8. Collection framework içerisindeki bütün yapıları önemli methodlarıyla örnekleyip açıklayınız. (20 PUAN)

Koleksiyon yazılım çatısı java.util paketinde bulunmaktadır. Temel amacı benzer verileri gruplar halinde tutarak bu veriler üzerinde kolay bir şekilde işlemler yapabilmemizi sağlamaktır.

Bu kütüphanede ana arayüz olarak 'Iterable' arayüzü yer almaktadır. Bu arayüz kendisinden kalıtım alan nesnelerin for each döngüsüne alınabilmesini sağlar. Bu döngü sayesinde dizi ve koleksiyon tabanlı nesne elemanlarını sırasıyla kolayca döndürerek bu elemanlar üzerinde işlemler yapabiliriz.

Devamında ise Iterable arayüzünü implemente eden 'Collection' arayüzü karşımıza çıkmaktadır. Bu arayüz koleksiyon yapılarının ekle sil, temizle gibi temel methodlarını barındırmaktadır.

LİST ARAYÜZÜ: Benzer veya farklı verilerin hafızada sıralı bir şekilde tutulmasını sağlar.

List Arayüzünü implemente eden list sınıfları: (ArrayList,LinkedList,ListedLinkedList,Vector)

ArrayList: Koleksiyon çatısı altında en çok kullanılan sınıftır. Generic(tip bağımlı) veya non-generic(tip bağımsız) olarak tanımlanabilir.

Tip bağımsız tanımlama: ArrayList myList = new ArrayList();

Tip bağımlı tanımlama : ArrayList <String> genericMyList = new ArrayList<>();

List arayüzü üzerinden tanımlama: List<veriTipi>myList = new ArrayList<>();

En çok tercih edilen tanımlama yöntemi List arayüzü üzerinden ArrayList nesnesinin oluşturulmasıdır. Bu sayede program içerisinde çok biçimlilikten yararlanılabilinir.

Önemli Methodlar:

Add methodu: ArrayList'e veri eklememize yarar. Ekleme işlemi generic list için veri tipine bağımlı non-generic list için veri tipine bağımsızdır. Yine bu methodu kullanarak istenilen indise ekleme de yapılabilir. Eklenilenecek veri parametre olarak geçilir.

myList.add(123); , myList.add("Veri"); , myList.add('v');

AddAll methodu: Mevcut ArrayList'imize aynı veri türünde bir başka ArrayList elemanlarının eklenmesi için kullanılır. Parametre olarak elemanları eklenecek olarak listeyi alır.

myList.addAll(myList2);

Remove methodu: ArrayList'den veri silmek için kullanılır. Bu method parametre olarak silinecek verinin indisini alır.

myList.remove(0);

Clear methodu: ArrayList'de bulunan bütün verileri silmemize yarar.

myList.clear();

Contains methodu: Bir verinin ArrayList içerisinde bulunup bulunmadığını denetlemek için kullanılır. Parametre olarak aranacak değer girilir. Geriye true veya false olarak boolean değer dönecektir.

myList.Contains(123);

Size methodu: ArrayList içerisinde bulunan eleman sayısını integer veri tipinde döndürür.

myList.Size();

toArray methodu: Mevcut ArrayList'i object türünden dizi kopyası döndürür.

myList.toArray();

IsEmpty methodu: ArrayList'in boş olma durumunu true veya false olarak boolean değer şeklinde döner.

myList.isEmpty()

TrimToSize methodu: Listeleri basitçe sürekli yeni elemanlar için kapasite barındıran diziler olarak düşünebiliriz. Bu sebeple yeni gelecek veriler için liste üzerinde her zaman boş yer mevcuttur. Listemizin boyutunu mevcut elaman sayımıza eşitlemek için bu methodu kullanırız.

myList.trimToSize();

Set methodu: ArrayList içerisinde bulunan belirli bir değeri bu değerin indis numarasını kullanarak değiştirmek için kullanılır.

myList.Set(0,"veri");

Get methodu: ArrayList içerisinde parametre olarak gönderilen değeri döndürür.

