3. Java'nın platform bağımsızlığını nasıl sağladığını anlatınız.

- Java hem derlenen hem de yorumlanan bir dildir. Yazmış olduğumuz Java kodları öncelikle Java compiler üzerinden derlendikten sonra çıktı olarak byte code oluşturur. Daha sonra JVM (Java Virtual Machine) sayesinde bu kodlar yorumlanarak çalışır halde getirilir. JVM'in kullanılması için çalıştığı sistem üzerinde kurulu olması gerekmektedir. Bu nedenle günümüzde neredeyse her platform için yazılmış bir JVM bulunmaktadır. Bu özellik sayesinde Java dilinde yazdığımız kodları hem platformdan bağımsız şelikde kullanabilmekteyiz. Bu sebeplerden dolayı Java'nın platform bağımsızlığını "write once run everywhere"(bir kere yaz her yerde çalıştır) cümlesi yeterince anlatmaktadır.

4. Java'da heap ve stack kavramlarını örneklerle açıklayın.

- Stack ve Heap bellekte bulunan mantıksal yapılardır. Heap, JVM tarafından nesne oluşturmak için kullanılan bellek alanıdır. Class type değişkenler referans tiplerdir referans ettikleri model (referans) stack de değerleri ise heapte saklanır. Genelde stack kısmından daha büyüktür.
- Primitif tipli yapılar int, byte, long, vs. gibi değer tipli yapılar stack de tutulur. Stack, programda aktif olarak çalışan metotlar için ayrılan bellek alanlarının olduğu yerdir. Java'da stackte önce main metodunun yeri açılır. Sonrasında Main'den devam ederek, kullanılan metotlar için yeni yerler açarak devam eder. Metotların çalışmaları bitince kendiliğinden silinir. Heapte ise verilerin silinmesi Garbage Collector tarafından yapılır. Stack kısmının büyüklüğü metotlarda tanımlamış olduğumuz değişkenlere göre değişir. Ayrıca normal değişkenleri tanımlarken bunlar belleğin stack bölümünde tutulur ancak bu değişkenleri nesne üreterek tanımladığımızda ise heapte tutulur.

5. String class'ı nasıl immutable olmayı sağlamaktadır örnek ve çizimlerle açıklayınız.

- Java'da 4. soruda da açıkladığım üzere primitif tipli yapılar bellekte stack kısmında tutulur ve bu oluşturduğumuz lokal değişkenleri ihtiyaç duymamız halinde tanımlanmış oldukları metodların içerisinde değiştirebilmekteyiz. İki değişken üzerinden basit bir örnek vermek gerekir ise;

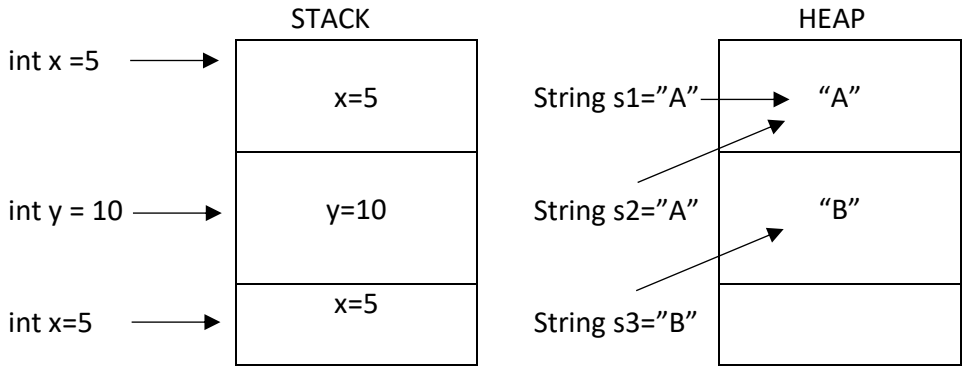
```
package deneme;
class Main {
        public static void main(String[]args) {
                int x = 10;
                int y = 20;
                x = y;
                y=35;
                System.out.println(x);}}
```

İşlemin sonucunda x= 20 değerini almayı bekleriz.

