BAZI PROGRAMLAMA DİLLERİNDEKİ DOĞRUDAN KULLANILABİLECEK VERİ YAPILARI

DERS 11

21.12.2022

İncelenen Diller

- Ruby
- Java
- Python

Ruby Programlama Dili Veri Yapıları

- Yığın
- Kuyruk
- Bağlı Liste
- Hash

Ruby Programlama Dili Veri Yapıları

- Bir dizi, bazı fonksiyonlar ile bir yığın veya kuyruk veri yapısı şeklinde kullanılabilmektedir. (push, pop, shift, unshift)

  -push / pop → FILO

  -unshift / pop → FIFO

- Bağlı listeler

  -Sınıf kullanarak bağlantılı bir liste uygulanabilir

  -Standart dizi yöntemlerini kullanarak bağlantılı liste benzeri davranışlar verecektir.

- Maps

  -Hash {}

Ruby Programlama Dili Veri Yapıları Örnekler

- Yığın (push / pop)

```ruby
a = []
puts a.empty? #=> true
a.push 5
a.push 7
a.push 8
a.push 9
p a #=> [5, 7, 8, 9]
p a.pop #=> 9
p a.pop #=> 8
p a #=> [5, 7]
puts a.empty? #=> false
```

Ruby Programlama Dili Veri Yapıları Örnekler

- Kuyruk (unshift / pop)

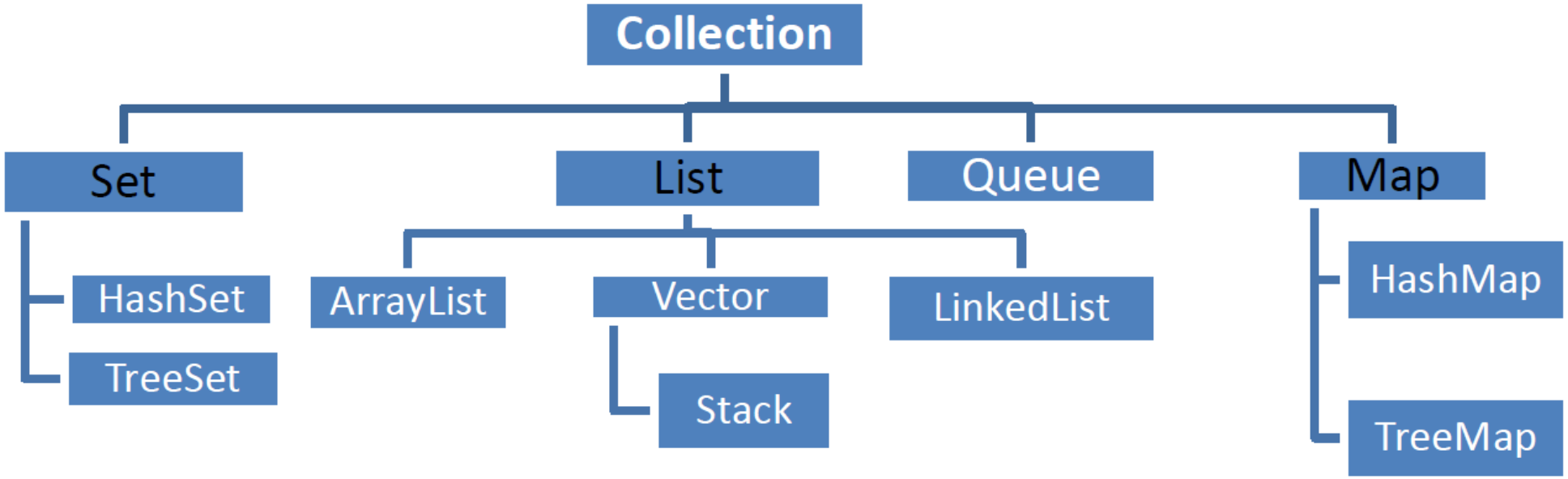
```ruby
a = []
puts a.empty? #=> true
a.unshift 5
a.unshift 7
a.unshift 8
a.unshift 9
p a #=> [9, 8, 7, 5]
p a.pop #=> 5
p a.pop #=> 7
p a #=> [9, 8]
puts a.empty? #=> false
```

Java Programlama Dili Veri Yapıları

- Set
- List
- Kuyruk
- Dinamik Diziler
- Vector
- ArrayList

Java Collection

- Collection arabirimi, içerisinde koleksiyon elemanları olarak bilinen bir grup nesne barındırır. Veriyi saklamak, geri almak, güncellemek ve bir yöntemden diğerine veri aktarmak amaçlarıyla kullanılır.



