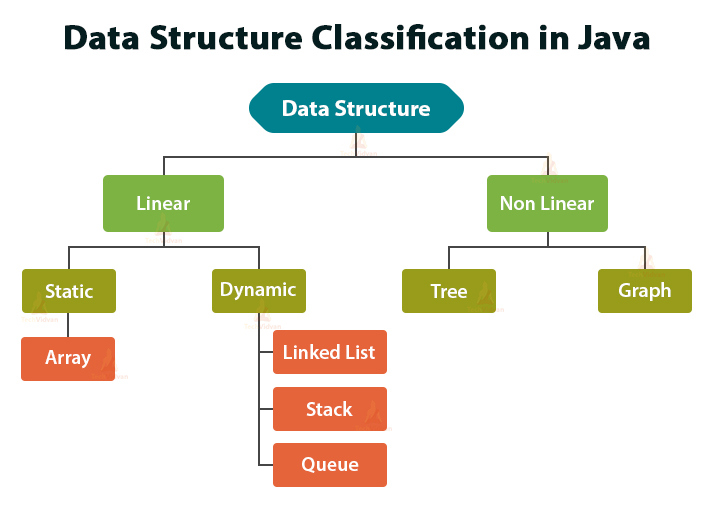
**VERİ YAPILARI**

Veriyi hafıza(ram) depolayan ve yönetilmesini sağlayan yapılardır. Yerleşik ve türetilmiş olmak üzere 2 ana yapıya ayrılırlar. Türetilmiş kısmı ise genel olarak şöyledir:



**Array(Dizi) and ArrayList(Dizi Listesi)**

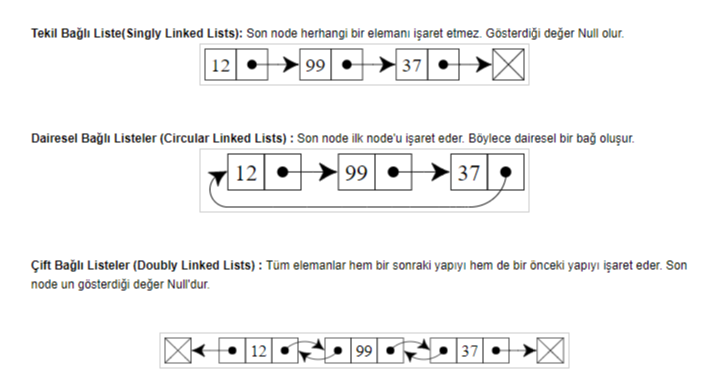
**Array**: Genellikle aynı tür verilerin depolandığı, static hafızaya sahip ve indexler(İndis) ile erişimin sağlandığı veri yapısıdır.

**ArrayList**: İndex(indis-dizin) olduğundan verilere rastgele erişilebilir. İstenilen elemanı silme ekleme zor. Tüm dizi değişir. Çünkü kayma olur. Dinamik dizi yapı.

**LinkedList (Bağlı Liste)**

Bir veri ve kendi tipinde yapıyı gösteren işaretçiye sahip düğümlerden oluşan yapıdır. İhtiyaç oldukça büyüyen **dinamik hafıza**lı bir yapıdır. **Data-Pointer(düğüm/node)** Baştan sıralı şekilde aranılan satıra gelir. İstenilen elemanı silme ekleme kolaydır. Çünkü tümü değişmez.

Bu yapıda kullanılan node yani düğüm ile hem veri depolanır hem de bir sonraki veriye bir bağlantı kurularak verilerin doğrusal olarak bağlantılı olması sağlanır.



ARRAYLİST LİNKEDLİST FARK

|  |
| --- |
|  |
| **Karşılaştırma için temel** | **ArrayList** | **Bağlantılı liste** |
| Temel | listedeki öğelere rasgele erişime izin verir. | LinkedList listedeki öğelere rasgele erişime izin vermiyor. |
| Veri yapısı | Öğeleri depolamak için kullanılan iç yapı dinamik dizidir. | Öğeleri saklamak için kullanılan iç yapı iki kat bağlantı listesidir. |
| Uzattı | ArrayList, AbstarctList sınıfını genişletir. | LinkedList, AbstractSequentialList öğesini genişletir. |
| uygular | AbstractList Liste arayüzünü uygular. | LinkedList, List, Deque, Queue uygular. |
| Erişim | ArrayList'te listedeki öğelere erişim daha hızlıdır. | LinkedList'te listedeki öğelere erişim daha yavaştır. |
| hile | ArrayList'te listedeki öğelere değişiklik yapmak daha yavaştır. | LinkedList'te listedeki öğelere değişiklik yapmak daha hızlıdır. |
| davranış | ArraylList, listeyi uyguladığı gibi Liste olarak davranır. | LinkedList, hem List hem de Kuyruğu uygulayan Sıra kadar Liste davranır. |

Stack (Yığın/Yığıt (LIFO (last-input-first-output))

Stack (yığıt) sınıfı nesnelerin LIFO (last-input-first-output) yapısıyla depolanmasını sağlayan bir veri tipidir. Bu yapıda veriler alınır ve üst üste konulur. Eleman çekilmek istendiğinde üstteki yani en son gelen eleman ilk olarak çekilir. Dolayısıyla alttan eleman çekilme işlemi yok. Üst üste dizilmiş tabakları(veya kitap) düşünürsek, alttakileri alabilmemiz için üsttekileri kaldırmamız gerekir.

