini (getName()) kullandık.

Bu anahtar kelimeyi, sınıfın değişkenine başvurmak için setName() içinde kullandık. Bu anahtar kelime hakkında daha fazla bilgi edinmek için Java this Keyword'ü ziyaret edin.

Default

Sınıflar, yöntemler, değişkenler vb. için açıkça herhangi bir erişim değiştiricisi belirtmezsek, varsayılan olarak varsayılan erişim değiştiricisi dikkate alınır. Örneğin,

```
package defaultPackage;
class Logger {
    void message(){
        System.out.println("This is a message");
    }
}
```

Burada, Logger sınıfı varsayılan erişim değiştiricisine sahiptir. Ve sınıf, defaultPackage paketine ait olan tüm sınıflar tarafından görülebilir. Ancak Logger sınıfını defaultPackage dışında başka bir sınıfta kullanmaya çalışırsak derleme hatası alırız.

Protected

Yöntemler ve veri üyeleri korumalı olarak bildirildiğinde, bunlara aynı paket içinde ve alt sınıflardan erişebiliriz. Örneğin,

```
class Animal {
    // protected method
    protected void display() {
        System.out.println("I am an animal");
    }
}

class Dog extends Animal {
    public static void main(String[] args) {

        // create an object of Dog class
        Dog dog = new Dog();
         // access protected method
        dog.display();
    }
}
```

Yukarıdaki örnekte, Animal sınıfı içinde display() adlı korumalı bir yöntemimiz var. Animal sınıfı, Dog sınıfı tarafından miras alınır. Kalıtım hakkında daha fazla bilgi edinmek için Java Kalıtımı sayfasını ziyaret edin.

Daha sonra Dog sınıfından bir dog nesnesi yarattık. Nesneyi kullanarak üst sınıfın korumalı yöntemine erişmeye çalıştık.

Korumalı metotlara alt sınıflardan erişilebildiği için, Animal sınıfının metoduna Dog sınıfından erişebiliyoruz.

PUBLIC

Yöntemler, değişkenler, sınıflar vb. genel olarak bildirildiğinde, bunlara her yerden erişebiliriz. Genel erişim değiştiricisinin kapsam kısıtlaması yoktur. Örneğin,

```
// Animal.java file
// public class
public class Animal {
    // public variable
    public int legCount;

    // public method
    public void display() {
        System.out.println("I am an animal.");
        System.out.println("I have " + legCount + " legs.");
    }
}

// Main.java
public class Main {
    public static void main( String[] args ) {
        // accessing the public class
        Animal animal = new Animal();

        // accessing the public variable
        animal.legCount = 4;
        // accessing the public method
        animal.display();
    }
}
```

35. Polymorphism

Polimorfizm, bir değişkenin, işlevin veya nesnenin birden çok biçim alma yeteneğini ifade eden nesne yönelimli bir programlama konseptidir. Polimorfizm sergileyen bir programlama dilinde, aynı hiyerarşik ağaca (ortak bir üst sınıftan miras alınan) ait sınıf nesneleri, aynı ada sahip ancak farklı davranışlara sahip işlevlere sahip olabilir.

Örneğin, AnimalSound() adlı bir yöntemi olan Animal adlı bir üst sınıf düşünün. Hayvanların alt sınıfları Domuzlar, Kediler, Köpekler, Kuşlar olabilir - Ve ayrıca kendi hayvan sesi uygulamalarına sahiptirler (domuz oinkleri ve kedi miyavları, vb.):

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}
```

Artık Pig ve Dog nesneleri oluşturabilir ve her ikisinde de animalSound() yöntemini çağırabiliriz:

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}

class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();  // Create a Animal object
        Animal myPig = new Pig();  // Create a Pig object
        Animal myDog = new Dog();  // Create a Dog object
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
```

36. Interface

Java'daki bir arayüz, bir sınıfın planıdır. Statik sabitleri ve soyut(abstract) yöntemleri vardır.

Java'daki arayüz, soyutlamaya ulaşmak için bir mekanizmadır. Java arabiriminde yalnızca soyut yöntemler olabilir, yöntem gövdesi olamaz. Java'da soyutlama ve çoklu kalıtım elde etmek için kullanılır.

Başka bir deyişle, arayüzlerin soyut metotları ve değişkenleri olabileceğini söyleyebilirsiniz. Bir yöntem gövdesine sahip olamaz.

