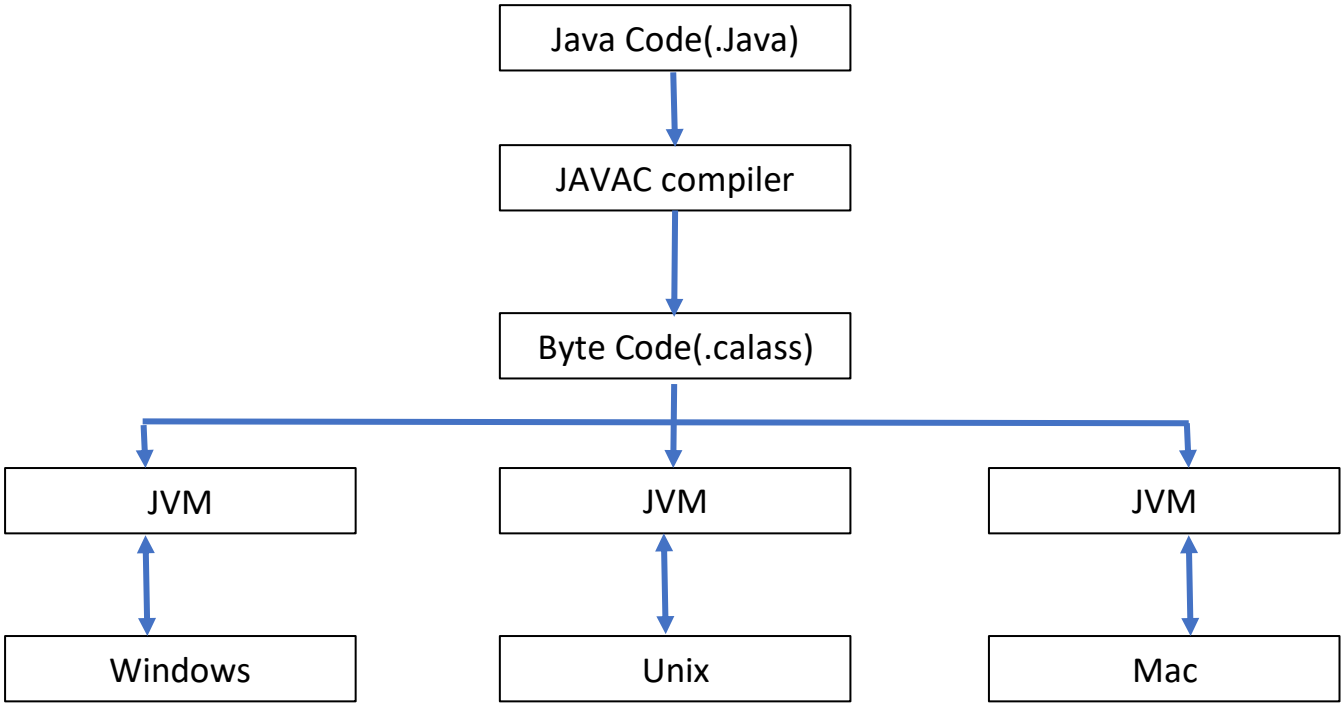
3. Java'nın platform bağımsızlığını nasıl sağladığını anlatınız.

Öncelikle platform bağımsızlığı nedir anlamamız gerekiyor. Platform bağımsızlığı, aynı kodu birden fazla platformda değişiklik yapmadan çalıştırabilmemizdir. Platforma örnek olarak Mac , Windows veya Linux gösterilebilir.

C, C++ gibi dillerde yazdığımız kodu compiler makine diline(binary code) çevirir. Ardından o makinede çalışılması sağlanır. Compiler kodu makine diline çevirirken o cihazın işletim sistemini kullanır. Farklı işletim sistemlerinde farklı binary code 'lar oluşur. Bu yüzden bu dillerde platformdan bağımsızlık söz konusu değildir.

