# OOP (Nesneye Yönelik Programlama)

## Nesneye Yönelik Programlamanin Temel Kavramları

* **Nesne (Object);** Kendine has özellikleri olan ve özelliklere bağlı olarak belirli bir durumda bulunarak çeşitli tutumlar sergileyen somut ya da soyut varlıklardır.
* **Sınıf (Class);** Aynı türden nesnelerin oluşturduğu bir kümedir.
* **Metot;** Bir nesnenin yapabileceklerini belirleyen kavramdır.
* **Kalıtım (Inheritance);** Bir sınıfın belirli özellik ve tutumlarını bir üst sınıftan alarak, kendisi için farklı olan özellik ve tutumları ayrıca uyguluyor olmasıdır.
* **Soyutlama (Abstraction);** Nesnenin uygulama için gerekli olan özelliklerinin kodlanarak, diğer özelliklerinin kodlanmaması anlamına gelir.
* **Çok biçimlilik (Polymorphism);** Nesnelerin kendi türlerinin dışında farklı türler olarak da temsil edilmelerini sağlar.
* **Kapsülleme (Encapsulation);** Nesnenin yaptığı işler ve durumuyla ilgili bilgilerin ve işlevsel karmaşıklığının gizlenmesi anlamına gelir.

## Erişim Belirleyiciler (Access Modifiers)

Access modifiers, sınıflara nerelerden ve ne şekilde erişileceğini belirtir. 4 adet anahtar kelimesi vardır.

* **Public :** Kod içerisinde herhangi bir yerden erişilebilir, hiç bir kısıtlama yoktur.
* **Protected :** Sadece bulunduğu class ve ondan türetilen diğer sınıflar içerisinden erişilebilir.
* **Private :** Sadece kendi class’ı içinden erişilebilir.
* **Internal :** Aynı program içerisinden(.dll veya .exe) erişilebilir, fakat farklı bir program içerisinden yani dışardan erişilemez durumdadır.

## 3. OOP Temel Prensipleri(Inheritence,Polymofizm Vs)

### 3.1. Inheritance (Kalitim/Miras Alma)

Kalıtım: Bazı özellikleri tekrar tekrar yazmak yerine kullanabileceğimiz bir yapı. Örnek vermek gerekirse bir ailemiz var. Ve bu ailenin bazı özellikleri anne ve babadan gelir. Bu anne babadan gelen özellikleri tekrar çocuk yazmak yerine , anne-babadan referans kullanarak bu bilgileri çekebiliriz. Fakat bununla birlikte çocuğun kendine has özelliklerini ayrıca da ekleyebiliyoruz. Kalıtımın en sade ifadesiyle böyle anlatabilecğimi düşünüyorum.

   public class cocuk extends annebaba {

//Yazılacak Kodlar

}

### 3.2. Encapsulation (Kapsülleme/Sarmallama)

Kapsülleme kavramı bir nesnenin bazı özellik ve işlevlerini başka sınıflardan ve nesnelerden saklamaktır. Kapsülleme (Encapsulation) sayesinde nesneler bilinçsiz kullanımdan korunmuş olur.  Genel tanımıyla kullanıcı tarafından **verilerin**, **sınıfların** ve **metotların** ne kadarının görüntülenebileceği ve değiştirilebileceğinin sınırlarının konulmasını sağlar.  Bir örnek ile açıklayacak olursak;  
  
  
**Örnek 1:**Ulke adında bir class’ımız var ve bu class Ad, Baskent, Nufus ve ParaBirimi’ni alıyor.  
Biz para birimini default olarak TL ayarlıyoruz ve get’te bunu gösteriyoruz. Ama eğer set edilirse set edileni gösterebiliriz . (Property i get ve setle kullanmaktır özetle)

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Örn2: Unvan null ise set metodda default bir şey set edilme kısmı

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

### 3.3. Abstraction (Soyutlama)

Öncelikle soyutlamanın Türkçe olarak ne ifade ettiğini kavrayalım; soyutlama bir metayı meydana getiren karmaşık oluşumları göz ardı ederek, o metanın işe yarayabilecek en öz haline inebilmektir. Diğer anlamda, temel özellikleri özetleyerek temel olmayan ayrıntıları dışarıda bırakmaktır.  
Bu bağlamda **soyutlama**: nesnenin program için ihtiyaç duyulan özelliklerin arka plan ayrıntıları içermeksizin ifade ediliş biçimidir.  
**Örneğin;** Console.WriteLine(“Merhaba”); ile ekrana bir şeyler yazdırırız arka planda neler döndüğünü bilmeden. İşte bu abstraction prensibidir. Daha derli toplu bir ifadeyle objelerin ayrıntılarıyla uğraşmak yerine yalnızca girdi ve çıktılarına odaklanarak tasarımı daha iyi oluşturmayı ve anlamayı sağlamaktır.Bu soyutlamayı başlı başına ayrı konular olan **Interface(arayüz)** ve **Abstract** **sınıflar** ile sağlayabiliriz.

