PATIKA-INNOVA

Homework1

Hatice ÖZTÜRK

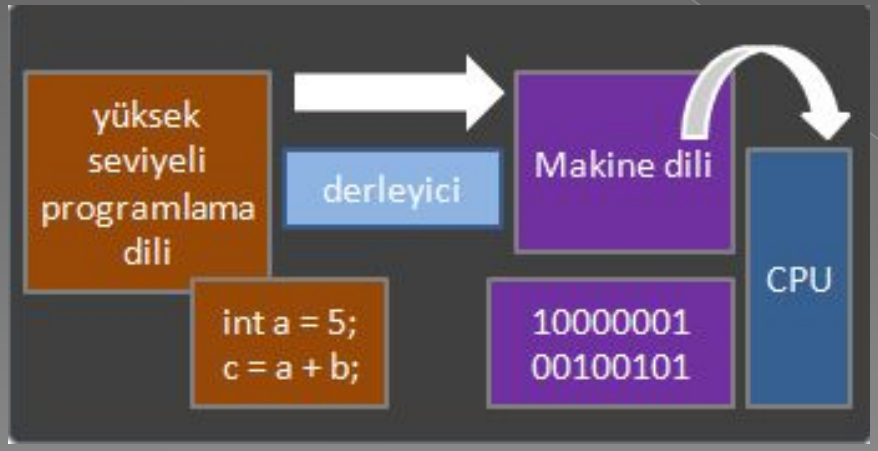
08.01.2022

HOMEWORK 1

- Compiler Nedir, Interpreter Nedir, Aralarındaki Fark Nedir?
- Java pass by value mi pass by refrerence mi?
- JDK, JRE,JVM,JIT Nedir?
- Primitive type ve Wrapper classlar arasındaki farklar nelerdir?
- Stack ve Heap hafıza nedir?
- Serileştirme Nedir?

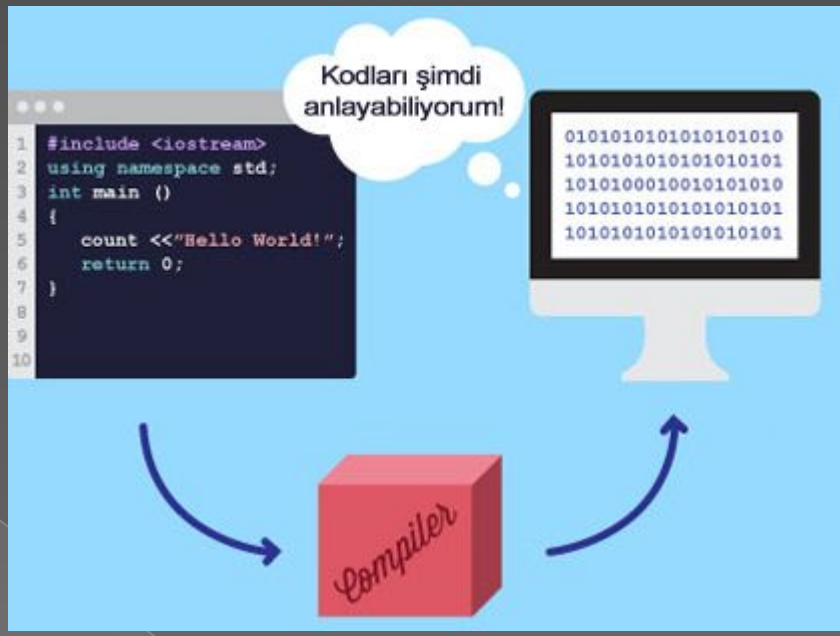
Compiler (Derleyici) Nedir?

- Programlama dilinde yazılmış bir kaynak kodu başka bir dile veya genellikle makine diline çeviren programdır.
- Derleyici programlar yaygın olarak executable code olarak tanımlanan hemen çalıştırılabilir kodlar üretmektedir.
- Bir derleyici kodun yazıldığı dil ile aynı seviyedeki başka bir dile çevirmekle beraber aynı zamanda üst seviye olan bir dilin kodunu daha alt seviyeli bir programlama diline de çevirir.



Compiler Nedir?

- Eğer Compiler adını verdiğimiz derleyiciler olmasaydı, geliştiricilerin yazılımlarının tümünü makine dilinde hazırlaması gerekecekti. Geliştiricilerin hayatını kolaylaştıran Compiler sayesinde geliştiriciler farklı programlama dillerini kullansalar bile bilgisayarların anlayabileceği dile dönüştürülen yazılımları kolayca oluşturabilirler.
- Java derleyicisi **javac** 'dır.
  Javac, *.java* uzantılı kaynak dosyasını Java Sanal Makinesi (Java Virtual Machine) olarak bilinen bir hayali makine için makine dili olan ***Java bytecode*** ile yazılmış *.class* dosyasına dönüştürür.



Compile Amacı Nedir?

- Programı bilgisayar tarafından anlaşılıp çalıştırılabilecek hale getirmek.