Java Arayüzü aynı zamanda IS-A ilişkisini de temsil eder.
Tıpkı abstract(soyut) sınıf gibi somutlaştırılamaz.

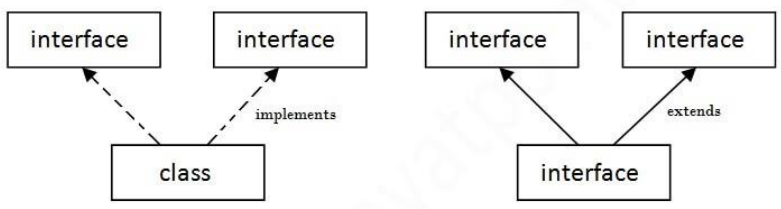
Java Interface neden kullanıyorsunuz?

Interface kullanmak için başlıca üç neden vardır. Aşağıda verilmiştir.

- Soyutlamayı sağlamak için kullanılır.
- Interface ile, çoklu kalıtımın işlevselliğini destekleyebiliriz.
- Gevşek bağlantı elde etmek için kullanılabilir.

```
interface Bank{
    float rateOfInterest();
}
class SBI implements Bank{
    public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
    public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
    public void main(String[] args){
        Bank b = new SBI();
        System.out.println("ROI: "+b.rateOfInterest());
    }}
```

Multiple Interface



Multiple Inheritance in Java

37. Abstract

Veri soyutlama, belirli ayrıntıları gizleme ve kullanıcıya yalnızca gerekli bilgileri gösterme işlemidir.

Soyutlama, soyut sınıflar veya arayüzler (sonraki bölümde hakkında daha fazla bilgi edineceğiniz) ile gerçekleştirilebilir.

abstract anahtar sözcüğü, sınıflar ve yöntemler için kullanılan erişim dışı bir değiştiricidir:

Abstract class: nesne oluşturmak için kullanılamayan kısıtlı bir sınıftır (erişmek için başka bir sınıftan miras alınmalıdır).

Abstract method: Yalnızca soyut bir sınıfta kullanılabilir ve gövdesi yoktur. Gövde, alt sınıf tarafından sağlanır (miras alınır).

ÖRNEK

```
// Abstract class
abstract class Animal {
    // Abstract method (does not have a body)
    public abstract void animalSound();
    // Regular method
    public void sleep() {
        System.out.println("Zzz");
    }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
}

class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

38. Abstraction

Soyutlama, OOP'de ve genel olarak da anahtar bir kavramdır. Gerçek dünyadaki nesneleri düşünün, doğanın nesneleri daha anlaşılır kılmak için ortaya koyduğu soyutlama nedeniyle göremediğimiz elektron, proton ve atom gibi ham maddelerin bir araya gelmesiyle oluşurlar. Benzer şekilde, bilgisayar biliminde soyutlama, donanım ve makine kodunun karmaşıklığını programcıdan gizlemek için kullanılır.

Bu, kullanımı derleme gibi düşük seviyeli dillerden daha kolay olan Java gibi daha yüksek seviyeli programlama dilleri kullanılarak elde edilir.

Java'da Abstraction

Java'da soyutlama, bir kodun uygulama ayrıntılarını gizlemek ve kullanıcıya yalnızca gerekli bilgileri göstermek anlamına gelir. Alakasız ayrıntıları göz ardı ederek ve karmaşıklığı azaltarak karmaşık sistemleri basitleştirme yeteneği sağlar. Java, kendi soyutlamalarımızı oluşturmak için birçok yerleşik soyutlama ve birkaç araç sağlar.

Java'da soyutlama, sağladığı aşağıdaki araçlar kullanılarak gerçekleştirilebilir:

- Abstract Class
- Interface

Abstraction Örneği

```java
//Example of an abstract class that has abstract and non-abstract methods
abstract class Bike{
    Bike(){System.out.println("bike is created");}
    abstract void run();
    void changeGear(){
        System.out.println("gear changed");
    }
}

//Creating a Child class which inherits Abstract class
class Honda extends Bike {
    void run(){System.out.println("running safely..");}
}