### 3.4. Polymorphism (Çok Biçimcilik)

En basit tanımıyla; bir nesnenin birden fazla nesne gibi davranma şeklidir. Bir sınıf başka bir sınıftan türetildiğinde tüm özelliklerini miras alır. Türemiş sınıf içinde türetilen sınıftan gelen bazı özellikleri değiştirmek isteyebiliriz (ve yapabiliriz de). Sınıfın miras aldığı özelliklerin değiştirilebilmesine -ve nesne üzerinden çağrılabilmesine çok biçimlilik denir.

Örn1: sürmek bir sınıf olsun, araba sürmek için gerekli özellikler farklı, bisiklet sürmek için gerekli özellikler farklı.

Örn2: Hayvan bir sınıf olsun. Bu sınıfta konuşma isimli bir fonksiyon içersin. Hayvan sınıfını kalıtan kuş köpek ve kedi sınıflarımz olsun. Her hayvan için konuşma özelliği farklı olacaktır. Method override ederek.

## 4.Abstract Kavramı- Soyut Sınıf ve Metodlar

### Abstract Kavramı:

* Soyut sınıflar kalıtım özelliğini kullanarak kod tekrarını azaltır.

### 4.1. Abstract Class

* Abstract classlardan nesne oluşturulamaz.
* Abstract classlardan intercafeler gibi objeler oluşturamıyoruz. Ancak abstract classların referansları alt sınıflara direk olarak referans olabiliyor.
* Abstract classlar hem gövdeli hem de gövdesiz metotlar bir arada bulunuyor. Yani hem somut hem de soyut metotlarımız bir arada bulunuyor.
* Abstract classlara özellikler eklenebiliyor.
* Bir classın abstract class olması demek en az 1 tane abstract metotunun yani bir tane gövdesiz metodu bulunduğu anlamına geliyor.
* Abstract classlarla oluşturulan bir tasarımda bir metotun alt sınıflarda mutlaka yazılması gerekiyorsa bu metodun bu sınıflara özgü olması için bu metodun abstract olarak tanımlanması bizim için çok daha doğru olacaktır.

### 4.2. Abstract Metod

* Abstract method, sadece Abstract Class’ların içerisinde tanımlanır ve Abstract Class’ı kalıtan sınıf tarafından override edilmek zorundadır.
* Oluşturduğumuz Abstract Class içerisine, Abstract Method yazılırken gövdesi yazılmaz ve daha sonra Abstract Class’ımızı kalıtacağımız sınıfta Abstract Method’u override ederiz.
* Abstract Method, sadece Abstract Class’ların içerisinde tanımlanabilen, tanımlanırken gövde bulundurmayan, daha sonra içerisinde bulunduğu Abstract Class kalıtılırken override edilen Method‘lardır.
* Abstract Method’lar, Private olarak tanımlanamaz.

## 5. Interface Kavramı

* Interface Classlara veya struct’lara  rehberlik(kılavuzluk) eden(ne yapacağını söyleyen) yapılardır.

**Tanım:**

* Microsoft’un sitesindeki yer alan tanıma göre interface: “Bir arabirim, **soyut olmayan bir sınıfın veya yapının uygulanması gereken ilgili işlevler grubu için tanımlar içerir”**.

**Örnek**:

* Örneğin; **İnsanda hayvanda bir canlıdır. Ortak özellikleri ve birbiri ile aynı olan ve birbirinden ayıran özellikleri ve fonksiyonları vardır. Buradaki birbirinden ayıran özellik ve fonksiyonları ile somutlama yapmak durumundayız.** **Bu işlemi ara birimler ile yaparız**. Aynı şekilde bir diğer örnek ise arabalardır. **Araçlardır : marka, model, tip olarak bir çok şekilde birbirinden ayrılır. Burada bu ayrımı(soyutlama) yapabilmek için yine interface’ler kullanmak zorundayız.**

**Özellikler:**

* Sadece kendisinden türeyen sınıfların içini doldurmak zorunda olduğu içi boş metot tanımlarının yapıldığı bir yapıdır.
* Genellikle çoklu kalıtım ihtiyacını gidermek için kullanılırlar.
* İçinde değişken tanımlanmaz ve erişim kısıtlayıcılar(access modifiers) interface’in içinde yazılamaz.