- Programdaki sözdizimi (syntax) hatalarını bulmak. Derleyici, kaynak kodu derlemeden önce kodları kontrol eder. Herhangi bir yazım hatası veya derleyicinin sevmediği durumlar var ise bu raporlanır. Bu da tam olarak çalışan program elde etmeden önce tüm kodlama hatalarını düzeltmemizi sağlar.

Interpreter (Yorumlayıcı) Nedir?

- Girdi olarak program için olan verilerle birlikte kaynak kodu alan, ve kaynak programı satır satır yürüten bir programdır.

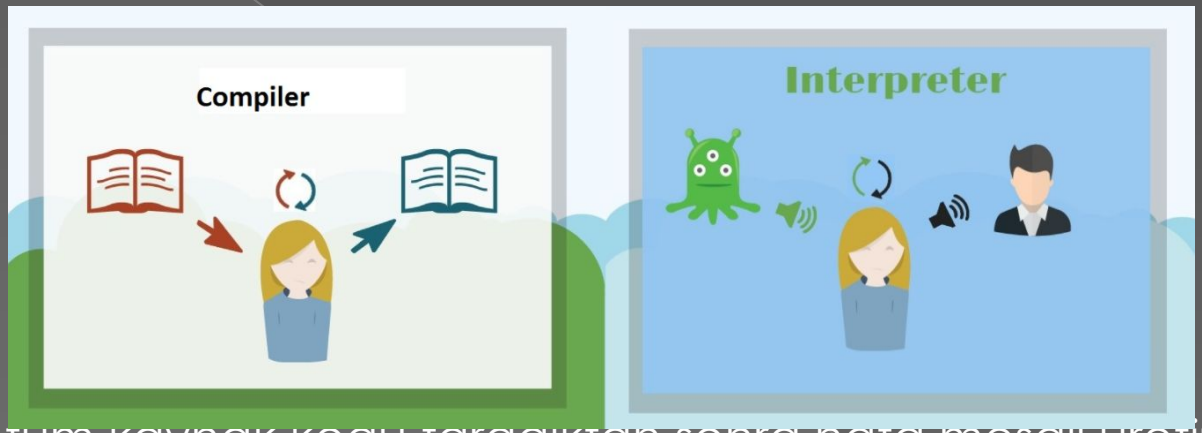
- Yorumlayıcılar (Interpreter), kodu satır ve/veya bloklar halinde çalıştırırlar ve bir sonraki satır / blokları sırası gelmeden değerlendirmezler. Bu nedenle, sonraki satırdaki hatalar ve kodun bütününü etkileyen iyileştirmeler yorumlayıcılar tarafından yakalanamazlar. Kodu parçalar halinde değerlendirmek amacıyla kullanılırlar.

- Java yorumlayıcısı , *.class* uzantılı dosyayı üzerinde çalıştığı makinede çalıştırılabilecek olan doğal makine kodlarına çevirir.

- Java'da derleyici ve yorumlayıcı beraber çalışır. Yani, önce oluşturulan kaynak koddan bir ara kod (bytecode) üretilmek için derlenir. Daha sonra bu derlenen bytecode Java Sanal Makinesi (JVM) üzerinde yorumlanarak yürütülür.

Compiler ve Interpreter Arasındaki farklar

- Derleyici, tüm programı tarar ve bir bütün olarak makine koduna çevirir, yorumlayıcı ise programı satır satır işler.
- Derleyici, kaynak kodun analizi için büyük zaman harcar. Ancak genel yürütme süresi daha hızlıdır. Yorumlayıcı ise, kaynak kodu analiz etmekle zaman harcamaz. Ancak genel yürütme süresi daha yavaştır.

