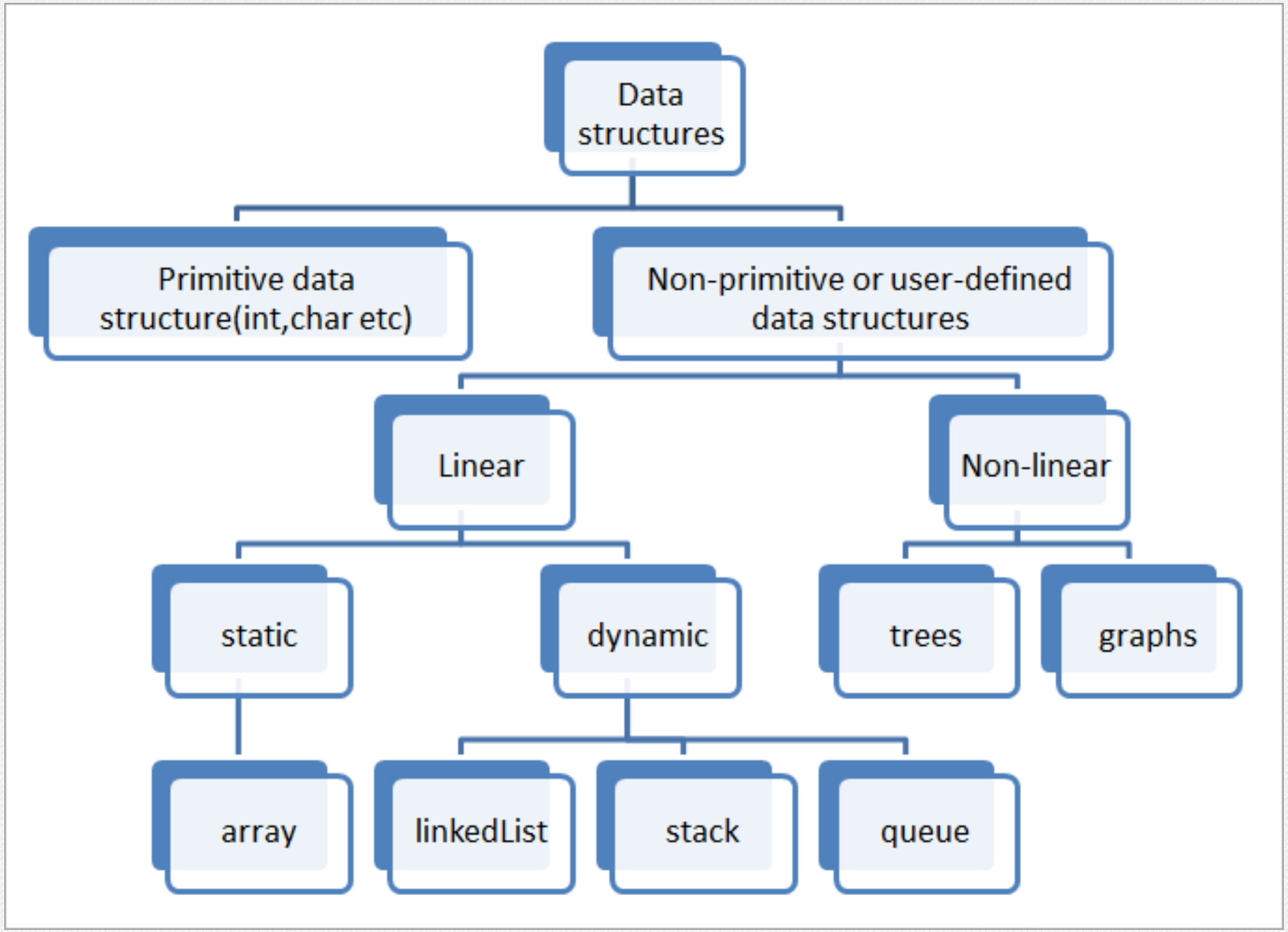
«Ծրագրավորում Java» դասընթաց

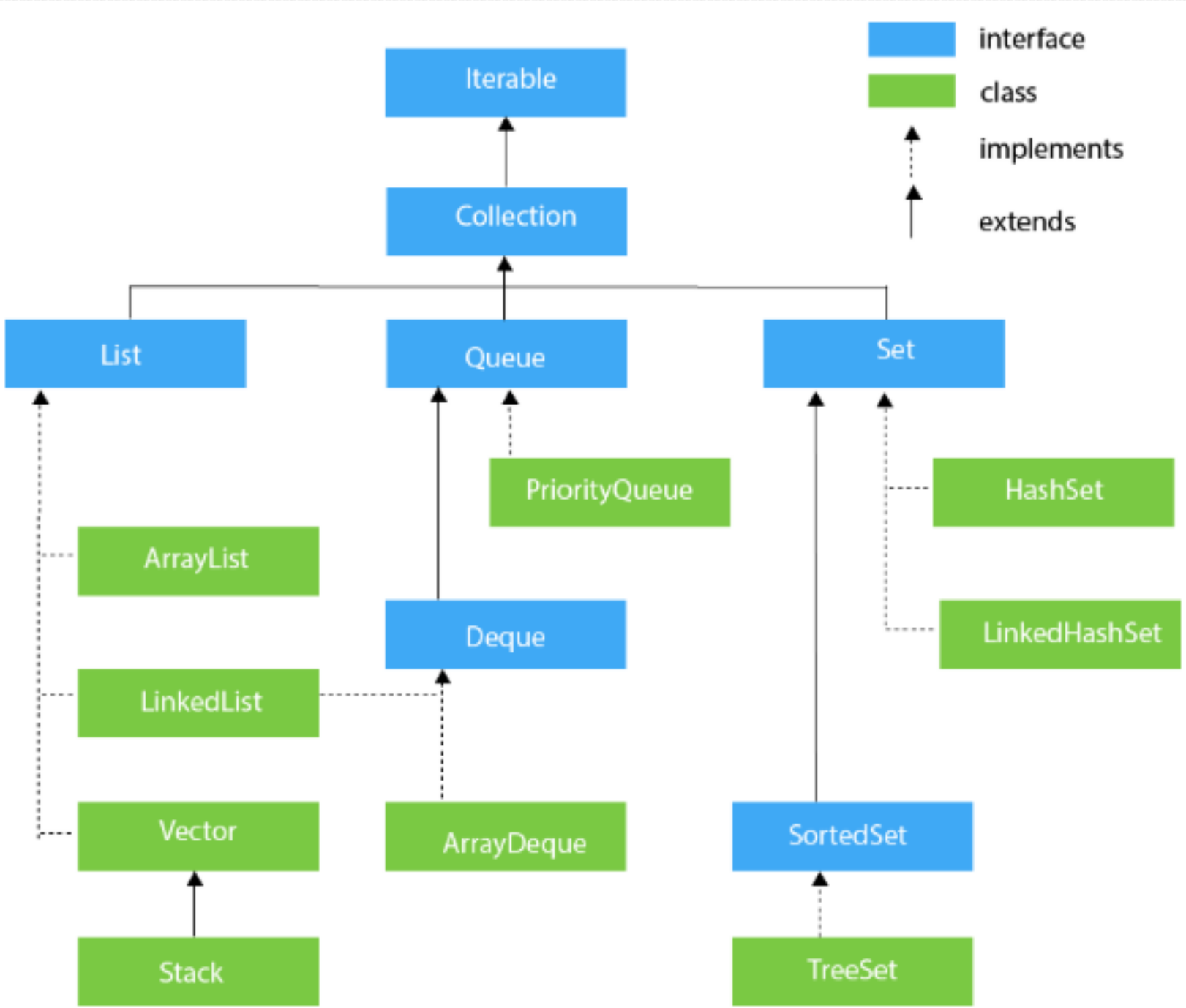
22.Collection

- Collection-ների իրականացումը տվյալների կառուցվածքներով (data structure)
- Collection Framework
- Collection ինտերֆեյսի մեթոդները


Data structures
Primitive data structure(int,char etc)
Non-primitive or user-defined data structures
Linear
Non-linear
static
dynamic
trees
graphs
array
linkedList
stack
queue