Push () =>  Yeni veri ekler

Peek()  =>  En üsteki veriyi okur.

Pop()   => En üsteki veriyi siler.

Contains () => Arama yapar.

Clear()   =>  Temizler.

toArray()  => Stack’ı diziye aktarır.



Queue (Kuyruk (FIFO - First in First Out)

Stack yapısında veri depolayabilen ancak FIFO yani ilk giren ilk çıkar yaklaşımı ile veriler üzerinde işlem yapan veri yapısıdır. Basitçe bir gişe önündeki bilet kuyruğu veya bilet sırası olarak düşünülebilir.



enqueue () - kuyruğa bir öğe ekleyin (saklayın).

dequeue () - kuyruktan bir öğeyi kaldırır (erişim).

peek () - Sıranın önündeki öğeyi çıkarmadan alır.

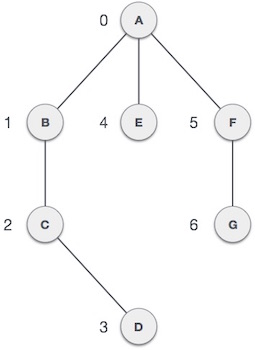
isfull () - Kuyruğun dolu olup olmadığını kontrol eder.

isempty () - Kuyruğun boş olup olmadığını kontrol eder.

**Tree(Ağaç):** Veriler hiyerarşik ve doğrusal olmayan bir şekilde tutulur. Verilerin oluşturduğu düzen ağaca benzediği için bu yapıya ağaç denmiştir.



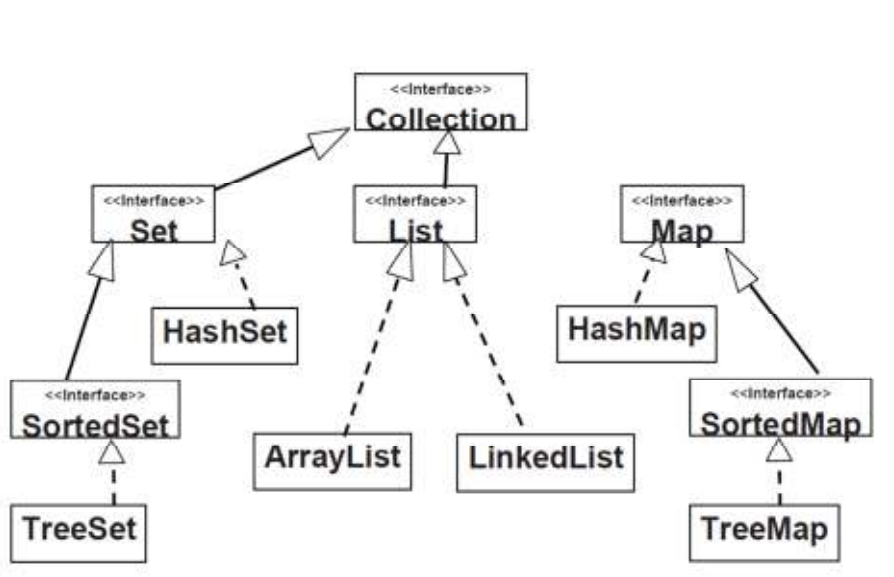
**Graph(Graf-Çizge):** Verileri doğrusal olmayan bir şekilde saklar. Graf, matematiksel anlamda, düğümlerden ve bu düğümler arasındaki ilişkiyi gösteren kenarlardan oluşan bir kümedir. Mantıksal ilişki, düğüm ile düğüm veya düğüm ile kenar arasında kurulur. Bu veri yapısı bilgisayar ağları gibi gerçek hayattaki sistemleri temsil etmek için kullanışlıdır. İnternet, şehirler arası en kısa yolun tespiti, havayolu ağı gibi işlemlerde graflar yaygın bir şekilde kullanılır.



**KOLEKSİYONLAR**

İçerisinde bir veya daha fazla nesne taşıyan ve gerektiğinde ekle-sil gibi işlemler yapan başka bir nesnedir. Koleksiyonun içerisindeki nesnelere koleksiyonun elemanları denir.

* **Set Nesnesi:** 1 eleman tutar. Aynı eleman yok.
* **List Nesnesi:** Elemanları sıralı şekilde tutar. Aynı eleman var.
* **Map Nesnesi:** Her biri birbirinden farklı anahtarlar ile eşleştirilen nesnelerden oluşur.