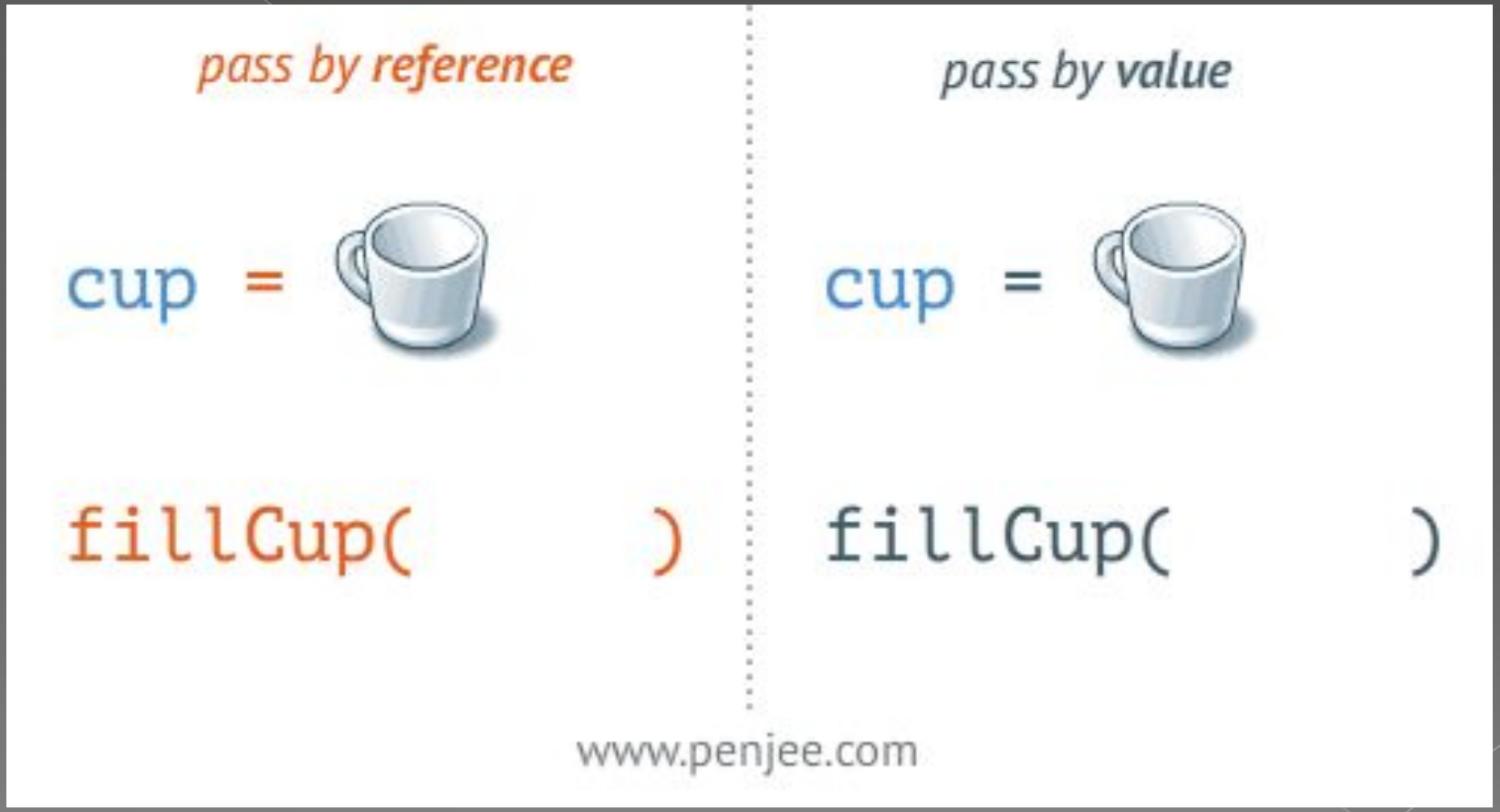


- Derleyici, tüm kaynak kodu taradıktan sonra hata mesajı üretir. Bu nedenle hata ayıklama nispeten zordur. Yorumlayıcı ise herhangi bir hata olana kadar programı çalıştırır. İlk hata gördüğü yerde durur. Bu nedenle hata ayıklama kolaydır.
- C, C++ gibi diller derleyici kullanır. Python, Ruby gibi diller yorumlayıcı kullanır. Java ise; her ikisini de kullanır.

Java pass by value mi pass by refrerence mi?

- Programlama dilleri metotlara parametre aktarılırken 2 farklı yaklaşım kullanır. **Pass by value(değere göre geçirme)** ve **pass by reference(referansa göre geçirme)** .
- Bu yaklaşımlar, değişkenlerin metotlara nasıl aktarıldığını tanımlamak için kullanılan 2 farklı tekniktir. Kısaca izah edecek olursak, **değere göre geçişte**, metoda gerçek değerin geçirildiği anlamına gelir. Referansla geçişte, değerin nerede saklandığını tanımlayan bir işaretçinin (bu, geçirilen değişkenin hafızadaki adresi olarak düşünülebilir) geçirildiği anlamına gelir.

Pass by Value vs Pass by Reference



Java pass by value mi pass by refrence mi?

- *Java'da **HER ZAMAN** pass by value yaklaşımı uygulanır.*
- Java üst düzey bir programlama dilidir. Bu, normal şartlar altında, bellekte ne olduğu konusunda endişelenmeniz gerekmediği anlamına gelir. Çünkü java hafıza yönetimini arka planda kendi halleder.
- Java'da da ilkel veri tipleri (int, double vb.) her zaman **değere göre iletilir**, yani bütün işlem aslında metoda geçirilen değişkenin değerin bir kopyası üzerinden gerçekleşir.

JDK, JRE, JVM, JIT Nedir?