## 6. Interface ile Abstract Class Arasındaki Farklar

* Interface ve abstract class’lar new anahtar sözcüğü ile oluşturulamazlar.
* Bir sınıf birden fazla interface’i kalıtım olarak alınabilir ama bir sınıfa bir tane abstract class kalıtım alınabilir.
* Interface içerisinde boş metodlar tanımlanabilir ama abstract class’larda hem boş metodlar tanımlanabilir hemde içi dolu metodlar tanımlabilir.
* Abstract sınıflar içerisinde metod gövdeleri tanımlanıp özellik değerleri ayarlandığı için genellikle sonradan geliştirilmek için kullanılıır ama interface ise body ve değer set edilemediği için tamamen interface üzerinden tüm üyeleri implemente edilerek geliştirmelere yapılması gereken durumlarda kullanılır.
* Abstract class’lar içerisinde sadece abstract olarak işaretlenmiş metod ve özellikler implement edilmek zorundadır fakat interface içerisindeki tüm özellik ve metodlar implement edilmek zorundadır.
* Bir class bir tane abstract class’ı kalıtım olarak alabilir ama bir class istenilen sayıda interface’i kalıtım olarak alabilir.
* Interface içerisinde özellik ve metodlarda erişim belirleyiciler kullanılmaz herşey public olarak kabul edilir fakat abstract sınıflarda kullanılabilir.
* Abstract sınıflara diğer sınıf ve interface’ler kalıtım olarak geçilebilir fakat interface’e herhangi bir yapı kalıtım olarak geçilemez.

## 7. Statik Kavramı

* Bir sınıf(class) içerisinde bulunan metodlar(methods) static olarak tanımlanabilir.
* Bir sınıf(class) içerisinde bulunan alanlar (fields) static olarak tanımlanabilir.
* Bir sınıfa ait static bir yapıcı metod (constructors) tanımlanabilir.
* Değişmezler (constants) bilinçsiz olarak (implicit) static tanımlanmışlardır.
* Readonly referanslar açıkça belirtilmedikçe static değildirler.

### 7.1. Static class ile normal class arasındaki fark

Static metotlar yeni bir instance yazılarak çağrılamaz, sadece statik üye içerebilirler, başka bir sınıftan inherit edilemez.

### 7.2. Statik Metod Çağırımı

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 8. Const ve ReadOnly Kavramı

### 8.1. Const Kavramı

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

### 8.2. ReadOnly Kavramı

* İsminden de anlaşılacağı üzere sadece okunabilir demektir.
* Nesnelere **ilk değerlerini verdikten sonra tekrar değiştirilmesini istemiyorsak** bunu yapabilmenin bir diğer yöntemi **Readonly kullanımıdır.** **Yalnız burada ilk değer yapıcı metod içinde değiştirilebilmektedir.** Önemli olan bizim için yapıcı metod dışında diğer metodlar içinde sadece okunabilir olmasıdır.
* Readonly değişkenler tanımlanacağı zaman **Class seviyesinde tanımlanırlar. Metodlar içinde tanımlanamazlar.**
* **Önemli ve const’dan ayıran bir diğer özelliği ilk değer ataması yapılmadan tanımlanabilir.**
* **Değişkene değer ataması yalnızca Yapıcı(Constructor) Metod içinde gerçekleştirilebilir. Bu sayede metodlar içinde sadece okunabilir özelliği ile yer alacaktır.**
* Yapıcı metod içinde değeri birden fazla kez değiştirilebilir. Son değeri tüm metodlar tarafından okunabilir değerini ifade eder.

### 8.3. Const ve ReadOnly Farkları

* **const**:**Class seviyesinde tanımlanır ve tanımlanma anında değeri verilmek zorundadır**. Sonradan değeri değiştirilemez.
* **readonly**:Class seviyesinde tanımlanır. **Tanımlandığı anda değeri verilebilir veya Class Constructor’ında değeri verilebilir.** Sonradan değeri değiştirilemez.