Platformdan bağımsızlığı sağlayabilmemiz için Compiler kodumuzu, farklı işletim sistemlerine sahip cihazların anlayacağı ortak bir yapıya çevirmesi gerekmektedir. Java da platform bağımsızlığı tamda bu şekilde sağlanıyor.



Yukarıdaki şekile bakacak olursak Javac Compiler kodumuzu C,C++ da ki gibi Binary Code yerine, Farklı işletim sistemlerinin anlayacağı yapı olan Byte Code a çeviriyor. Peki bu Byte Code' ları Farklı işletim sistemlerine ait makineler nasıl anlayabiliyorlar. İşte tamda burada devreye JVM(Java Virtual Machine) giriyor. Java sanal makinesi kendisine gelen Byte Code' ları bulunduğu ortamdaki işletim sisteminin anlayacağı makine diline(Binary Code) çevirir.

Özetleyecek olursak Javada yazdığımız code' u Javac Compiler, Byte Code' a çeviriyor. Farklı platformlarda bulunan JVM' ler ise bu Byte Code' ları bulundukları platformdaki işletim sistemini kullanarak o makinenin anlayacağı dile çeviriyor. Böylelikle platform bağımsızlığı sağlanmış oluyor.

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

Heap ve Stack bellekte(RAM' da) bulunan mantıksal yapılardır. Stack' de değer tipleri ve adresler saklanırken Heap de ise referans tipleri saklanır.

Stackte veriler üst üste LIFO(Son giren ilk çıkar) mantığı ile dizilir, sırası gelmeden aradaki bir değer ile işlem yapılamaz.

Örnek verecek olursak Primitive tipler (int, float, double, short, byte, decimal, long) stack de tutulur. Referans tip referansları stack de tutulurken değerleri heap de tutulur.

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

String class'ını incelediğimizde öncelikle String class'ı final tanımlanmıştır. Bunun sebebi başka classlar tarafından extends edilmesini önlemektir. Değişkenlerine baktığımızda hepsinin private tanımlandığını görüyoruz. Bunun sebebi değişkenlere dışarıdan herhangibir müdahaleyi engellemektir. Biz private değişkenleri dışarıdan ulaşmak ve değiştirmek istediğimizde bunu private değişkenin bulunduğu class'lardaki fonksiyonlar ile yapıyorduk. Dışarıdan gelecek müdahalelere engel olmak için dışarıdan bu değişkenlere ulaşıp değiştirebilecek fonksiyonları yani setter fonksiyonlarını oluşturmamalıyız. String sınıfının içine baktığımızda setter fonksiyonları olmadığını görüyoruz. String classını incelemeye devam ettiğimizde değişkenlerin final olarak tanımlandığını görüyoruz, final tanımlanan değişkenler ilk değerlerini aldıktan sonra değiştirilemezler. İncelediğimiz bu özellikler ile String immutable olmayı sağlamaktadır.

6. Java neden çoklu kalıtımı desteklemez açıklayın.

Java'nın çoklu kalıtımı desteklememesindeki temel amaç karmaşıklığı azaltmaktır. Karmaşıklıklara örnek olarak diamond problemi gösterilebilir. Java bu problemi çözmek için karmaşık bir yol sağlamayı tercih etmemiştir.