Ancak referans tipli yapılara geldiğimizde iş değişmekte çünkü referans tipli heapte tutulur ve bunlar değerleri üzerinden değil de heap üzerinde gösterdikleri adrese göre tutulurlar. String classı da böyle bir yapı olduğundan immutable olmayı sağlar. Yine yukarıdaki örneğin bir benzerini String classı ile ilgili olarak da vermek gerekirse;

```
package deneme;
class Main {
        public static void main(String[] args) {
                String x = "Mahmut";
                String y = "Ali";
                x=y;
                y="Kaya";
                System.out.println(x);}}
```

bu işlemin sonucunda eğer String sınıfı primitif tipli bir yapı olup belleğin stack bölümünde tutulmuş olsa idi "Kaya" sonucunu almayı beklerdik ancak String sınıfının referans tipli bir yapı olması bu sınıfa immutable olmayı sağlamaktadır.



6. Java neden çoklu kalıtımı desteklemez açıklayın?

Bunu bir örnekle açıklamak gerekirse A,B ve C olarak üç tane sınıfımızın olduğunu ve A sınıfının B ve C sınıflarını miras aldıklarını varsayalım. Bu durumda B ve C sınıflarından nesne çağırdığımızda ve aynı yöntem veya yöntemlere sahiplerse programımızı çalıştırdığımızda derleme zamanı hatası alırız. Bu gibi durumların önüne geçmek ve karmaşıklığı azaltmak için Java çoklu kalıtımı desteklemez. Ancak çoklu kalıtımın bir benzerini Java’da interfaceler ile kullanabilmekteyiz.

7. Build Tool nedir? Java ekosistemindeki build toollar neler açıklayın?

Build toollar geliştirmeye çalıştığımız projenin yönetimini sağlayarak, bağımlılıkları en aza indirmeye çalışan programlardır. Çeşitli insanların üzerinde çalıştığı projelerde yapılan değişikliklerin programda birşeyleri bozmaması için kullanılır. Java ekosistemindeki build toollar; Maven, Gradle, Apache Ant, Apache Ivy ve sbt dir.

- **Maven :** Özetle proje bağımlılık yönetim aracı olduğunuz söyleyebiliriz. Projeyi geliştirirken bir standart oluşturmaya, geliştirme kısmını basitleştirmeye ve bağımlılıklarını ortadan kaldırmaya yarayan bir araçtır.
- **Gradle :** Android Studio üzerinde kurulu olarak kullanılan ve android uygulaması geliştirme aşamalarını otomatize eden bir build tooldur.
- **Apache Ant :** Java kodunun taşınabilirliği ile Java uygulamalarını oluşturmak için kullanılan bir komut satırı aracıdır. Dokümantasyon oluşturma vb. gibi işleri otomatize eden bir build toldur.
- **Apache Ivy :** Yukarıda belirtmiş olduğum Apache Ant projesinin bir alt projesidir. Proje bağımlılıklarını tanımlamak ve proje oluşturmak için gerekli kaynakları isteyen bir paket yöneticisidir.
- **sbt :** Scala ve Java projeleri için açık kaynaklı bir oluşturma aracıdır. Bağımlılıkları ve eklentileri yönetmemize yardımcı olan bir tooldur. Ayrıca kimi kaynaklarda ismi Scala Build Tool olarak geçse de sitelerinde sadece sbt olarak tanımlıyorlar toolun ismini.

8. Collection framework içerisindeki bütün yapıları önemli methodlarıyla örnekleyip açıklayınız.

Collection : Collection interfacelerinin hiyerarşik olarak en tepesinde bulunan, en genel collection interface sınıfıdır.
List: List yapısını örnek alan collection tipidir. İçerisinde bir elemandan birden fazla bulundurmaya izin verir. Elemanları sıralı olarak tutar. Index değerleri ile işlem yapılır.
Set : içerisinde sakladığı elemanlar sıralı olarak tutulmaz. Ayrıca bu collection tipi içerisinde kopya eleman bulundurmaz. Yani her elemandan sadece bir tane bulunabilir.
SortedSet : verileri sıralı bir biçimde tutar. Set tipi gibi kopya eleman bulundurmaya izin vermez.
Queue : kuyruk yapısını benimseyen collection tipidir. List tipinden oluşturulmasına karşın List tipinin özelliği olan index değerleriyle işlem yapmamasıyla liste türünden ayrılır.
Map: elemanlarla, bunlara karşılık verilen anahtar değerleri eşleştiren interfacedir. Veriler key-value şeklinde tutulur. Tekrarlı anahtar değerleri içermez. Map interface i, collection interface ile benzerlik gösterir. Ama aralarındaki temel fark map interface inin elemanlarının her birinin bir anahtar değer ile eşleştirilerek saklanmasıdır. Collection interface ini kullanmazlar.
SortedMap: Map interface inin özelliklerini taşır ancak bu interface de anahtar değerleri artan sırada saklanır.