**SortedSet**: Artan sırada sıraya dizilmiş nesneler kümesi. Set interface i ile implement edilmiştir.

**SortedMap**: Anahtarlarına göre artan sırada dizilmiş nesneler topluluğu.

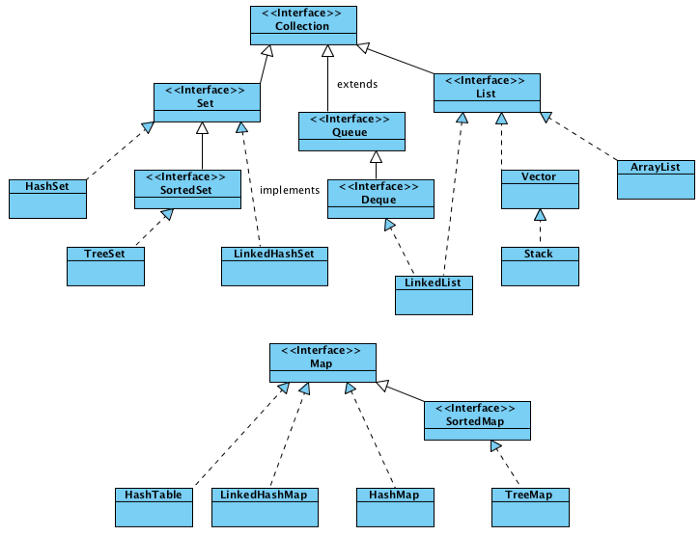
**HashMap:** Map interface’ini implemente etmiştir. Key değerlerinden hash üretilerek objeler saklanır.

**HashSet:** Set arayüzünü implemente eder. Veriler sırasızdır.

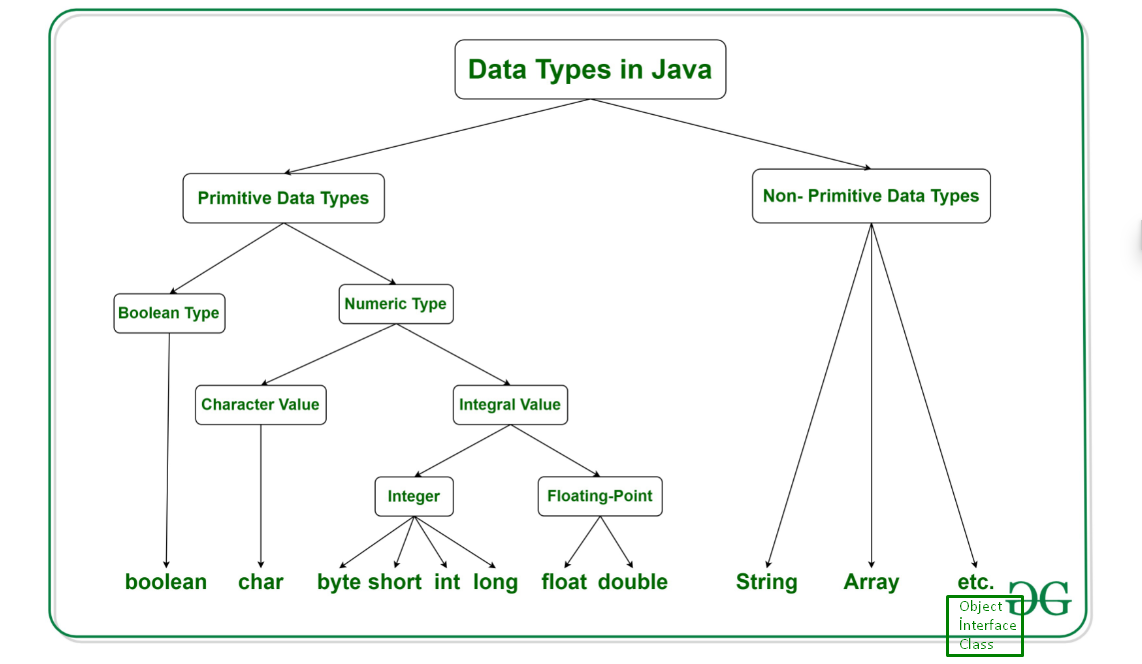
**TreeSet:** Benzersiz öğeleri korur. Artan sırada saklar.

**TreeMap:** Veriyi anahtarına(key/value) göre artan sırada saklar.

*TreeMap* ile HashMap arasında ki temel fark treemap in key değerlerini kendisi otomatik olarak sıralayabiliyor olmasıdır.



**VERİ TÜRLERİ**



ENUM

* Tüm enumlar gizlice java.lang.Enum sınıfından türer. Java’da bir sınıfın yalnızca bir ebeveyni olabildiğinden, bir enum başka birşeyden türeyemez.
* Enum’lar herhangi bir sınıfın üst sınıfı olamaz.
* new anahtar kelimesi ile Enum nesneleri oluşturulamaz.
* Enum’lar yapılandırıcılara, bloklara ve statik bloklara, değişkenlere ve metotlara sahip olabilir.
* Enum’lar interface’leri implement edebilir.