myList.Get(0);

IndexOf methodu: Belirtilen değerin ArrayList içerisindeki indisini döndürür. Eğer aynı değerden birden fazla mevcut ise ilk değerin indisi döner. Değer mevcut değil ise -1 döndürecektir.

myList.indexOf();

lastIndexOf methodu: Belirtilen değerin ArrayList içerisindeki indisini döndürür. Eğer aynı değerden birden fazla mevcut ise son değerin indisi döner. Değer mevcut değil ise -1 döndürecektir.

myList.lastIndexOf();

LinkedList Sınıfı: ArrayList'te implemente edilen metodların çoğu LinkedList için de geçerlidir. Benzer yapılardır ancak arka planda çalışma mantıkları farklıdır.

Tip bağımsız tanımlama: LinkedList myLinkedList = new LinkedList();

Tip bağımlı tanımlama: LinkedList <String> myLinkedList = new LinkedList<>();

Önemli farklı methodlar:

AddFirst methodu: LinkedList'lerde listenine başına ekleme yapmamızı sağlar.

myLinkedList.addFirst("veri");

AddLast methodu: LinkedList'lerde listenine sonuna ekleme yapmamızı sağlar.

myLinkedList.addLast("veri");

GetFirst methodu: Listenin ilk elemanını döndürür.

myLinkedList.getFirst();

GetLast methodu: Listenin son elemanını döndürür.

myLinkedList.getLast();

Vector Sınıfı: Array Listlere benzer yapıdadırlar. Mevcut yapısı gereği ArrayListlere göre daha düşük performanslı çalıştığından pek tercih edilmezler. ArrayList'te implemente edilen metodların çoğu Vektor için de geçerlidir. Vector'ün nesnesi oluşturulduğunda 10 elemanlık bir dizi gibi oluşacaktır. Eklemeler sonrası kapasite dolduğunda 10'ar 10'ar olmak üzere kapasite otomatik olarak artar. Nesne oluşturma sırasında Vector boyutu yapıcı methoda girilebilir. Bu şekilde oluşturulan Vectorlerin mevcut kapasitesi dolduğunda girilen değer kadar kapasite artışı gerçekleşecektir.

Tip bağımsız tanımlama: Vector myVector = new Vector();

Tip bağımlı tanımlama: Vector <String> myVector = new Vector<>();

Önemli farklı methodlar:

Capacity methodu: Vector'ün mevcut kapasitesini getirir.

myVector.Capacity();

SET ARAYÜZÜ: Verilerin benzersiz şekilde tutubilmesini sağlamaktadır. Bu arayüze bağlı sınıflarda birden fazla null değer de barınamaz.

Set arayüzünü implemente eden set sınıflar: (HashSet,LinkedHashSet,TreeSet)

HashSet Sınıfı: Özellikle tekrarlanan veriler barındırmak istemediğimizde kullanırız. Bu koleksiyon türünde veriler karma şekilde tutulur ve index numaralarına göre istediğimiz elemanı getirme gibi bir şansımız bulunmamaktadır. Bu karma depolama koleksiyonun performansını arttırmaktadır. Varsayılan kapasitesi 16'dır.

Java'da HashSet Tanımlama:

Tip bağımsız tanımlama: HashSet myHashSet = new HashSet();

Tip bağımsız tanımlama: HashSet <String> myHashSet = new HashSet<>();

Önemli HashSet Metodları: ArrayList sınıfında mevcut olan çoğu methodu barındırır. Ancak get methodu mevcut değildir.

Add Methodu: Parametre olarak girilen değeri ekler. Girilen değer HashSet içerisinde mevcutsa ekleme işlemini gerçekleştirmez.

myHashSet.Add("veri");

Remove Methodu: List yapılarından farklı olarak index üzerinden değil doğrudan değer üzerinden kullanılır. Silinmesi istenilen değer parametre olarak girilir. Eğer HashSet içerisinde mevcutsa silme işlemi gerçekleşir.