//Creating a Test class which calls abstract and non-abstract methods
class TestAbstraction {
    public static void main(String args[]){
        Bike obj = new Honda();
        obj.run();
        obj.changeGear();
    }
}
```

39. Inheritance

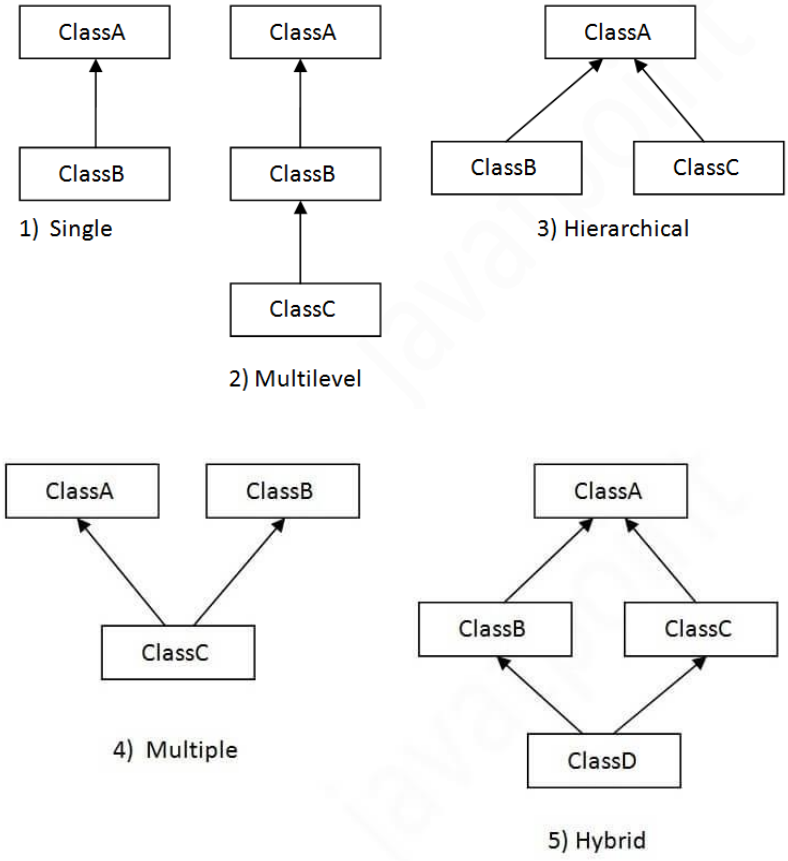
Java'da kalıtım, Nesne Yönelimli Programlamanın temel kavramlarından biridir. Java Kalıtımı, nesneler arasında is-a ilişkisi olduğunda kullanılır. Java'da kalıtım, extends anahtar sözcüğü kullanılarak uygulanır.

Java'da kalıtım, diğer sınıflardan kalıtım yoluyla sınıflar arasında bir hiyerarşi oluşturma yöntemidir.

Java Kalıtımı geçişlidir - bu nedenle Sedan, Araba'yı genişletirse ve Araba, Araç'ı genişletirse, Sedan da Araç sınıfından miras alınır. Araç, hem Otomobilin hem de Sedan'ın üst sınıfı haline gelir.

Kalıtım, java uygulamalarında yaygın olarak kullanılır; örneğin, hata kodları gibi daha fazla bilgi içeren uygulamaya özel bir İstisna sınıfı oluşturmak için İstisna sınıfını genişletmek. Örneğin, NullPointerException.

Java'da Interitance Türleri



.

Kaynakça:

https://www.javatpoint.com/inheritance-in-java

https://www.digitalocean.com/community/tutorials/inheritance-java-example

40. Override

Alt sınıf (alt sınıf), üst sınıfta belirtilen yöntemle aynı yönteme sahipse, Java'da yöntem geçersiz kılma olarak bilinir.

Başka bir deyişle, bir alt sınıf, üst sınıflarından biri tarafından bildirilen yöntemin özel uygulamasını sağlıyorsa, bu yöntem geçersiz kılma olarak bilinir.

Java Method Overriding Kullanımı

- Yöntem geçersiz kılma, üst sınıfı tarafından zaten sağlanan bir yöntemin özel uygulamasını sağlamak için kullanılır.
- Yöntem geçersiz kılma, çalışma zamanı polimorfizmi için kullanılır.

Java Method Overriding Kuralları

1. Yöntem, üst sınıftakiyle aynı ada sahip olmalıdır.
2. Yöntem, üst sınıftaki ile aynı parametreye sahip olmalıdır.
3. Bir IS-A ilişkisi (miras) olmalıdır