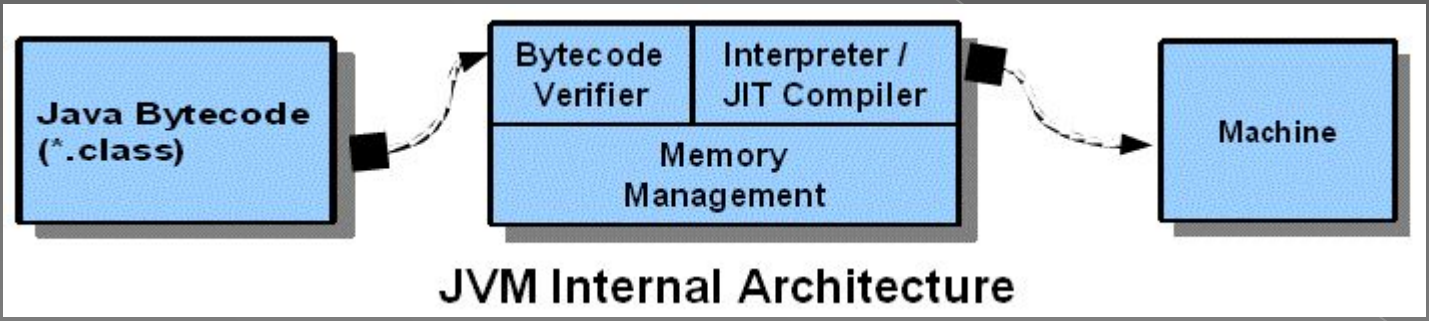
- Bytecode olarak derlenen Java sınıfları Java sanal makinesi (Java Virtual Machine – JVM) bünyesinde yorumlanır. Tek derleme işlemi Java sınıflarının bytekoda dönüştürülmesi esnasında yapılmaz. JVM bünyesinde de bytekodun makine koduna dönüştürüldüğü bir derleme gerçekleştirilir. Bu işleme Just in time (JIT) compilation ismi verilmektedir.
- JIT(Just In Time): Girdi olarak sadece kod değil aynı zamanda kullanıcı alışkanlığı, kullanım sıklığı gibi başka parametreleri de alan derleyici tipidir.
- Java JIT compiler çalışma anında bytecode bloklarını native koda dönüştürür. Yani direkt platform bağımlı bir hale gelir. Bunu yapmasındaki amaç, çok kullanılan veya çağrılan blokların performansını arttırmaktır.

JIT compiler kodu olduğu gibi native kod yapısına çevirmemektedir. Belli bir aşamadan ve iyileştirmelerden geçerek nihai native kod oluşur. Bunlardan bazılarını listelemeye çalıştım.

- Dead Code: Kullanılmayan metotlar değişkenler JIT compiler tarafından elenir. Böylece boşuna yük alınmamış olunur.
- Inlining: Her ne kadar kodumuz küçük bloklardan oluşsa da native kod çalıştırılırken bunlar devamlı bir jump anlamına gelir ve maliyetli işlemlerdir. Bundan dolayı JIT compiler bazı blokları iç içe gömerek bu jump adımlarını minimum sayıya indirmeye çalışır.
- Local Optimization: Blok ve metot içerisindeki küçük parçalarda iyileştirmeler yapılır.
- Control Flow Optimization: Döngü, if, switch gibi akış ve kontrollerde iyileştirmeler yapılır.
- Global Optimization: Metotun tamamı dikkate alınarak iyileştirmeler yapılır.
- Native Code Generation: Son adım olarak native kod oluşur.

JVM(Java Virtual Machine)

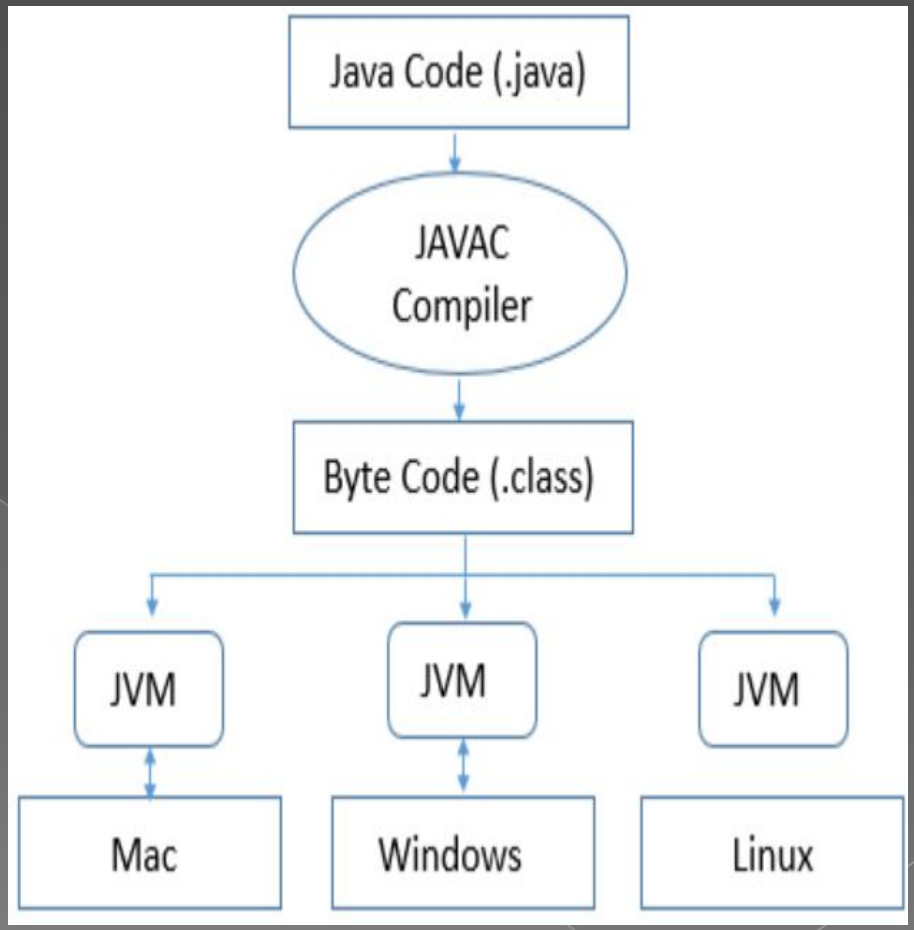
- JVM(Java Virtual Machine) için java programının çalıştığı platform ile java programı arasında soyut bir ara katman diyebiliriz.
- JVM; platforma bağımlı olarak çalışır.
- Java, bir sanal makine üzerinde çalışan yapıya sahiptir. Bu yüzden Java'da yazılan uygulamaları çalıştırabilmek için bilgisayarımıza bir Java sanal makinesi kurmamız gerekiyor. JVM(Java Virtual Machine) işte bu işe yaramaktadır.
- Her sistem için aynı olan bu bytecode ları alıp çalıştığı sisteme özgü bir şekilde yorumlamaktadır.