LinkedHashSet Sınıfı: Standart HashSet yapısından farklı olarak verileri belli bir düzende barındırır. Eğer verilerin sırası önemli değilse HashSet önemli ise önemli ise LinkedHashSet tercih edilebilir. HashSet'e oranla performansı daha düşüktür. HashSet ile aynı methodları barındırır.

Java'da LinkedHashMap Tanımlama:

Tip bağımsız tanımlama: LinkedHashSet myLinkedHashSet = new LinkedHashSet();

Tip bağımsız tanımlama: LinkedHashSet <String> myLinkedHashSet = new LinkedHashSet<>();

TreeSet Sınıfı: Set yapısına bağlı olduğu için tekrarlanan değerlere izin vermez. Genellikle büyük miktarlı verileri tutmak için kullanılır. Bu yapı null değer kabul etmez. İçerisine girilen değerleri default olarak küçükten büyüğe sıralı şekilde tutar. Doğal olarak eklenme sırası yapı içerisinde tutulmaz.

Java'da TreeSet Tanımlama:

Tip bağımsız tanımlama: TreeSet myTreeSet = new TreeSet();

Tip bağımsız tanımlama: TreeSet <String> myTreeSet = new TreeSet<>();

Önemli Methodlar:

Ceiling Methodu: Parametre olarak girilen değer yapıda mevcutsa onu döndürür değil ise en yakın en son değeri döndürür.

myTreeSet.Ceiling();

Comparator Methodu: Mevcut sıralama tipini öğrenmek veya değiştirmek için kullanılır. Eğer default sıralama mevcut ise null değeri döndürür.

myTreeSet.Comparator();

QUEUE ARAYÜZÜ: Verilerin kuyruk halinde tutulabilmesini sağlar. İlk atanan değerin yapıdan ilk ayrılmasını sağlar.

Queue Arayüzünü implemente eden sınıflar: (PriorityQueue)

PriorityQueue Sınıfı: İçerdiği elemanları bir öncelik sırasına göre sıralamaktadır. Örnek olarak integer veri tipinde bir PriorityQueue yapısı elemanlarını default olarak büyükten küçüğe olacak şekilde barındıracaktır.

Java'da PriorityQueue Tanımlama:

Tip bağımsız tanımlama: PriorityQueue myPriorityQueue = new PriorityQueue();

Tip bağımsız tanımlama: PriorityQueue<Integer> myPriorityQueue = new PriorityQueue<>();

Önemli methodlar:

Add ve Offer Methodları: Parametre olarak aldığı değeri PriorityQueue yapısına ekler.

myPriorityQueue.add(5);

myPriorityQueue.offer(7);

Peek Methodu: Yapıdaki ilk değeri verir. Yapı boş ise null değeri döner.

myPriorityQueue.peek();

Pool Methodu: Yapıdaki ilk değeri verir ve bu değeri siler. Boş ise null değeri döner.

myPriorityQueue.pool();

Comparator Methodu: Yapıyı sıralayacak olan comparatoru döner. Eğer default sıralama mevcut ise null değerini döner.

myPriorityQueue.comparator();

MAP ARAYÜZÜ: Bu kısımda benzerli/benzersiz veriler(value) benzersiz anahtar karşılıkları(key) ile beraber tutulur.

Map Arayüzünü implemente eden sınıflar: (HashTable,HashMap,LinkedHashMap,TreeMap)

HashTable Sınıfı: İçerisinde null key veya value tutulmasına izin vermez. Yapı içerisindeki değerlere anahtarları üzerinden ulaşılır. Veriler yapı içerisinde sıralı şekilde tutulmaz. Yapı içerisine benzersiz değerler ve anahtarlar eklenmesi gerekmektedir.

Java'da HashTable Tanımlama:

Tip bağımsız tanımlama: HashTable myHashTable = new HastTable();

Tip bağımsız tanımlama: HashTable<Integer,String> myHashTable = new HashTable <>();

Karo içerisindeki ilk tip key tipini, ikinci tip ise value tipini belirtmektedir.