```java
//Java Program to demonstrate the real scenario of Java Method Overriding
//where three classes are overriding the method of a parent class.
//Creating a parent class.
class Bank {
    int getRateOfInterest(){return 0;}
}

//Creating child classes.
class SBI extends Bank{
    int getRateOfInterest(){return 8;}
}


class AXIS extends Bank{
    int getRateOfInterest(){return 9;}
}

//Test class to create objects and call the methods
class Test{
    public static void main(String args[]){
        SBI s=new SBI();
        AXIS a=new AXIS();
        System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
        System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
    }
}
```

41. Overloading

Bir sınıfın, aynı ada sahip ancak parametreleri farklı olan birden fazla yöntemi varsa, Yöntem Aşırı Yüklemesi olarak bilinir.

Eğer tek bir işlem yapacaksak metot isimlerinin aynı olması programın okunabilirliğini arttırır.

Verilen sayıları toplamanız gerektiğini varsayalım, ancak herhangi bir sayıda argüman olabilir, iki parametre için a(int,int) ve üç parametre için b(int,int,int) gibi bir yöntem yazarsanız, o zaman adı farklı olduğu için yöntemin davranışını anlamak hem sizin hem de diğer programcılar için zor olabilir.

- Yöntemin aşırı yüklenmesi, programın okunabilirliğini artırır.

Java'da yöntemi aşırı yüklemenin iki yolu vardır.

1. Argüman sayısını değiştirerek
2. Veri türünü değiştirerek

```
class Adder {
    static int add(int a, int b){
        return a+b;
    }
    static double add(double a, double b){
        return a+b;
    }
}

class TestOverloading {
    public static void main(String[] args){
        System.out.println(Adder.add( a: 11, b: 11));
        System.out.println(Adder.add( a: 12.3, b: 12.6));
    }
}
```

42. POJO

Java'daki POJO(Plain Old Java Object), Düz Eski Java Nesnesi anlamına gelir. Herhangi bir özel kısıtlamaya bağlı olmayan sıradan bir nesnedir. POJO dosyası herhangi bir özel sınıf yolu gerektirmez. Bir Java programının okunabilirliğini ve yeniden kullanılabilirliğini artırır.

POJO'lar, kolay bakımları nedeniyle artık geniş çapta kabul görmektedir. Okuması ve yazması kolaydır. Bir POJO sınıfı, özellikler ve yöntemler için herhangi bir adlandırma kuralına sahip değildir. Herhangi bir Java Çerçevesine bağlı değildir; herhangi bir Java Programı kullanabilir.

POJO terimi, 2000 yılında Martin Fowler (Amerikalı bir yazılım geliştiricisi) tarafından tanıtıldı. Java'da sun microsystem tarafından EJB 3.0'dan edinilebilir.

Genel olarak, bir POJO sınıfı değişkenleri ve bunların Getters ve Setter'larını içerir.

POJO sınıfları, her ikisi de okunabilirliği ve yeniden kullanılabilirliği artırmak için nesneleri tanımlamak için kullanıldığından Beans'e benzer. Aralarındaki tek fark, Bean Dosyalarının bazı kısıtlamalara sahip olmasıdır, ancak POJO dosyalarının herhangi bir özel kısıtlaması yoktur.

ÖRNEK

POJO sınıfı, nesne varlıklarını tanımlamak için kullanılır. Örneğin, nesnelerini tanımlamak için bir Çalışan POJO sınıfı oluşturabiliriz.

Aşağıda bir Java POJO sınıfı örneği verilmiştir:

```java
// POJO class Example
public class Employee {
    private String name;
    private String id;
    private double sal;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public double getSal() {
        return sal;
    }
    public void setSal(double sal) {
        this.sal = sal;
    }
}
```

43. Java Bean

Bir JavaBean, aşağıdaki kurallara uyması gereken bir Java sınıfıdır:

- No-arg Constructor'a sahip olmalıdır.
- Serializable olmalıdır.
- Getter and Setters yöntemleri olarak bilinen özelliklerin değerlerini ayarlamak ve almak için yöntemler sağlamalıdır.

Neden Kullanırız?

Java teknik incelemesine göre, yeniden kullanılabilir bir yazılım bileşenidir. Fasulye, birçok nesneyi tek bir nesneye sığdırır, böylece bu nesneye birden çok yerden erişebiliriz. Üstelik kolay bakım sağlar.