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

Реализации списков:

	get	add	contains	next	remove(0)	iterator.remove
ArrayList	O(1)	O(1)	O(n)	O(1)	O(n)	O(n)
LinkedList	O(n)	O(1)	O(n)	O(1)	O(1)	O(1)
CopyOnWrite-ArrayList	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)

Установить реализации:

	add	contains	next	notes
HashSet	O(1)	O(1)	O(h/n)	h is the table capacity
LinkedHashSet	O(1)	O(1)	O(1)	
CopyOnWriteArraySet	O(n)	O(n)	O(1)	
EnumSet	O(1)	O(1)	O(1)	
TreeSet	O(log n)	O(log n)	O(log n)	
ConcurrentSkipListSet	O(log n)	O(log n)	O(1)	

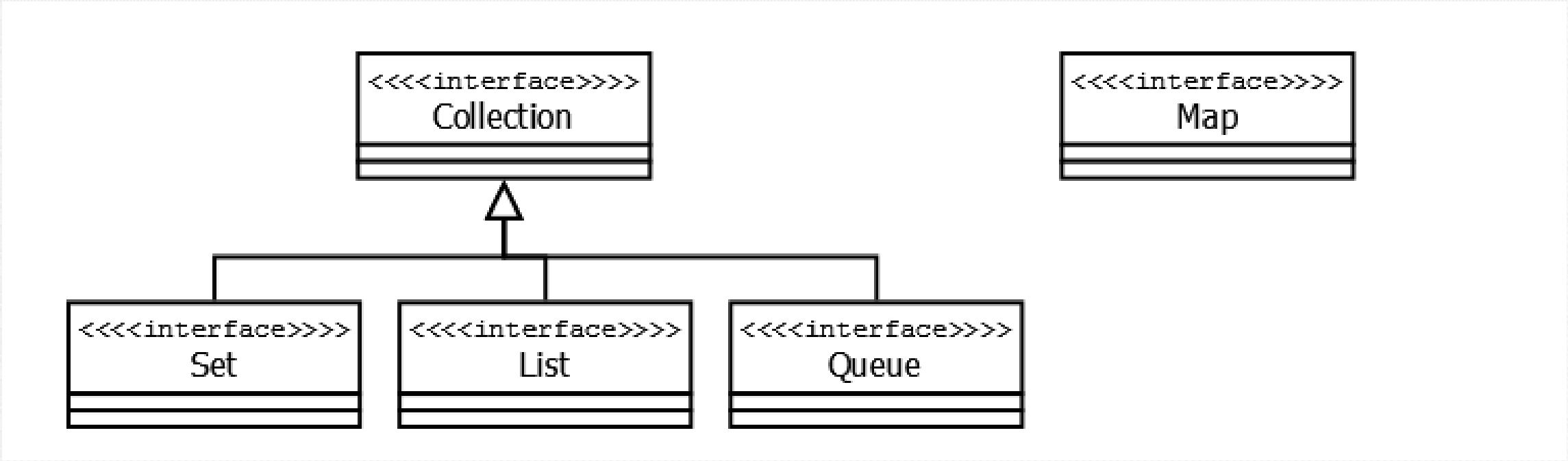
Реализации карт:

	get	containsKey	next	Notes
HashMap	O(1)	O(1)	O(h/n)	h is the table capacity
LinkedHashMap	O(1)	O(1)	O(1)	
IdentityHashMap	O(1)	O(1)	O(h/n)	h is the table capacity
EnumMap	O(1)	O(1)	O(1)	
TreeMap	O(log n)	O(log n)	O(log n)	
ConcurrentHashMap	O(1)	O(1)	O(h/n)	h is the table capacity
ConcurrentSkipListMap	O(log n)	O(log n)	O(1)	