JVM Internal Architecture

JVM-devam

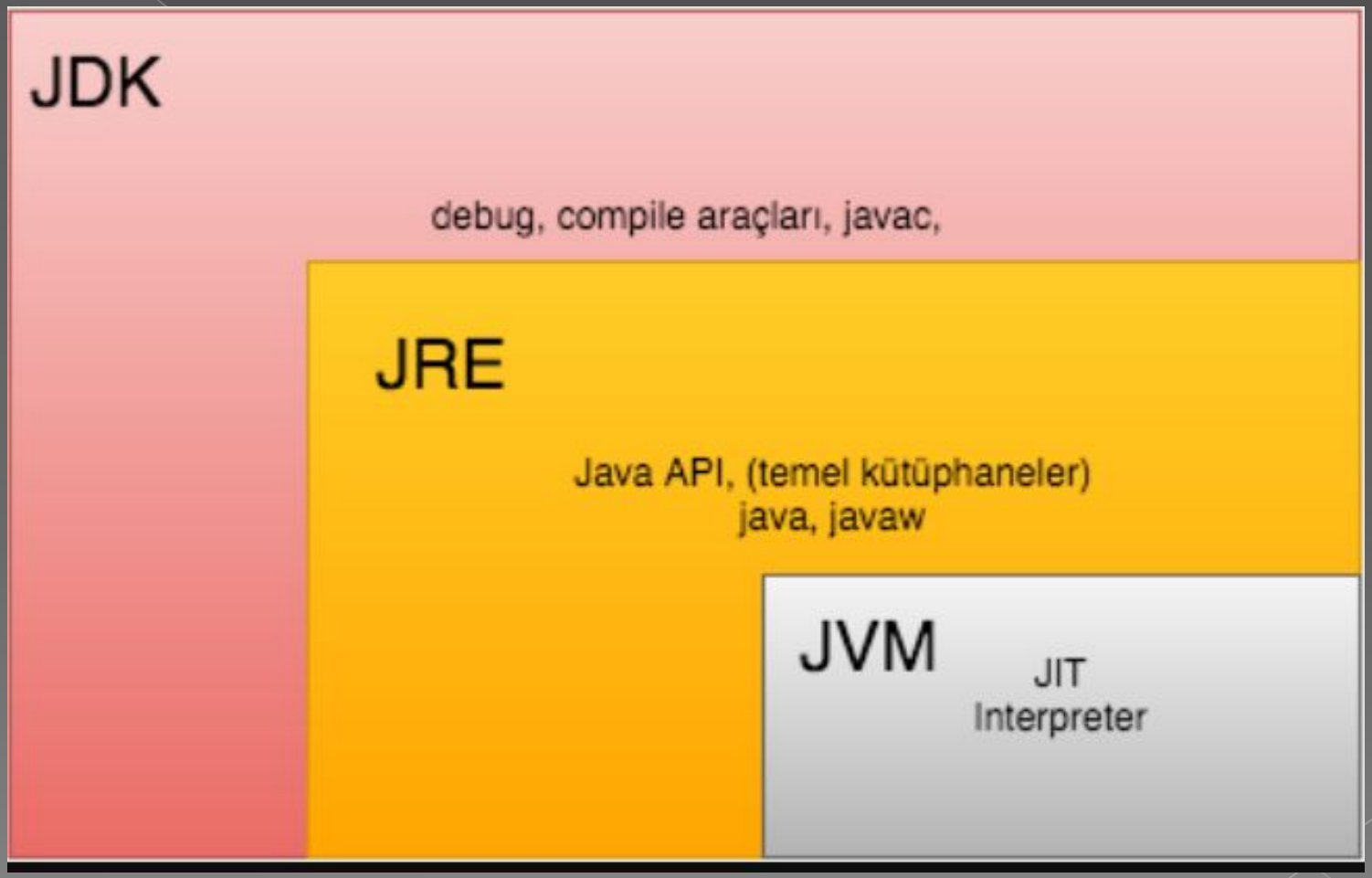
- JVM; bizim yazdığımız .java uzantılı dosyaları anlamaz onun yerine derlenmiş .class uzantılı dosyaları anlar. Çünkü .class uzantılı dosyalar içlerinde bytecode lar içerirler. Bu özelik sayesinde Java da "Write once,Run everywhere" özeliğini kullanabiliyoruz. Yani bu şu demek oluyor; bizim windows bir makinede yazmış olduğumuz uygulama önce Compiler tarafından bytecode lara çevriliyor daha sonra bu bytecode lar diğer platformlarda kurulu olan JVM ler aracılığıyla tüm platformlarda çalışıyor.