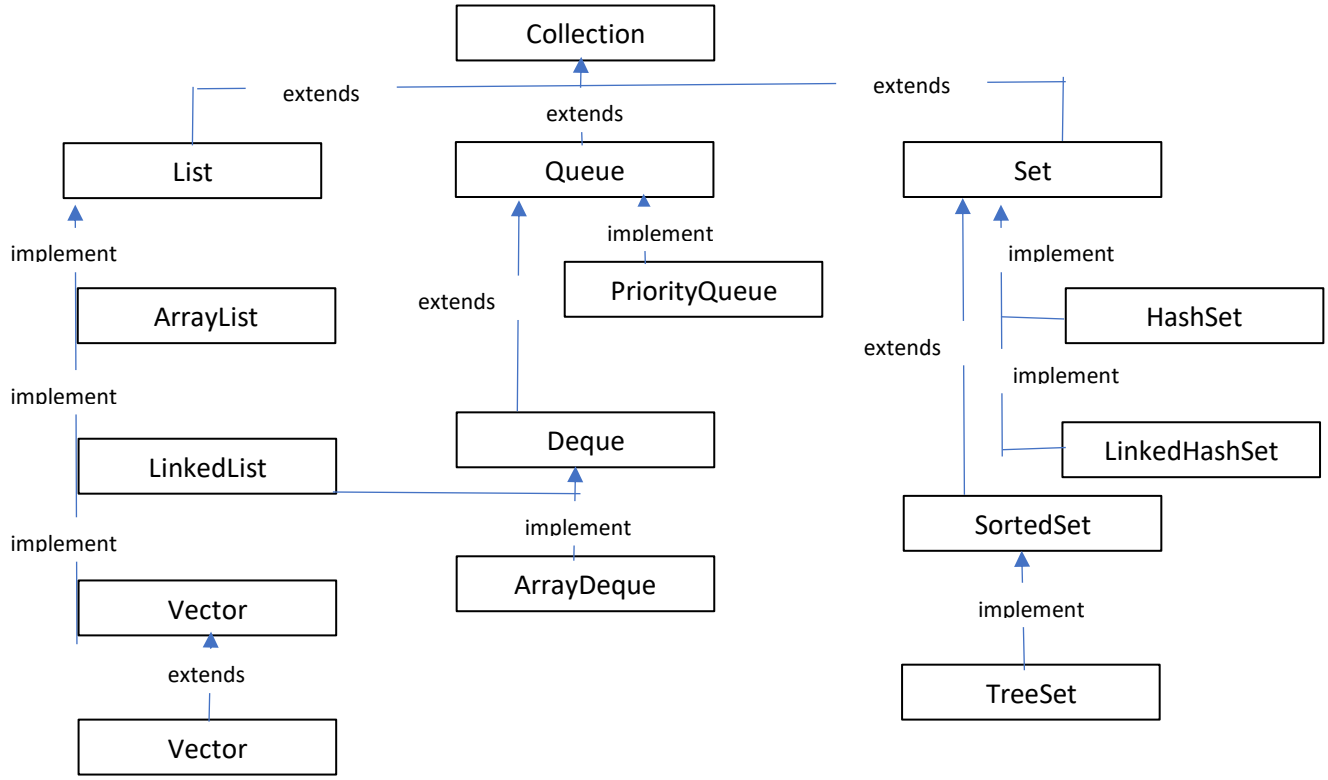
7. Build Tool nedir? Java ekosistemindeki build toollar neler açıklayın.

Build tool uygulamaların kaynak kodundan oluşturulmasını otomatik bir şekilde sağlar. Bu araçlar kaynak kod derleme gibi tekrarlanan görevleri otomatik bir hale getirmeyi sağlayıp bu iş yükünü geliştiriciler üzerinden almak için vardırlar.

Java eksisteminde bulunan build toollara örnek verecek olursak, Ant, Maven, Gradle örnek verilebilir.

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

Öncelikle Collection framework içindeki tüm yapıları şekil üzerinde gösterelim.



Sıra ile bu yapıları inceleyecek olursak, Collection interfacesinden List, Queue, Set interfaceleri extends almaktadır. Yani tepede çatı olarak Collection interfacesi vardır.

List interfacesinden kalıtım alan alt sınıflara bakacak olursak bunlar, ArrayList, LinkedList, Vector ve Stack dir. Bu sınıfları ayrıntılı inceleyecek olursak ;

1. **ArrayList**

Veri saklamak ve veriye erişimin yoğun olduğu durumlarda tercih edilir. Tanımlanırken <> işaretleri arasında içinde tuttuğu verinin tipi yazılır.Arraya

ekleme yada silme işlemi yapıldığında tüm indisler kaydırılır bu yüzden verimi düşüktür.