Java Collection

- **Set:** Tekrarlı ya da kopya elemanları içerisinde barındırmayan bir koleksiyondur.

- **List(Liste):** Sıralı elemanları içerisinde barındıran bir koleksiyondur. İçerisinde tekrarlı elemanlar olabilir.

- **Map (Eşlem):** Birbirinden farklı anahtarlar ile eşleştirilen nesnelerden oluşur. Bir Map içerisinde tekrarlı anahtar değeri bulunmaz. (Ruby programlama dilindeki Hash gibi)

Java Collection

- Java dilinde bağlı liste (LinkedList), kuyruk ve yığın yapılarının hepsi **Collection** sınıfından türetilmiştir.
- **Collection** sınıfı birden çok elemanı aynı grupta toplayan ara yüzdür.
- Verileri saklamak, geri almak, güncellemek ve bir yöntemden diğerine veri aktarmak amaçlarıyla kullanılır.

Set

- Tekrarlı ya da kopya elemanları içerisinde barındırmayan bir koleksiyondur.
- Elemanlar sırasız bir şekilde tutulur.
- Basit ekleme ve silme yöntemleri vardır.
- Tekrarlanan elemanlara izin verilmez.
- Bir setten bir nesneyi silmeden önce, nesnenin orada var olduğunu bilmeniz gerekir.

Set

- **HashSet:** Hızlı bir kümeleme yapar. Saklama ve geri çağırma için HashMap kullanır. Kullanım Şekli;

```
HashSet <String> diziAdi = new HashSet <String>( );
```

- **TreeSet:** Ağaç yapısını kullanarak kümeleme yapar. TreeMap kullanarak nesneleri artan sırada dizer. Kullanım şekli;

```
TreeSet <String> diziAdi= new TreeSet <String>( );
```

Set

```java
import java.util.*;

public class Main {
public static void main(String[] args) {
    HashSet<String> hs = new HashSet<String>();
    hs.add("Java");
    hs.add("Ruby");
    hs.add("Python");
    hs.add("Ruby"); //çift veriye izin verilmez
    hs.add("C");
    hs.add("C++");
    System.out.println("Dillerin Sırasız hali");
    System.out.println(hs);
    System.out.println("Dillerin Sıralı hali");
    TreeSet<String> sirali = new TreeSet<String> (hs);
    System.out.println(sirali);
  }
}
```

```
Dillerin Sırasız hali
[Java, C++, C, Ruby, Python]
Dillerin Sıralı hali
[C, C++, Java, Python, Ruby]
```

List

- Sıralı elemanları içerisinde barındıran bir koleksiyondur.
- İçerisinde tekrarlı elemanlar olabilir.
- Java dilinde veri yapıları algoritmaları için özel sınıflar vardır. Yığın(stack), dinamik diziler(ArrayList, Vector) ve Bağlı listeler(LinkedList) gibi.
- Bu sınıfların tamamı List sınıfından türetilmiştir.
- List sınıfının temel yöntemleri: ekleme, silme ve ters çevirme

LinkedList (Bağlı Liste)

List sınıfı ve alt sınıflara (ArrayList, LinkedList) ait bazı yöntemler;	
add	Listeye eleman ekler
addAll (Collection c)	Parametrede verilen koleksiyonun bütün öğelerini listenin sonuna ekler
clear()	Listedeki tüm elemanları siler
get (indis)	İndisi belirtilen öğeyi listeden seçer
set(i, b)	v elemanını i indisli yere yerleştirir.

LinkedList (Bağlı Liste)

indexOf()	**Aranan elemanın indisini döndürür, eleman yoksa -1değerini döndürür**
lastIndexOf()	**Aranan elemanın son indisini döndürür, eleman yoksa -1 değerini döndürür**
listIterator ()	**Bir işaretçi(pointer) mantığı ile listedeki elemanlar üzerinde işlem yapar.**
remove (int indis)	**İndis numarası verilen elemanı listeden kaldırır.**
size()	**Listedeki eleman sayısını verir**
toArray()	**Listedeki elemanları dizi elemanlarına dönüştürür.**