JRE(Java Runtime Environment)

- Bir Java uygulamasını çalıştırmak için gerekli minimum gereçleri içeren yapıya verilen isimdir. Yani bilgisayarımızda bir Java uygulamasının çalışabilmesi için JRE'nin kurulu olması gerekir. JRE içerisinde JVM de bulunur.
- JRE, java programlama dili ile yazılmış olan uygulama ve appletlerin çalışmasını sağlayan bileşenler ile JVM e kütüphaneler sağlar.Derlenmiş byte codelar direk olarak CPU üzerinde çalışmazlar. CPU tarafından anlaşılması için aradaki JVM bytecode ları okunabilir makine kodları olarak yorumlar. Aslında; java bytecode ların bütün platformalarda çalışması JRE sayesindedir. İçerisinde; JVM, Core kitaplıkları ve Java yazılımında yazılan uygulamaları ve küçük uygulamaları çalıştırmak için diğer ek bileşenleri içerir. JRE'nin görevi Java kodları derlendikten sonra bir ara dil olarak kabul edilen Java bayt kodlarını oluşturmaktır. Bu bayt kodlar bütün işletim sistemleri için aynıdır.

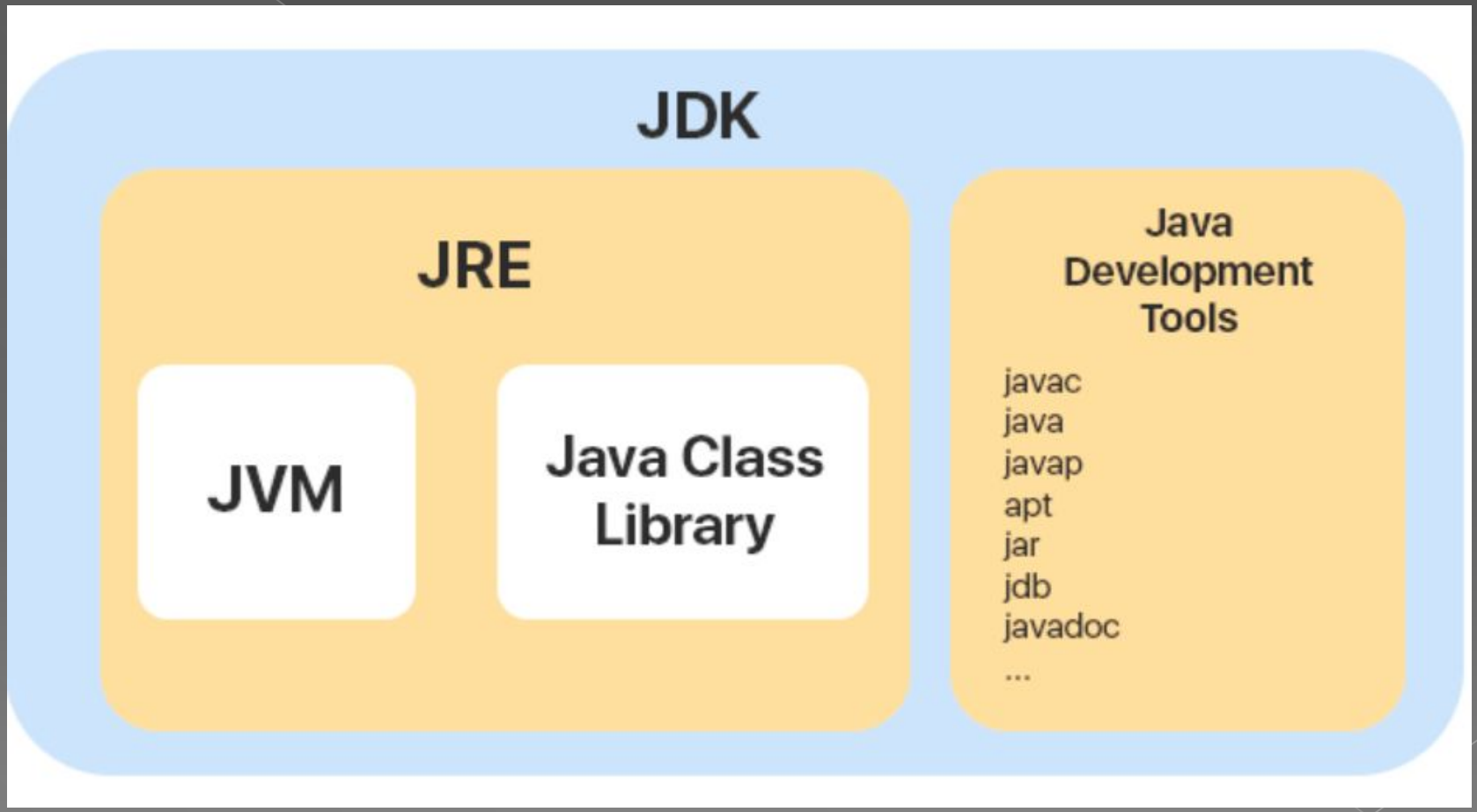
JRE(Java Runtime Environment)-devam



JDK (Java Development Kit)

- JDK dediğimiz yapı içerisinde Java Compiler, Java Interpreter, geliştirici toolları, Java API kütüphaneleri, Java geliştiricileri tarafından Java uygulamaları geliştirmek için kullanılan dokümantasyonlar bulunur. Ayrıca JVM ve JRE içerir.
- JDK = JRE + JVM + derleyici + diğerleri diyebiliriz.
- Java ile geliştirme (development) yapmak için Java Development Kit'e (JDK) ihtiyacınız var. Bu paket Java ile geliştirme yapmak için bütün araçları içeriyor. Bu tool'lara editör (düzenleyici) dahil değildir.
- Ayrıca oldukça güçlü bir makinede bile oldukça ağır çalışırlar. Programın nasıl çalıştığına zaman harcayıp öğrenmemiz gerekir ; üstelik, en önemlisi, kodun içine kendiliğinden bir sürü kod eklerler, bu da yazdığınız kodu anlamamanıza yol açar.

JDK-devam


JDK
JRE
JVM
Java Class Library
Java Development Tools
javac
java
javap