Temel İşlemler

- **add :** Collection nesnesine referans verilen elemanı ekler.
- **remove:** Collection nesnesinden referans verilen elemanı siler.
- **size :** Collection nesnesindeki eleman sayısını verir.
- **isEmpty :** Collection nesnesinin boş olup olmadığını kontrol eder.
- **contains :** Collection nesnesini içerisinde referans olarak belirtilen elemanı arar.
- **iterator :** Collection nesnesini iterasyon nesnesine referans olarak göndermeye yarar.
- **clear :** Collection nesnesindeki elemanları temizler.

Iterator : Iterator() metodu collection nesnelerini yinelememizi sağlar.

- **hasNext :** Collection nesnesinin bir sonraki elemanının var olup olmadığını kontrol eder.
- **next :** Collection nesnesini bir sonraki elemana ulaşması için iterasyon yapar ve iterasyon nesnesinin gösterdiği elemanı geri döndürür.
- **remove :** Collection nesnesi üzerindeki iterasyon nesnesinin gösterdiği elemanı siler.

```
package iteratorOrnek;
import java.util.HashSet;
import java.util.Iterator;
public class iteratorOrnek {
	public static void main(String[] args) {
		HashSet hs = new HashSet();
		Iterator i = hs.iterator();
		while(i.hasNext()){
			i.next();}}}
```

SET (KÜMELER) : Set interface i, collection interface ini miras alır. Matematikteki kümelere benzetebiliriz. Elemanlar kümeler içerisinde sıralı bir şekilde tutulmaz. Bir elemandan bir kümede bir tane bulunur.
HASHSET : Set collection tipinin bir örneğidir. Bu collection tipi de tam olarak kümelere benzer. Elemanların sıralı olması şartı yoktur ve her elemandan bir küme içerisinde yalnız bir tane bulunabilir. Diğer özelliği de erişim sırasının belirsiz olmasıdır.

```
package hashSetOrnegi;
import java.util.HashSet;
import java.util.Iterator;
public class hashSetOrnegi {
	public static void main(String[] args) {
		HashSet<String> hs = new HashSet<String>();
```

```
		boolean b1,b2;
		b1 = hs.add("Mahmut");
		hs.add("Ali");
		hs.add("Şahin");
		hs.add("Kaya");
		hs.add("Java");
		hs.add("Ahmet");
		b2 = hs.add("Fatma");
		Iterator i = hs.iterator();
		System.out.println(b1 +" - "+ b2);
		while(i.hasNext()){
			System.out.println(i.next());}}}
```

LINKEDHASHSET : HashSet collection tipine genel olarak benzer. Ama bir farkı bu collection tipinde elemanlar birbirine çift yönlü bağlıdır.

```
package linkedSetOrnegi;
import java.util.Iterator;
import java.util.LinkedHashSet;
public class linkedSetOrnegi {
	public static void main(String[] args) {
		LinkedHashSet<String> lhs = new LinkedHashSet<String>();
		lhs.add("Mahmut");
		lhs.add("Ali");
		lhs.add("Şahin");
		lhs.add("Kaya");
		lhs.add("Ahmet");
		lhs.add("Mehmet");
		Iterator i = lhs.iterator();
		while(i.hasNext()){
			System.out.println(i.next());}}}
```

SORTED SET : Set collection interface inin alt interface i dir. Genel olarak aynıdır ancak elemanları sıralı tutması ile HashSet collection tipinden ayrılır. Bu interface in kullanılan öğesi TreeSet tir.