List<String> nameList = new ArrayList<String>(); (Listemizi tanımlıyoruz)

Sıfırıncı indisten başlayarak elemeanları sırası ile yerleştirir.

```
nameList.add("Beytullah");
nameList.add("Cem");
nameList.add("Sami");
nameList.add("Mehmet");
nameList.add("Cem");
```

size() metodu ile listenin boyunu bulabiliriz.

```
System.out.println("Listenin boyutu : " + nameList.size());
```

listede get ile istediğimiz indisdeki elemanı getirebiliriz.

```
System.out.println("1. index: " + nameList.get(1));
System.out.println("2. index: " + nameList.get(2));
```

Liste başından aramaya başlarayarak ilk karşılaştığı Cem kelimesinin indisini verir.

```
System.out.println(" Cem' in bulunduğu ve ilk indise en yakın indis: " +
nameList.indexOf("Cem"));
```

Liste sonundan taramaya başlayarak ilk karşılaştığı noktadaki Sami bilgisinin indis değerini verir.

```
System.out.println("Sami' nin bulunduğu ve son indise en yakın indis: " +
nameList.lastIndexOf("Sami"));
```

İlk olarak verilen indise gidip verilen elemanı koyar.
Verilen indiste daha önceden tanımlanmış değer varsa, fonksiyondaki elemanı o indise koyar ve geri kalan elemanları kaydırır.

```
nameList.add(3, "Ali");
```

set() fonksiyonuna ilk verilen argüman indis, sonraki elemandır.
Verilen indise gidip verilen elemanı koyar.

Verilen indis, liste boyutunun dışında olmamalıdır.Yani listede elemean olmayan bir indise setleme işlemi yapılmaz.
nameList.set(2, "Ayşe");

contains ile liste içinde aradığımız eleman var mı yok mu belirtir.
varsa true, yoksa false döner
System.out.println(nameList.contains("Ali"));
System.out.println(nameList.contains("Kemal"));

remove fonksiyonu ile listeden belirttiğimiz indisteki değer silinir.
silinen değer fonksiyondan bize geri döndürülür.
String ilkEleman = nameList.remove(0);
System.out.println(ilkEleman + " elemanı listeden silinmiştir..");

List<String> newNameList = new ArrayList<String>();
newNameList.add("Merve");
newNameList.add("Melike");

Bir listenin tüm elemanlarını başka bir listeye eklemek için "addAll" fonksiyonu kullanılır.
nameList.addAll(newNameList);

listeden alt bir liste oluşturmak için "sublist" fonksiyonunu kullanılır.
Başlangıç ve bitiş indisleri verilir.
Başlangıç indisindeki eleman dahil, bitiş indisindeki eleman hariç yeni bir liste oluşturulur.
List<String> subList = nameList.subList(4, 6);

System.out.println("Name listin alt listesi (4. eleman dahil 6. eleman dahil degil)");
System.out.println(subList);

toArray fonksiyonu parametresiz çağırırsak Object tipinde bir dizi dönmüş olur.
Object[] objectArray = nameList.toArray();

toArray fonksiyonuna hangi tipte bir dizi oluşturmak istiyorsak,

o tipten bir nesne üretip parametre olarak göndeririz.
String tipinden bir dizi almak istediğimiz için "new String[0]" şeklinde bir nesne üretip, "toArray" fonksiyonuna gönderdik.

```
String[] stringArray = nameList.toArray(new String[0]);
```

listedeki tüm elemanları siler.

```
nameList.clear();
```

2. **LinkedList**

Bağlı listeler olarak düşünülebilirler.Her eleman önceki ve sonraki elemanı işaret edecek şekilde adreslerini tutar.Array listlerdeki gibi eleman eklenirken yada silinirken kaydırma yapılmaz.Veri ekleme çıkarmada arrayliste göre verimlidir.Array listteki çoğu metod LinkedList içinde geçerlidir.

3. **Vector**

Vector<String> vector = new Vector<String>(); Şeklinde tanımlanır

Arraylistlere benzerler,

Vector.add(“Ankara”);

Şeklinde eleman ekleyebiliriz. Arraylistlerdeki gibi boyutunu vector.size() ile bulabiliriz.