LinkedList Üzerinde Gezinme

- **ListIterator,** elemanlar arası istenilen yönde ilerlemek için kullanılır.
- Özellikle LinkedList içindeki elemanları sıralı olarak işlemek için kullanılır.
- Listedeki elemanları sıralamak için **Collections.sort()**, ters sırada listelemek içinse **Collections.reverse()** metotları kullanılır.
- Üç temel metodu vardır:

    1) next() →Bir sonraki elemanı çağırır

    2) hasNext() → Bir sonraki eleman olup olmadığı kontrol edilir

    3) remove() → next() metodu ile döndürülen son elemanı siler

LinkedList Örnek

```java
import java.util.*;
public class Main {
  public static void main(String[] args) {
    LinkedList<String> diller = new LinkedList <String>();
    diller.add("Python");
    diller.add("Lua");
    diller.add("R");
    diller.add("Ruby");
    diller.add("C");
    diller.add("C#");
    System.out.println("Liste içindekiler " + diller);
    diller.add(4, "Julia");
    System.out.println("Değişen liste =" + diller);
    System.out.println("ilk eleman=" + diller.getFirst());
    System.out.println("5.nci eleman=" + diller.get(5));
    System.out.println("silinen=" + diller.removeFirst());
    System.out.println("silinen=" + diller.removeLast());
    System.out.println("Liste son hali =" + diller);
  }
}
```

```
Liste içindekiler [Python, Lua, R, Ruby, C, C#]
Değişen liste =[Python, Lua, R, Ruby, Julia, C, C#]
ilk eleman=Python
5.nci eleman=C
silinen=Python
silinen=C#
Liste son hali =[Lua, R, Ruby, Julia, C]
```

LinkedList Örnek

```java
import java.util.*;
public class Main {
  public static void main(String[] args) {
    LinkedList<String> diller = new LinkedList <String>();
    diller.add("Python");
    diller.add("Lua");
    diller.add("R");
    diller.add("Ruby");
    diller.add("C");
    diller.add("C#");
    System.out.println("Liste içindekiler " + diller);
    diller.add(4, "Julia");
    System.out.println("Değişen liste =" + diller);
    System.out.println("ilk eleman=" + diller.getFirst());
    System.out.println("5.nci eleman=" + diller.get(5));
    System.out.println("silinen=" + diller.removeFirst());
    System.out.println("silinen=" + diller.removeLast());
    System.out.println("Liste son hali =" + diller);
    ListIterator<String> gez = diller.listIterator();
    while(gez.hasNext()) {
      System.out.println(gez.next());
    }
  }
}
```

```
Liste içindekiler [Python, Lua, R, Ruby, C, C#]
Değişen liste =[Python, Lua, R, Ruby, Julia, C, C#]
ilk eleman=Python
5.nci eleman=C
silinen=Python
silinen=C#
Liste son hali =[Lua, R, Ruby, Julia, C]
Lua
R
Ruby
Julia
C
```

Queue

Queue Sınıfının Metotları	
element ( )	Kuyruğun önündeki(başındaki) nesneyi döndürür (fakat silmez)
offer ( )	Yeni bir elemanı kuyruk sonuna ekler
peek ( )	Kuyruğun önündeki(başındaki) elemanı döndürür fakat silmez. Eğer kuyruk boş ise **null** değerini döndürür.
poll ( )	Kuyruğun önündeki(başındaki) elemanı döndürür ve siler. Eğer kuyruk boş ise **null** değerini döndürür.
remove ( )	Kuyruk önündeki nesneyi kuyruktan siler ve nesneyi döndürür
size ( )	Kuyruktaki eleman sayısını döndürür. (Öncelik kuyruğu için)
clear ( )	Kuyruktaki tüm elemanları siler. (Öncelik kuyruğu için)

Queue Örnek

```java
import java.util.*;
public class Main {
  public static void main(String[] args) {
    Queue <String> kuyruk = new LinkedList<String>();
    kuyruk.offer("A");
    kuyruk.offer("B");
    kuyruk.offer("C");
    kuyruk.offer("D");
    System.out.println("Kuyrukta bekleyenler: " + kuyruk);
    kuyruk.poll();
    System.out.println("Kuyrukta bekleyenler: " + kuyruk);
    System.out.println("Kuyruktan çıkartılıyor:");
    while (!kuyruk.isEmpty())
      System.out.print(kuyruk.remove() + " ");
  }
}
```

```
Kuyrukta bekleyenler: [A, B, C, D]
Kuyrukta bekleyenler: [B, C, D]
Kuyruktan çıkartılıyor:
B C D >
```

Stack

- **push(değer) :** Parametre olarak alınan değeri stack içine atar.
- **pop() :** Stack'te en üstteki elemanı döndürür ve stack içerisinden siler.
- **peek() :** Stack'te en üstteki elemanı döndürür. (Stack içinden silinmez.)
- **size() :** Stack eleman sayısını döndürür.
- **isEmpty() :** Stack boş ise TRUE, dolu ise FALSE döndürür.