TREESET : Yukarıda belirttiğim gibi SortedSet interface inin alt sınıfıdır. Genel olarak HashSet ile aynı metodları kullanır. Bu collection tipinde tip kontrolü yapılır. Yani kümeye farklı tipte bir eleman eklemeye çalışırsak derleme anında hata alırız.

```
package sortedSetOrnegi;
import java.util.Iterator;
import java.util.TreeSet;
public class treeSetMetotOrnegi {
	public static void main(String[] args) {
		TreeSet ts = new TreeSet();
		ts.add(1);
		ts.add(17);
		ts.add(42);
		ts.add(33);
		ts.add(19);
		ts.add(71);
		System.out.println("Ağaç yapımızın elemanları :" + ts);
		System.out.println("Ağaç yapımızın en küçük elemanı : " + ts.first());
		System.out.println("Ağaç yapımızın en büyük elemanı : " + ts.last());
		System.out.println("22'den küçük elemanlar : " + ts.headSet(22));
		System.out.println("7 ile 27 arasındaki değerler" + ts.subSet(7, 27));
		System.out.println("33 ten büyük elemanlar : " + ts.tailSet(33)); }}
```

LIST Yapıları : List collection interface i, verileri dizi halinde tutar. Dizilerde olduğu gibi burada da elemanların index değerleri vardır. Index değerleri ile elemanlara erişebilir ve arama yapabiliriz. List yapılarını Array lerden ayıran en önemli özellik, List yapılarının boyutlarının önceden ayarlanmış olmamasıdır.

LISTITERATOR : ListIterator () metodu ile elemanları gezebiliriz. Metodları ise;

- **add:** referans olarak verilen elemanı listeye ekler.

- **hasPrevious:**ListIterator() metodu ters yönde bir elemanı işaret ediyorsa geriye true değer döndürür. Eleman yoksa geriye false döndürür.
- **nexIndex:** next() metodu tarafından geri döndürülmesi gereken elemanın index değerini döndürür. İterasyon yapmaz.
- **previous :** Listesi sondan başlayarak bir önceki elemana ulaşması için iterasyon yapar ve iterasyon nesnesinin gösterdiği elemanı geri döndürür.
- **previousIndex :** previous () metodu tarafından geri döndürülmesi gereken elemanın index değerini döndürür ve iterasyon yapmaz.
- **set :** next() veya previous () metodu tarafından döndürülen elemanı referans verilen eleman ile değiştirir.

ARRAYLIST : Collectionlarda en çok kullanılan tiptir. List interface ini kullanırlar. ArrayList sınıfının elemanlarına index değerleriyle erişmek veya silmek mümkündür. Liste elemanları sıralı olarak tutulur. Null değerleri alabilir ve aynı elemandan birden fazla eklenebilir. Boyutu içine eleman eklendikçe büyür. Elemanları bir dizi biçimde saklar. ArrayList, List interface ini implemente eder.

- **add()** metodu ile eleman eklenir.
- **get()** metodu ile elemana ulaşılır.
- **remove()** metodu ile de index i verilen eleman silinir.

```
package listeOrnekleri;
import java.util.ArrayList;
import java.util.ListIterator;
public class listeOrneleri {
	public static void main(String[] args) {
		ArrayList listeYapisi = new ArrayList();
		listeYapisi.add("Edirne");
		listeYapisi.add("İstanbul");
		listeYapisi.add("Kırklareli");
		listeYapisi.add("Tekirdağ");
		listeYapisi.add("Çanakkale");
		listeYapisi.add(1);
		listeYapisi.add("Edirne");
		listeYapisi.set(5, "Bursa");
		ListIterator li = listeYapisi.listIterator();
		System.out.println("Baştan tarama");
		while(li.hasNext()){
			System.out.print("Index değeri : " + li.nextIndex() + " ");
			System.out.println("Elemanımız : " + li.next());
		}
		System.out.println("Sondan tarama");
		while(li.hasPrevious()){
			System.out.print("Index değeri : " + li.previousIndex() + " ");
			System.out.println("Elemanımız : " + li.previous());
		}
		System.out.println(listeYapisi.get(4));}}
```