İlk 3 örneğimiz list türünden örneklerdi. Şimdi Set interfacesini inceleyelim.

Set interfacesinin alt sınıfları HashSet, LinkedHashSet, TreeSet olmak üzere 3’ e ayrılır.

1. **HashSet**

Set dediğimizde aklımıza matematik deki kümeler gelebilir, kümelerin en önemli özelliği küme içinde tekrar eden yani aynı değerden bulunamamasıdır. HashSet veri kümesi içindeki elemanlara ekleme silme ve erişim imkanı sağlar.

HashSet<String> hashSet = new HashSet<>(); şeklinde tanımlanır.

hashSet.add(“saat”);

hashSet.add(“fare”);

Şeklinde eleman eklenir .

hashSet.remove(“saat”); ile bu elemanı silebilriz.

HashSet.contains(“saat”); contains ile istediğimiz elemanı içerip içermediğini kontrol edebiliriz.

Iterator kullanarak elemanlarını yazdırabiliriz.

```
Iterator<String> itr = hashSet.iterator();
    while (itr.hasNext()) {
       System.out.println(itr.next());
    }
```

2. **LinkedHashSet**

HashSet de herhangibir sıra sözkonusu değildir LinkedHashSet de eklenilen sıra ile geri dönüş yapılır.HashSetin sıralı halidir denebilir. HashSet deki fonksiyonlar LinkedHashSettede Geçerlidir.

3. **TreeSet**

TreeSet de eklenen verilerin istenilen sıraya göre sıralanmasını sağlayabiliriz. Tabiki bunu sağlayabilmemiz için kullanacağımız sınıfın sıralanabilir olması gerekmektedir bunun içinde Comparable interfaesinden kalıtım alıp comparaTo metodunun doldurulması gerekmektedir, compareTo metodu default olarak küçükten büyüğe doğru sıralama yapar.

Örnek verecek olursak ;

TreeSet<Integer> numbers = new TreeSet<>();

```
numbers.add(5);
numbers.add(2003);
numbers.add(1);
numbers.add(-23);
numbers.add(33);

for (Integer number : numbers) {
   System.out.println(number);
}
```

Şeklinde yazdırmak istediğimizde alacağımız çıktı küçükten büyüğe doğru sıralanmış şekilde olacaktır.

Queue interfacesinin alt sınıfları, Queue LinkedList ve PiorityQueue dir.

1. **Queue LinkedList**

İşlemlerden geçmeden önce ögelerin depolanmasını sağlar.Collection arayüzünün bir alt arayüzü olduğundan onun ullandığı tüm metodları kullanır.

Onlara ek olarak kuyruk yapısına ekleme silme gibi işlemleri kolaylaştıran metodlarıda kullanır.

- element() : kuyruğuna başındaki ögeyi bize verir. Verilen öge kuyruktan atılmaz
- add(eleman): parametrede verdiğimiz elemanı kuyruğa ekler. İşlem başarısız olursa hata fırlatır.
- offer(eleman): parametrede verdiğimiz elemanı kuyruğa ekler. İşlem başarısız olursa null döner.
- poll(): Kuyruğun başındaki elemanı kuyruktan çıkartır.
- peek(): Kuyruktaki sıradaki elemana ulaşmak istersek kullanırız.

Yukarıdaki tanımlara örnek yapacak olursak ;

```
Queue<String> queue = new LinkedList<>();

queue.add("Beytullah");
queue.add("Cem");
queue.add("Veli");
queue.offer("Ali");
queue.offer("Ayşe");
```

```
System.out.println(queue.remove());
System.out.println(queue.element());
System.out.println(queue.poll());
```

2. **PriorityQueue**

Bir öncelik belirleyip bu önceliğe göre işlemlerin yapıldığı bir koleksiyon çeşididir. Bu önceliği Comparator interfacesindeki compare metodunu kendimiz oluşturarak isteğimize göre sıralayabiliriz.

Queu LinkedList deki metodlar burada da kullanılabilirler.