```
public class Employee implements java.io.Serializable{

    private int id;
    private String name;

    public Employee(){}

    public void setId(int id){
        this.id=id;
    }
    public int getId(){
        return id;
    }
    public void setName(String name){
        this.name=name;
    }
    public String getName(){
        return name;
    }
}
```

JavaBean sınıfına erişmek için getter ve setter metotlarını kullanmalıyız.

```
public class Test {
    public static void main(String args[]) {
        Employee e = new Employee();//object is created
        e.setName("Arjun");//setting value to the object
        System.out.println(e.getName());
    }
}
```

44. Java @Builder

Project Lombok'un @Builder'ı, Builder kalıbını ortak kod yazmadan kullanmak için yararlı bir mekanizmadır. Bu ek açıklamayı bir Sınıfa veya yönteme uygulayabiliriz. Bu hızlı öğreticide, @Builder için farklı kullanım durumlarına bakacağız.

Lombok'un @Builder ek açıklaması, standart kodu azaltmayı amaçlayan oluşturucu modelini uygulamak için yararlı bir tekniktir.

@Builder annotation'ı, açıklamalı POJO sınıfları için karmaşık oluşturucu API'leri üretir.

Örneğin, @Builder ile açıklama eklenmiş bir Sınıf Makalesine açıklama eklersek, oluşturucu API'sini kullanarak Makale örnekleri oluşturabiliriz. Dahili olarak bir ArticleBuilder sınıfı ve bir build() yöntemi, oluşturucu sınıfının her parametresi için ayarlayıcı benzeri yöntemlerle birlikte otomatik olarak oluşturulur.

```java
import java.util.Collections;
import java.util.List;
import lombok.Builder;
import lombok.Getter;
import lombok.ToString;

@Builder
@Getter
@ToString
public class Article {
    private Long id;
    private String title;
    private List<String> tags;


    public static void main(String[] args) {
        Article a = Article.builder()
                .id(1L)
                .title("Test Article")
                .tags(Collections.singletonList("Demo"))
                .build();
    }
}
```

@Getter ek açıklaması, erişimci yöntemlerini ekler ve @ToString ek açıklaması, ilgili alanların değerlerini yazdıran toString() yöntemini geçersiz kılar.

https://projectlombok.org/features/Builder

https://www.baeldung.com/lombok-builder

45. Singleton Design Pattern

Singleton Pattern, Java'daki en basit tasarım kalıplarından biridir. Bu tür tasarım deseni, bir nesne yaratmanın en iyi yollarından birini sağladığından, yaratıcı desen kapsamına girer.

Bu model, yalnızca tek bir nesnenin oluşturulmasını sağlarken bir nesne oluşturmaktan sorumlu olan tek bir sınıfı içerir. Bu sınıf, sınıfın nesnesini başlatmaya gerek kalmadan doğrudan erişilebilen tek nesnesine erişmenin bir yolunu sağlar.

UYGULAMA

SingleObject sınıfı oluşturacağız. SingleObject sınıfının yapıcısı özeldir ve kendisinin statik bir örneğine sahiptir.

SingleObject sınıfı, statik örneğini dış dünyaya ulaştırmak için statik bir yöntem sağlar. SingletonPatternDemo, demo sınıfımız, bir SingleObject nesnesi elde etmek için SingleObject sınıfını kullanacaktır.

Bir Singleton Sınıfı oluşturun. (*SingleObject.java*)

```java
public class SingleObject {

    //create an object of SingleObject
    private static SingleObject instance = new SingleObject();

    //make the constructor private so that this class cannot be
    //instantiated
    private SingleObject(){}

    //Get the only object available
    public static SingleObject getInstance(){
        return instance;
    }

    public void showMessage(){
        System.out.println("Hello World!");
    }
}
```

Singleton sınıfından tek nesneyi alın.