# 9.Final Kavramı

## 9.1. Final Sınıf Değişkenleri:

* Final olan bir sınıf değişkenine sadece bir kere değer ataması yapılabilir ve bu atama sadece sınıf contructor da gerçekleşebilir.
* Final değişkenler ilk değerini aldıktan sonra bir daha değiştirilemiyorlar. Örnek :
* final int a=4;
* int a=3; // –> Hatalı’dırr final değer ilk değeri aldıktan sonra değiştirilemez..
* Not : \* Eğer final değişkeni tanımlayıp ilk değeri belirtmezseniz yine hata alırsınız. null,0 gibi değerler vermesini beklemeyin.

## 9.2. Final Class Kavramı

* Final metodlara benzer bir yapı vardır. Final kelimesi oluşturulan sınıflarda kalıtım yapılamaz. Yani sınıfın alt sınıfları oluşturulamaz.

## 9.3. Final Metot ve Parametreleri

* Final method override olmasına izin vermeyen method’tur..(Override : Kalıtım yoluyla miras olarak aldığımız metodu ezmektektir..)

**Final Metod Parameter**

* Final olarak tanımlanmış bir metot parametresine sadece bir kere değer atanabilir. Metot parametrelerinin tamamen final olarak tanımlamış olmalarında büyük fayda vardır. Bu şekilde parametrenin metot bünyesinde değişikliğe ugrama tehlikesi ortadan kaldırılmış olur.

# SOLID Yazılım Geliştirme Prensipleri

## Single Responsibility Principle

Türkçe karşılığı “Tek Sorumluluk” anlamına gelen bu prensipte amaç; geliştirilen projede bir güncelleme veya değişiklik yapılması istendiğinde kodların içinde kaybolmadan, yalnızca ilgili metoda giderek istenilen değişikliğin yapılmasının sağlanmasıdır. Biraz daha açacak olursak; **bir fonksiyona birden fazla iş verip onu birçok işten sorumlu tutmak yerine, her bir iş için ayrı bir metot oluşturmalı ve ilerleyen zamanlarda bir değişiklik yapılacağında da kolaylıkla ilgili metoda giderek gerekli değişiklikleri yapabilmeliyiz.**

metin, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

## ****Open/Closed Principle****

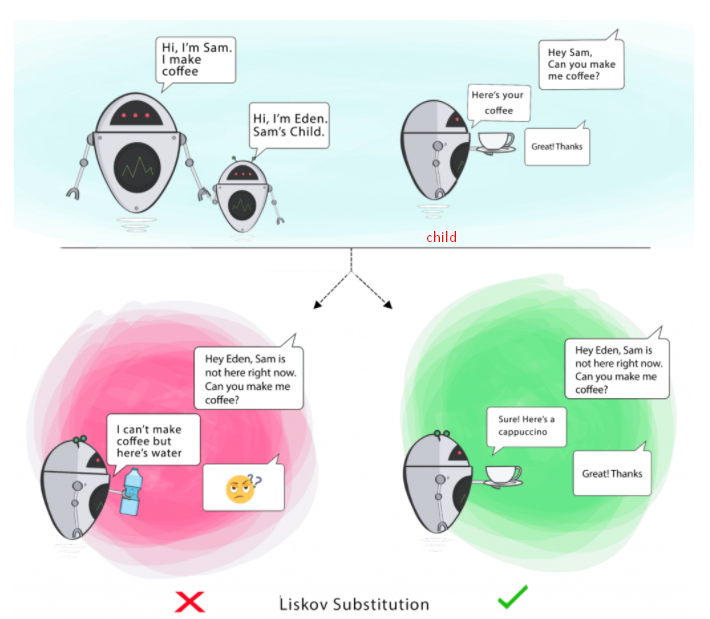
Türkçe çevirisi “Açık/Kapalı” olan prensip, projede geliştirilen nesnelerin geliştirilmeye açık ama değişime kapalı olmaları gerektiğini ifade eder. Yani bir nesne davranışını değiştirmeden yeni özellikler kazabiliyor olmalıdır. Bu prensip, sürdürülebilir ve tekrar kullanılabilir yapıda kod yazmanın temelini oluşturur.

iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

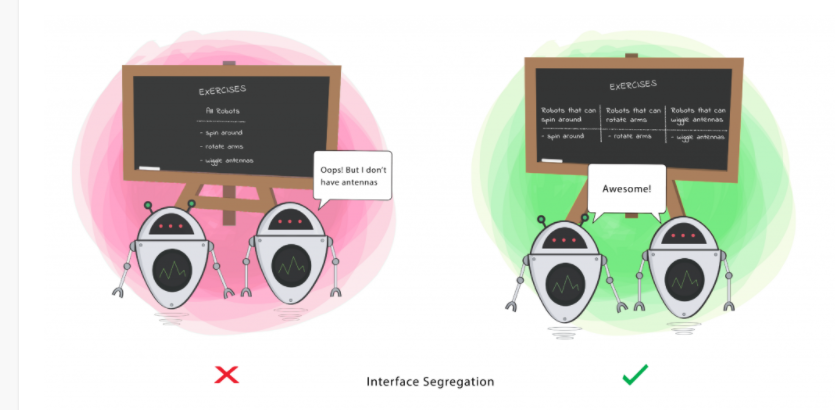
## ****Liskov Substitution Principle****

**Yerine Geçme”** olarak Türkçeye çevirdiğimiz prensibe göre; miras alarak türemiş olan class’ların önce miras aldıkları nesnenin tüm özelliklerini kullanması, daha sonra da kendi özelliklerini barındırması gerekir. Eğer oluşturduğumuz class, miras aldığı nesnenin ‘tüm’ özelliklerini kullanmayacaksa ortaya gereksiz kod blokları çıkar ve bu da bir geliştiricinin isteyeceği en son şeydir. Çünkü bir geliştirici her daim ‘Clean Code’ yazmaya çalışır.



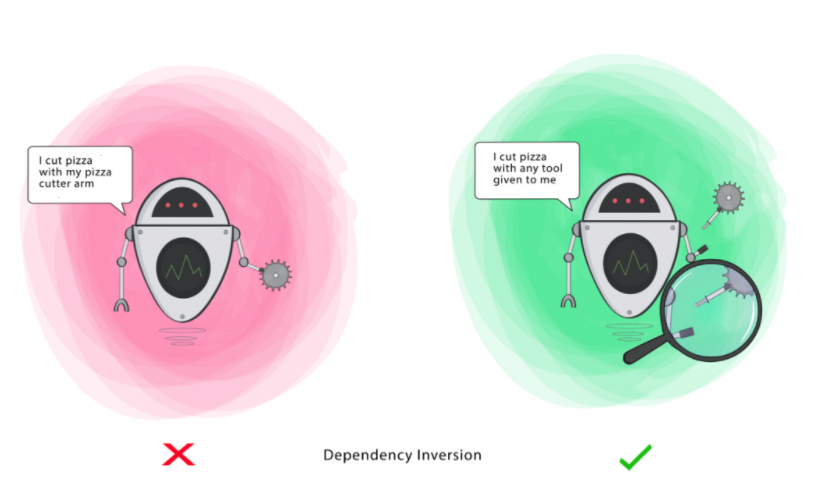
## ****Interface Segregation Principle****

“Arayüz Ayırımı” prensibinde; bir interface’e gerekenden fazla sorumluluk eklemek yerine, daha özelleştirilmiş birden fazla interface oluşturulmalıdır. Nesneler, ihtiyacı olmayan özellik veya metotlar içeren interface’leri miras almaya zorlanmamalıdır. **Sizinde farkettiğiniz üzere “Single Responsibility” ve “Interface Segregation” prensipleri birbirine oldukça benzemekte ve aynı amaca hizmet etmektedirler. Ancak burada gözden kaçırılmaması gereken en önemli husus şudur ki; ‘Interface Segregation’ prensibi interface’ler ile ilgilenirken, ‘Single Responsibility’ prensibi class’lar ile ilgilenmektedir.**



## ****Dependency Inversion Principle****

Türkçe karşılığı “Bağımlılığın Ters Çevrilmesi” olan bu prensibe göre; alt sınıflarda yapılan değişiklikler üst sınıfları etkilememelidir yani sınıflar arası bağımlılıklar olabildiğince az olmalıdır ve özellikle üst seviye sınıflar, alt seviye sınıflara bağımlı olmamalıdır. Peki burada ne yapmalıyız? Burada yüksek seviye sınıf ile düşük seviye sınıf arasında bir soyutlama katmanı oluşturarak her iki sınıfı da soyut kavramlar üzerinden yönetmeliyiz.



# Design Patterns (Tasarım Kalıpları)

## Singleton Pattern

Yapacağımız uygulamada **bir sınıfımızdan sadece bir nesne oluşturulmasını sağlamak istediğimizde** uygulayacağımız Design Pattern, Singleton Pattern’dir. Uygulamamızda oluşturacağımız bu nesne Singleton Pattern sayesinde ikinci bir nesnenin oluşturulmasını engelleyecektir.