LINKEDLIST : LinkedList yapısında elemanlar birbirine eklenirken bir bağ konur. Bu bağ tek veya çift yönlü olabilir. List interface inin alt sınıfıdır. List interface inin sınıflarının kullandığı ortak metodların hepsini kullanabilir. Ancak bu collection yapısı listenin sonuna veri ekleme, listenin başından veri çekme gibi bazı özelleşmiş metodlara sahiptir.

```
package bagliListeOrnek;
import java.util.LinkedList;
public class bagliListeOrnek {
	public static void main(String[] args) {
		mahmut yeniMahmut = new mahmut();
		yeniMahmut.ekle(1);
		yeniMahmut.ekle(2);
		yeniMahmut.ekle(3);
		yeniMahmut.ekle(4);
```

```
                yeniMahmut.ekle(5);
                yeniMahmut.ekle(6);
                System.out.println("Mahmut yapısının elemanları : " + yeniMahmut.bagliListe);
                System.out.println("Tepe Eleman : " + yeniMahmut.tepeEleman());
                System.out.println("Mahmut yapımızın ilk elemanı : " + yeniMahmut.getir());
                System.out.println("Tepe Eleman : " + yeniMahmut.tepeEleman());
                System.out.println("Mahmut yapımızın ilk elemanı : " + yeniMahmut.getir());
                System.out.println("Tepe Eleman : " + yeniMahmut.tepeEleman());
                System.out.println("Mahmut yapısının elemanları : " + yeniMahmut.bagliListe);}}
class mahmut {
        public LinkedList bagliListe = new LinkedList();
        public void ekle(Object veri){
                bagliListe.addFirst(veri);
        }
        public Object getir(){
                return bagliListe.removeFirst();
        }
        public Object tepeEleman(){
                return bagliListe.getFirst();}}
```

VECTOR : Vectorler içine eleman yüklendikçe büyüyen dizilerdir. Vector sınıfı senkronizedir. Bu nedenle ArrayList sınıfı Vector sınıfından daha hızlıdır. Ayrıca Vectorlerin kapasitesinin ne değerde artacağını uygulamalar belirleyebilir.

QUEUE : FIFO (ilk giren ilk çıkar) mantığı ile çalışır. En belirgin özelliği index değerleri ile işlem yapmazlar. Yani Queue içerisindeki elemana index değerleri ile erişilemez.

```
package kuyrukOrnegi;
import java.util.LinkedList;
import java.util.Queue;
public class kuyrukOrnegi {
        public static void main(String[] args) {
                Queue kuyruk = new LinkedList();
                kuyruk.add("Bir");
                kuyruk.add("İki");
                kuyruk.add("Üç");
                kuyruk.offer("Dört");
                kuyruk.offer("Beş");
                kuyruk.offer("Altı");
                System.out.println("kuruktaki elemanlar ---> " + kuyruk );
                while(!kuyruk.isEmpty()){
                        System.out.println("Kuyrukta bekleyen eleman : " + kuyruk.peek());
                        System.out.println("Kuyruktan çekilen eleman : " + kuyruk.poll());}}}
```

MAP : Map interface i, Collection interface inin bir parçası değildir. Bu interface, elemanlarını anahtar değerleri ile eşleştirerek saklar. Anahtar değerler kopya eleman bulundurmaya izin vermez, elemanlar arasında ise kopya değer bulunabilir. Bazı metodları ise;

- **containsKey (Object key):** Map içerisinde referans olarak verilen anahtar değer varsa true yoksa false değer döndürür.
- **containsValue(Object Value) :** Map içerisinde referans olarak verilen değer varsa true yoksa false değer döndürür.
- **entrySet():** Map içerisindeki elemanları Set (Küme) collection öğesi olarak verir.
- **get (Object key) :** Map içerisinden, referans olarak verilen anahtar değere karşılık gelen elemanı verir.
- **keySet():** Map içerisindeki anahtarları Set (Küme) collection öğesi olarak verir.
- **put (Object key, Object value) :** Map içerisine referans olarak verilen elemanı anahtar değere eşleyerek kaydeder.
- **remove (Object key) :** Map içerisinden anahtar değere karşılık gelen eleman silinir.
- **values () :** Map içerisindeki değerleri Collection öğesi olarak verir.