```java
public class SingletonPatternDemo {
    public static void main(String[] args) {

        //illegal construct
        //Compile Time Error: The constructor SingleObject() is not visible
        //SingleObject object = new SingleObject();

        //Get the only object available
        SingleObject object = SingleObject.getInstance();

        //show the message
        object.showMessage();
    }
}
```

46. Abstract ve Inheritance Arasındaki Farklar

S.No.	Abstract Class	Interface
1.	Soyut bir sınıf, hem soyut hem de soyut olmayan yöntemleri içerebilir.	Arayüz yalnızca soyut yöntemleri içerir.
2.	Soyut bir sınıf dördüne de sahip olabilir; statik, statik olmayan ve nihai, nihai olmayan değişkenler.	Yalnızca nihai ve statik değişkenler kullanılır.
3.	Soyut sınıfı bildirmek için soyut anahtar kelimeler kullanılır.	Arayüz, interface anahtar sözcüğü ile bildirilebilir.
4.	Çoklu kalıtımı destekler. (Multiple Inheritance)	Çoklu kalıtımı desteklemez.
5.	'Extend' anahtar kelimesi soyut bir sınıfı genişletmek için kullanılır.	implement anahtar sözcüğü interface'i uygulamak için kullanılır.
6.	Private, protected vb. gibi sınıf üyelerine sahiptir.	Varsayılan olarak public sınıf üyeleri vardır.

47. Private Constructor

Private constructor, bir sınıfın örneklenmesini (instance oluşturmasına) izin vermez. Basitçe söylemek gerekirse, sınıfın kendisi dışında herhangi bir yerde sınıf örneklerinin oluşturulmasını engellerler. Birlikte kullanılan public ve private constructorlar, sınıflarımızı nasıl başlatmak istediğimiz üzerinde kontrol sağlar - bu, constructor delegasyonu olarak bilinir.

Kullanıldığı Yerler:
Singleton Pattern, Delegating constructors, Uninstantiable Classes, Builder Pattern

```java
public class PrivateConstructorDemo {
    private static PrivateConstructorDemo pcd;
    private PrivateConstructorDemo() {}

    public static PrivateConstructorDemo getInstance() {
        if(pcd == null) {
            pcd = new PrivateConstructorDemo();
        }
        return pcd;
    }