Önemli HashTable Methodları:

Put Methodu: Yapımızın içerisine yeni bir anahtar-değer ilişkisi eklemeye yarar. İki parametre alır. İlk parametre anahtarı ikinci parametre ise değeri ifade eder.

myHashTable.put(1,"Şafak");

Get Methodu: Parametre olarak anahtar alır. Sistemde bu anahtar değeri mevcutsa anahtar ile ilişkili değeri döndürür. Eşleşen anahtar mevcut değil ise null değeri döndürür.

myHashTable.get(1);

Equals Methodu: Parametre olarak bir değer alır ve bu değerin yapı içerisinde eşitinin olup olmadığını kontrol eder.

myHashTable.put("Şafak");

Keys Methodu: HashTable içerisinde bulunan bütün anahtarları sıralar.

myHashTable.keys();

Contains & ContainsKey & ContainsValue Methodları:Verilen anahtar ve değere göre arama gerçekleştirerek eşleyen bir sonuçta true değerini döndürür.

myHashTable.contains("Şafak");

myHashTable.containsKey(1);

myHashTable.containsValue("Şafak");

HashCode Methodu: HashTable'ın hash kodunu döndürür.

myHashTable.hashCode();

HashMap Sınıfı: Yapı içerisine benzersiz anahtarlar eklenmelidir ancak benzer değerler eklenebilir. Veriler yapı içerisinde sıralı bir şekilde tutulmaz. Bu yapı içerisinde bir adet null anahtar ve birden fazla null değer tutulabilir. HashTable'a oranla daha hızlı bir yapıdır.

Java'da HashMap Tanımlama:

Tip bağımsız tanımlama: HashMap myHashMap = new HastMap();

Tip bağımsız tanımlama: HashMap<Integer,String> myHashMap = new HashMap <>();

Karo içerisindeki ilk tip key tipini, ikinci tip ise value tipini belirtmektedir.

HashTable ile aynı methodlara sahiptir.

Values Methodu: HashMap içerisinde bulunan değerli koleksiyon olarak geriye döndürür.

myHashMap.values();

LinkedHashMap Sınıfı: Standart HashMap yapısına oranla daha yavaştır. Verileri sıralı şekilde tutar.

Java'da LinkedHashMap Tanımlama:

Tip bağımsız tanımlama: LinkedHashMap myLinkedHash = new LinkedHashMap();

Tip bağımsız tanımlama: LinkedHashMap<Integer,String> myLinkedMap = new LinkedHashMap <>();

Karo içerisindeki ilk tip key tipini, ikinci tip ise value tipini belirtmektedir.

HashTable ile çoğunlukla aynı methodlara sahiptir.

TreeMap Sınıfı: Default olarak verileri key değerlerine göre küçükten büyüğe sıralı şekilde barındırırlar. Null veri girişine izin verir ancak null anahtar girişine izin vermez. Tekrarlı veri bulundurmaz.

Java'da LinkedHashMap Tanımlama:

Tip bağımsız tanımlama: TreeMap myTreeMap = new TreeMap);

Tip bağımsız tanımlama: TreeMap<Integer,String> myTreeMap = new TreeMap <>();

Karo içerisindeki ilk tip key tipini, ikinci tip ise value tipini belirtmektedir.

HashTable'da mevcut olan methodları barındırır.

Önemli Methodlar:

CeilingEntry & CeilingKey Methodları: Verilen anahtar değerine eşit veya büyük olan en küçük değeri döndürür.

myTreeMap.ceilingEntry(1);

myTreeMap.ceilingKey(1);

FirstEntry & FirstKey Methodları: En küçük anahtar ve eşli değerini döndürür.

myTreeMap.FirstEntry();

myTreeMap.FirstKey();

LastEntry & LastKey Methodları: En büyük anahtar ve eşli değerini döndürür.

myTreeMap.LastEntry();

myTreeMap.LastKey();

Values Methodu: Yapı içerisindeki anahtarlara karşılık gelen değerleri bir koleksiyon halinde döner.

myTreeMap.Values();