HASHMAP : Map interface inin bir örneğidir. Bu sınıf her elemana karşılık bir anahtar değeri saklar. Anahtar değerler veya veriler sıralı olmak zorunda değillerdir. Çekilen verilerin sırası belirsizdir.

```
package mapOrnekleri;
import java.util.HashMap;
import java.util.Set;
public class hashMapOrnegi {
	public static void main(String[] args) {
		HashMap hm = new HashMap(10,0.5f);
		hm.put(17, "Çanakkale");
		hm.put(22, "Edirne");
		hm.put(39, "Kırklareli");
		hm.put(59, "Tekirdağ");
		hm.put(59, "İstanbul");
		System.out.println("HashMap öğemiz içerisindeki veriler : " + hm);
		Set anahtarlar = hm.keySet();
		Set veriler = hm.entrySet();
		System.out.println("Anahtarlarımız : " + anahtarlar);
		System.out.println("Verilerimiz : " + veriler);}}
```

LINKEDHASHMAP : Map interface inin bir öğesidir. Genel olarak HashMap ve LinkedList yapılarının özelliklerini içerir. Öğelerin dönüş sıralarını tahmin edebiliriz.

```
package mapOrnekleri;
import java.util.Collection;
import java.util.LinkedHashMap;
import java.util.Set;
public class linkedHashMapOrnegi {
	public static void main(String[] args) {
		LinkedHashMap<String, Integer> lhm = new LinkedHashMap<>();
		lhm.put("Çanakkale", 17);
		lhm.put("Tekirdağ", 59);
		lhm.put("Edirne", 22);
		lhm.put("Kırklareli", 39);
		System.out.println("Bağlı HashMap yapımız : " + lhm);
		Set anahtar = lhm.keySet();
		System.out.println("Anahtarlarımız : " + anahtar);
		Collection deger = lhm.values();
		System.out.println("Degerlerimiz : " + deger);}}
```

SORTEDMAP : Map interface inin alt interface idir. Verileri sıralı olarak saklar. Verileri artan sırada saklar ve verileri anahtarlarla eşleyerek saklar. Map interface ine ek olarak iki metod daha sunar bunlar;

- firstKey(): Öğe içerisindeki ilk anahtar değerini döndürür.
- lastKey(): Öğe içerisindeki son anahtar değerini döndürür.

TREEMAP : İçerisine girilen verileri sıralı bir biçimde tutar. Diğer Map interface öğelerinden farklı bir takım metodlara sahiptir.

```
package sortedMapOrnek;
import java.util.TreeMap;
import java.util.TreeSet;
public class treeMapOrnegi {
	public static void main(String[] args) {
		TreeMap agac = new TreeMap<>();
		agac.put(17, "Çanakkale");
		agac.put(22, "Edirne");
		agac.put(39, "Kırklareli");
		agac.put(16, "Bursa");
		agac.put(34, "İstanbul");
		agac.put(71, "Kırıkkale");
		System.out.println("TreeMap yapımız : " + agac);
		System.out.println("22 ve 22den büyük degerler : " + agac.ceilingEntry(22));
		System.out.println("22 ve 22den büyük anahtarlar : " + agac.ceilingKey(22));
		System.out.println("Azalan sırda anahtarlar : " + agac.descendingKeySet());
		System.out.println("Azalan sırada yapımız : " + agac.descendingMap());
```

```
System.out.println("22 ve 22den küçük degerler : " + agac.floorEntry(22));
System.out.println("22 ve 22den küçük anahtarlar : " + agac.floorKey(22));
System.out.println("22den küçük degerler : " + agac.headMap(22));
System.out.println("22den büyük degerler : " + agac.higherEntry(22));
System.out.println("22den büyük anahtarlar : " + agac.higherKey(22));
System.out.println("22den küçük anahtarlar : " + agac.lowerKey(22));
System.out.println("22den büyük degerler : " + agac.lowerEntry(22));
System.out.println("22 ile 39 arasındaki deger : " + agac.subMap(22, 39));
System.out.println("22 ve 22den büyük deger : " + agac.tailMap(22));}}
```