Örn : **Bazı nesneler vardır ki her bir istek için ayrı ayrı oluşturmaya gerek yoktur.Örneğin db connection gibi.Her istemci için ayrı ayrı bu nesneyi oluşturmak performans kaybına sebep olacaktır**.Singleton ile oluşturulan nesneler tek bir örnek üzerinde kalır ve her istemci bu örneği kullanır.

**Singleton tasarım kalıbını bu şekilde her nesne için uygulayamayız.** Şöyle ki projemizde her kullanıcı için farklı işlemler gerçekleştiren bir class için singleton uygularsak her bir kullanıcı için aynı verileri kullanmış oluruz. Örneğin her kullanıcının kullanıcı adı aynı olur gibi.

**Nerelerde kullanabiliriz ? :**Gerçek hayatta proje geliştirirken config, connection gibi her istemci için yeni örnek oluşturulan yerlerde ve yardımcı fonksiyonları çağırdığımız kısımlarda singleton tasarım kalıbını rahatlıkta kullanabiliriz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

# KISALTMALAR

# AJAX: [Asynchronous JavaScript And XML](https://en.wikipedia.org/wiki/Ajax_(programming)).

# API: [Application programming interface](https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/).

# CRUD: [Create, read, update, delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete).

# CSS: Abbreviation of [Cascading Style Sheets](https://www.w3.org/Style/CSS/Overview.en.html)

# DOM: [Document Object Model](https://en.wikipedia.org/wiki/Document_Object_Model#:~:text=The%20Document%20Object%20Model%20(DOM,document%20with%20a%20logical%20tree.).

# EOF: [End of File](https://en.wikipedia.org/wiki/EOF).

# FTP: [File Transfer Protocol](https://www.digitaltrends.com/computing/what-is-ftp-and-how-do-i-use-it/).

# GUI: [Graphic User Interface](https://www.computerhope.com/jargon/g/gui.htm#:~:text=A%20GUI%20(graphical%20user%20interface)%20is%20a%20system%20of%20interactive,the%20user%20interacts%20with%20them.).

# HTML: [HyperText Markup Language](https://en.wikipedia.org/wiki/HTML#:~:text=Hypertext%20Markup%20Language%20(HTML)%20is,scripting%20languages%20such%20as%20JavaScript.).

# IDE: [Integrated Development Environment](https://en.wikipedia.org/wiki/Integrated_development_environment#:~:text=An%20integrated%20development%20environment%20(IDE,automation%20tools%20and%20a%20debugger.)

# JDBC: Abbreviation of [Java Database Connectivity](https://en.wikipedia.org/wiki/Java_Database_Connectivity#:~:text=Java%20Database%20Connectivity%20(JDBC)%20is,Edition%20platform%2C%20from%20Oracle%20Corporation.).

# JDK: [Java Development Kit](https://en.wikipedia.org/wiki/Java_Development_Kit).

# JRE: Abbreviation of [Java Runtime Environment](https://www.infoworld.com/article/3304858/what-is-the-jre-introduction-to-the-java-runtime-environment.html).

# JSON: [JavaScript Object Notation](https://www.json.org/json-en.html).

# JVM: Abbreviation of [Java Virtual Machine](https://www.guru99.com/java-virtual-machine-jvm.html#:~:text=Java%20Virtual%20Machine%20(JVM)%20is,code%20for%20a%20particular%20system.).

# MVC : Model, View Controller.

# REGEX: [Regular expression](https://regexr.com/).

# REST: Abbreviation of [Representational state transfer](https://en.wikipedia.org/wiki/Representational_state_transfer).

# SDK: [Software development kit](https://en.wikipedia.org/wiki/Software_development_kit#:~:text=A%20software%20development%20kit%20(SDK,and%20perhaps%20a%20software%20framework.).

# SOAP: Abbreviation of [Simple Object Access Protocol](https://en.wikipedia.org/wiki/SOAP).

# SOAP: Simple Object Access Protocol.

# SQL: Abbreviation of [Structured Query Language](https://www.webopedia.com/TERM/S/SQL.html).

# XML: [eXtensible Markup Language](https://en.wikipedia.org/wiki/XML#:~:text=Extensible%20Markup%20Language%20(XML)%20is,free%20open%20standards%E2%80%94define%20XML.).