apt
jar
jdb
javadoc
...

Primitive Type ile Wrapper Class Arasındaki Farklar

- **Primitive Type:** Primitive değişkenler hafızadaki boyutları belli olan değişkenlerdir. Null değer alamazlar java başlangıçta değer atanmadı ise bunlara otomatik sıfır değeri atar.
- Java dili tarafından sağlanan sekiz ilkel tür vardır. Bunlar;
- 1-int 5- char
- 2-double 6- byte
- 3-float 7- short
- 4-long 8- boolean

Veri Tipi	Alan Büyüklüğü	Kategori
byte	8 bit	Tamsayı Tipleri
short	16 bit	
int	32 bit	
long	64 bit	
float	32 bit	Kesirli Sayı Tipleri
double	64 bit	
char	16 bit	Karakter Tipi
boolean	-	Mantıksal Tip (ture/false)

Wrapper Class

- Java'da bir Wrapper sınıfı, ilkel bir veri türünü bir nesneye ve nesneyi ilkel bir türe dönüştürmek için kullanılır. İlkel veri türleri bile birincil veri türlerini depolamak için kullanılır.
- Wrapper sınıfı olan rerefans türü bir object'tir ve null değer alabilir.
- Primitive tiplere karşılık gelen wrapper classlar :

  1-Integer
  2-Double
  3-Float
  4-Long
  5-Character
  6-Byte
  7-Short
  8-Boolean

Veri	Uzunluk	Gömüldüğü Sınıf
char	16-bit	Character
Byte	8-bit	Byte
Short	16-bit	Short
İnt	32-bit	Integer
Long	64-bit	Long
Float	32-bit	Float
Double	64-bit	Double
boolean	1-bit	Boolean

Aralarındaki Farklar

Java'da Sarıcı Sınıfı ve İlkel Tür	
Wrapper sınıfı, ilkel türü nesneye ve nesneyi ilkel türe dönüştürmek için bir mekanizma sağlar.	İlkel 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.
Boş Değerler	
Sarıcı sınıf nesneleri boş değerlere izin verir.	İlkel bir veri türü boş değerlere izin vermez.
Gerekli Bellek	
Gerekli bellek, ilkel türlerden daha yüksektir. Kümelenmiş Dizin, ek bir alan gerektirmez.	Gerekli bellek, sarmalayıcı sınıflara kıyasla daha düşüktür.
Koleksiyonlar	
Bir Wrapper sınıfı, ArrayList vb. gibi bir koleksiyonla kullanılabilir.	Koleksiyonlarda ilkel bir tür kullanılmaz.

Stack Memory, Heap Memory Nedir?