    public static void main(String args[])
    {
        PrivateConstructorDemo pcd = PrivateConstructorDemo.getInstance();
        PrivateConstructorDemo pcd1 = PrivateConstructorDemo.getInstance();
        System.out.println(pcd.equals(pcd1));
    }
}
```

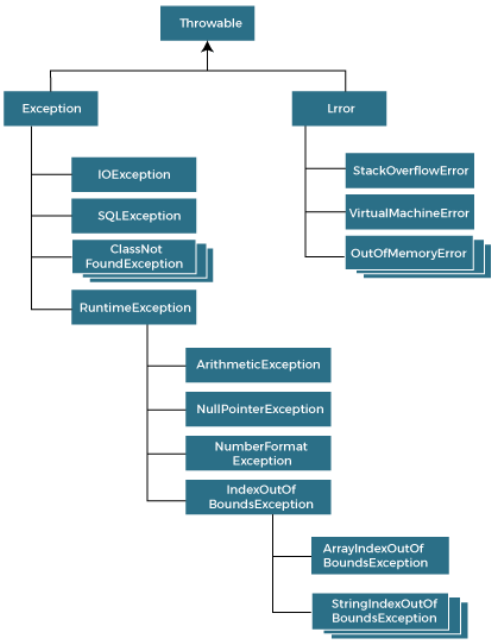
48. Exception Handling

Java'daki İstisna İşleme, uygulamanın normal akışının sürdürülebilmesi için çalışma zamanı hatalarını işleyen güçlü mekanizmalardan biridir.

Sözlük Anlamı: İstisna, anormal bir durumdur. Java'da istisna, programın normal akışını bozan bir olaydır. Çalışma zamanında atılan bir nesnedir.

Exception Handling, ClassNotFoundException, IOException, SQLException, RemoteException, vb. gibi çalışma zamanı hatalarını işleyen bir mekanizmadır.

Avantajı: İstisna işlemenin temel avantajı, uygulamanın normal akışını sürdürmektir. Bir istisna normalde uygulamanın normal akışını bozar; bu yüzden istisnaları ele almamız gerekiyor.



Java Exception Anahtar Kelimeleri

Kelime	Tanım
try	"try" anahtar sözcüğü, bir istisna kodu yerleştirmemiz gereken bir bloğu belirtmek için kullanılır. Bu, try bloğunu tek başına kullanamayacağımız anlamına gelir. try bloğunun ardından ya catch ya da finally gelmelidir.
catch	"catch" bloğu, istisnayı işlemek için kullanılır. Try bloğundan önce gelmelidir, bu da catch bloğunu tek başına kullanamayacağımız anlamına gelir. Bunu daha sonra nihayet blok takip edebilir.
finally	" finally" bloğu, programın gerekli kodunu yürütmek için kullanılır. Bir istisna ele alınsa da alınmasa da yürütülür.
throw	Bir istisna atmak için "throw" anahtar sözcüğü kullanılır.
throws	İstisnaları bildirmek için "throws" anahtar sözcüğü kullanılır. Yöntemde bir istisna olabileceğini belirtir. Bir istisna atmaz. Her zaman metot imzasıyla kullanılır.

49. Spring Boot Database Initialization

Spring Boot veritabanı başlatma hakkında her şeyi öğrenelim. Bir veritabanını farklı şekillerde oluşturabilir ve doldurabiliriz. Bunu manuel olarak yapmak yaygın olsa da, otomatik bir yaklaşım kötü bir şey değildir.

Örneğin, Spring Boot, JPA modülü aracılığıyla Şema ve Veri başlatma sunar. Aşağıdaki özellikleri kullanarak bu davranışı kontrol edebiliriz.

Bu iki seçenek arasındaki temel fark, birincisinin satıcıdan bağımsız olmasıdır. Bildiğiniz gibi Spring JPA, sağlayıcısı olarak hibernate'i kullanıyor. Ancak yeterli yapılandırma ile Eclipse-link, OpenJPA, DataNucleus, vb. gibi diğer satıcıları kullanabilirsiniz. Bu gibi durumlarda, Spring Boot satıcıya göre uygun bir başlatma modu seçecektir.

Database Schema initialization through Hibernate

Henüz fark etmediyseniz, Yapılandırma adı her şeyi söylüyor. Hazırda bekletme satıcısına özgü, Spring JPA ile ilgili bir özelliktir. Ve DDL ile ilgisi var.

spring.jpa.hibernate.ddl-auto hiçbiri, doğrulama, güncelleme, oluşturma ve oluşturma-bırakma işlemlerinden birini alır. Bu seçeneklerden birini açıkça belirterek, Spring Boot'a şemayı nasıl başlatacağını öğretiyorsunuz.

Kelime	Tanım
none	Veritabanı Şeması başlatma yok
create	Uygulama başlangıcında şemayı bırakır ve oluşturur. Bu seçenekle, her açılışta tüm verileriniz kaybolacaktır.
create-drop	Başlangıçta şema oluşturur ve bağlam kapanışında şemayı yok eder. Birim testleri için kullanışlıdır.
validate	Yalnızca Şemanın Varlıklarla eşleşip eşleşmediğini kontrol eder. Şema eşleşmezse uygulama başlatma işlemi başarısız olur. Veritabanında herhangi bir değişiklik yapmaz.
update	Şemayı yalnızca gerekirse günceller. Örneğin, bir varlığa yeni bir alan eklendiyse, verileri bozmadan tabloyu yeni bir sütun için değiştirir.

Kaynakça:

https://springhow.com/spring-boot-database-initialization/

50. Iterable

JDK 1.5'te tanıtıldı. Java.lang paketine aittir. Genel olarak, Iterable Gerçekleştiren bir nesne, yinelenmesine izin verir. Yinelenebilir bir arabirim, bir nesnenin gelişmiş for döngüsü (for-each loop) hedefi olmasını sağlar.

Iterable nesnelerinin yinelenebilmesinin üç yolu vardır.

- Gelişmiş for loop(for-each loop) kullanma
- Iterable forEach döngüsünü kullanma
- Iterator<T> arayüzünü kullanma

For Döngüsü ile Iterable

```
class IterateUsingEnhancedForLoop {
    public static void main (String[] args) {

        // create a list
        List<String> list = new ArrayList<String>();

        // add elements
        list.add("Geeks");
        list.add("for");
        list.add("Geeks");

        // Iterate through the list
        for( String element : list ){
            System.out.println( element );
        }
    }
}
```

Foreach Döngüsüyle Iterable

```
class IterateUsingforEach {
    public static void main(String[] args)
    {
        // create a list
        List<String> list = new ArrayList<>();

        // add elements to the list
        list.add("Geeks");
        list.add("for");
        list.add("Geeks");

        // Iterate through the list
        list.forEach(
                (element) -> { System.out.println(element); });
    }
}
```

KAYNAKÇA

https://www.javatpoint.com/
https://www.w3schools.com/
https://www.baeldung.com/
https://stackoverflow.com/
https://swagger.io/
https://docs.spring.io/
https://www.ibm.com/
https://www.tutorialspoint.com/