Stack Örnek

```java
import java.util.*;
public class Main {
  public static void main(String[] args) {
    Stack<Integer> yigin = new Stack<Integer>();
    yigin.push(13);
    yigin.push(120);
    yigin.push(6);
    yigin.push(90);
    System.out.println("Yığın durumu : " + yigin);
    System.out.println("Yığının en üstü :" + yigin.peek());
    System.out.println("Yığın durumu : " + yigin);
    System.out.println("Bir eleman çıkar:" + yigin.pop());
    System.out.println("Yığın durumu : " + yigin);
    System.out.println("Yığından çıkatılıyor:");
    while(!yigin.isEmpty())
      System.out.print(yigin.pop() + " ");
  }
}
```

```
Yığın durumu : [13, 120, 6, 90]
Yığının en üstü :90
Yığın durumu : [13, 120, 6, 90]
Bir eleman çıkar:90
Yığın durumu : [13, 120, 6]
Yığından çıkatılıyor:
6 120 13 :
```

Python Programlama Dili Veri Yapıları

- tuple: ekleme ve çıkarmanın yapılamayacağı veri yapısıdır
- set: içinde eşsiz elemanlar bulunduran veri yapısıdır
- list: Ruby dilindeki dizilere benzer
- dict: Ruby dilindeki hash yapısına benzer
- stack
- queue

tuple Örnek

```
1  x = ('Samsun', 55, 34, "İstanbul")
2  print(type(x))
3  print(x)
4  print(x[1])
5  print(len(x))
6  #x.append(5) hata üretir bu satır
```

```
<class 'tuple'>
('Samsun', 55, 34, 'İstanbul')
55
4
```

set Örnek

```
1  a = {1, 2, 3}
2  print(type(a))
3  print(a)
4  a.add(10)
5  print(a)
6  a.add(10)
7  print(a)
```

```
<class 'set'>
{1, 2, 3}
{10, 1, 2, 3}
{10, 1, 2, 3}
```

Stack Örnek-1

```
1  stack = []
2  stack.append('a')
3  stack.append('b')
4  stack.append('c')
5  print('Mevcut yığın:')
6  print(stack)
7
8  print('Elemanlar çıkartılıyor')
9  print(stack.pop())
10 print(stack.pop())
11 print(stack.pop())
12
13 print('\nYığın son durum:')
14 print(stack)
```

```
Mevcut yığın:
['a', 'b', 'c']
Elemanlar çıkartılıyor
c
b
a

Yığın son durum:
[]
```

Stack Örnek-2

```python
from collections import deque

stack = deque()
stack.append('a')
stack.append('b')
stack.append('c')

print('Mevcut yığın:')
print(stack)

print('Elemanlar çıkartılıyor')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nYığın son durum:')
print(stack)
```

```
Mevcut yığın:
deque(['a', 'b', 'c'])
Elemanlar çıkartılıyor
c
b
a

Yığın son durum:
deque([])
```

Queue Örnek-1

```
1  queue = []
2  queue.append('a')
3  queue.append('b')
4  queue.append('c')
5
6  print("Kuyruk mevcut durum:")
7  print(queue)
8
9
10 print("Elemanlar çıkartılıyor")
11 print(queue.pop(0))
12 print(queue.pop(0))
13 print(queue.pop(0))
14
15 print("Kuyruk son durum:")
16 print(queue)
```

```
Kuyruk mevcut durum:
['a', 'b', 'c']
Elemanlar çıkartılıyor
a
b
c
Kuyruk son durum:
[]
```

Queue Örnek-2

```python
from collections import deque

q = deque()
q.append('a')
q.append('b')
q.append('c')

print("Mevcut kuyruk")
print(q)

print("Elemanlar çıkartılıyor")
print(q.popleft())
print(q.popleft())
print(q.popleft())

print("Kuyruk son durum:")
print(q)
```

```
Mevcut kuyruk
deque(['a', 'b', 'c'])
Elemanlar çıkartılıyor
a
b
c
Kuyruk son durum:
deque([])
```