- **Stack:**Stack, öğeleri aslında bir yığın şeklinde yöneten ve erişimi kolay bir bellektir.
- Thread run edildiğinde oluşturulan bir memorydir. Genellikle boyutu önceden bilinen öğeleri kaydeder.
- Primitive tipler(byte, short, int, long, float, double, boolean, char) stack üzerinde tutulur.
- Stack'e boyutun kaldırabileceğinden fazla veri koymaya çalışırsak **stackoverflow** hatasıyla karşılaşırız.
- Stackte öğelerin eklenmesi ve çıkarılması her zaman stackin üst tarafında gerçekleşir. Yani LIFO mantığı ile çalışır, en son kullanılan blok her zaman serbest bırakılacak ilk bloktur.

Stack'de yapılan bazı işlemler:

- Push => Stack'in en tepesine bilgi ekleme işlemidir.
- Pop =>Stack'in en tepesinden bilgi alınır.
- Top => Stack'in en tepesindeki bilgiyi okur ama stackten çıkartmaz.

you can imagine stack like this

Heap Memory

- Heap memory, bilgisayar belleğinin bizim için otomatik olarak yönetilmeyen bölgesidir. Çalışma zamanında tam olarak ne kadar veriye ihtiyacımız olacağını bilmiyorsak veya çok fazla veri ayırmamız gerekiyorsa heap kullanırız.
- Referance tipler ve dinamik olarak oluşturulan değişkenler burada depolanır, bu durum kullanımdan sonra ayrılan belleğin temizlenmesini gerektirir.(Java , C# gibi dillerde Garbage Collection bu işi yapıyor.)

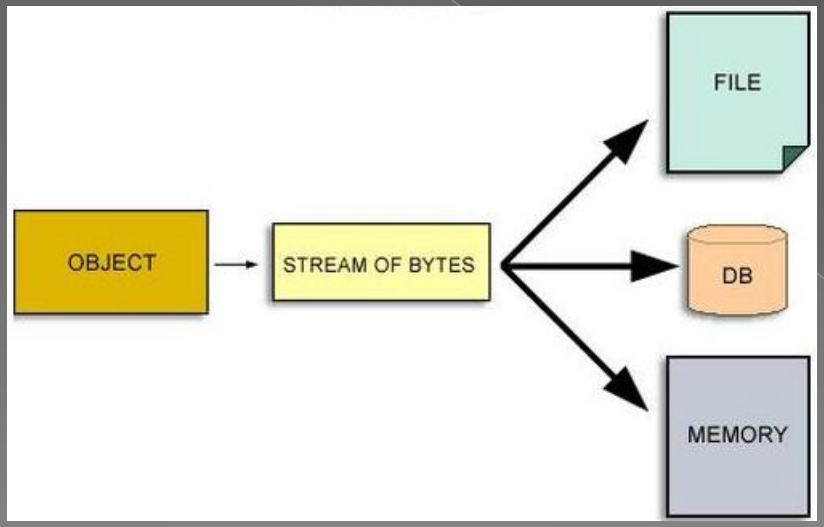
heap should be like this :)

Stack ve Heap Karşılaştırması

- Hem stack hem de heap RAM'de tutulur.
- Stack, heap'e göre daha hızlıdır.
- Stack yapısı lineerdir, heap yapısı hiyerarşiktir.
- Stack'in implementasyonu kolayken, heapde bu durum daha zordur.
- Allocation ve de-allocation stackde compiler tarafından otomatik yapılırken, heapde programcı tarafından manuel olarak yapılır.
- Stack, thread-safe'dir. Heap ise thread-safe değildir. Depolanan veriye farklı threadlardan erişim olabilir.
- Stackde memory contiguous(bitişik) bloklardan allocate edilirken, heapde rastgele bloklardan allocate edilir. Bu durum stack'in heap'e göre daha basit olmasını sağlar.
- Stack'in boyutu sabit iken, heap boyutu esnektir ve boyutunu değiştirmek mümkündür.

Serileştirme

- Java'da, tasarlanan nesnelerin tekrar kullanılabilmesi (reuse) önemli bir konu olduğuna göre, bu nesneleri Java platformu dışında da hayata geçirmek gerçekten önemlidir. Bahsedilen bu problem, Java Serialization API sayesinde çok kolay bir şekilde aşılabiliyor.
- Java Serialization API sayesinde bir nesnenin birebir kopyasını, Java platformu dışında da depolayabiliriz. Bu mekanizma ile daha sonra, nesneyi depolanan yerden çekip, aynı durum (state) ve özellikleri ile kullanmaya devam edebiliriz. Tüm bu sisteme, **Object Serialization** (Nesne Serileştirme) adı verilir.



Kaynakça:

- https://kplnosmn94.medium.com/jvm-jre-ve-jdk-nedir-6cfee2727812
- https://lafayettefirefighters.com/tr/difference-between-wrapper-class-and-vs-primitive-type-in-java
- https://medium.com/@haticeozturk/stack-ve-heap-51f427f00c81
- http://volkanozturk.net/java-serialization-serilestirme-nedir/