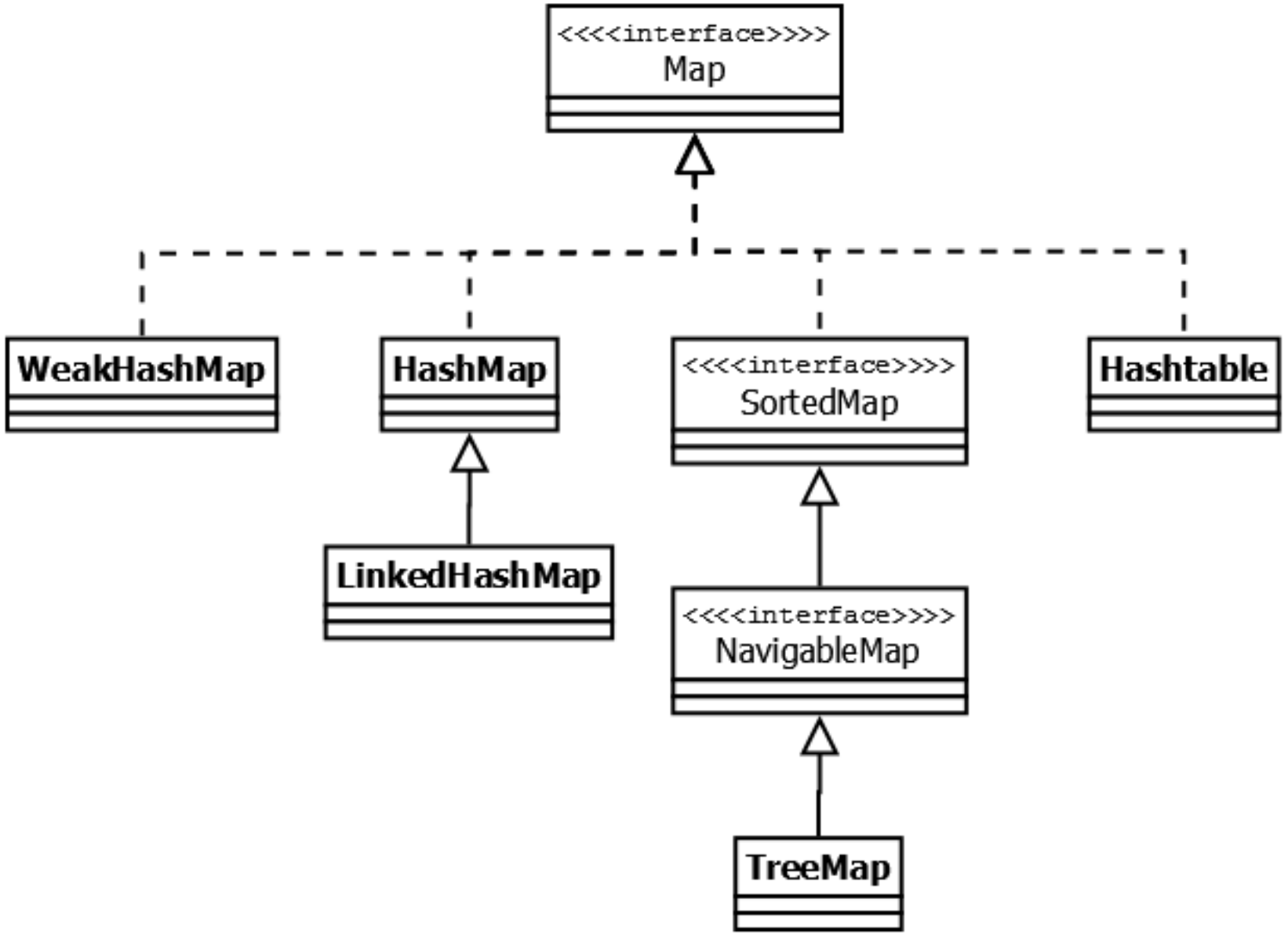
Реализации очереди:

	offer	peek	poll	size
PriorityQueue	O(log n)	O(1)	O(log n)	O(1)
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(1)
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(1)
PriorityBlockingQueue	O(log n)	O(1)	O(log n)	O(1)
DelayQueue	O(log n)	O(1)	O(log n)	O(1)
LinkedList	O(1)	O(1)	O(1)	O(1)
ArrayDeque	O(1)	O(1)	O(1)	O(1)
LinkedBlockingDeque	O(1)	O(1)	O(1)	O(1)

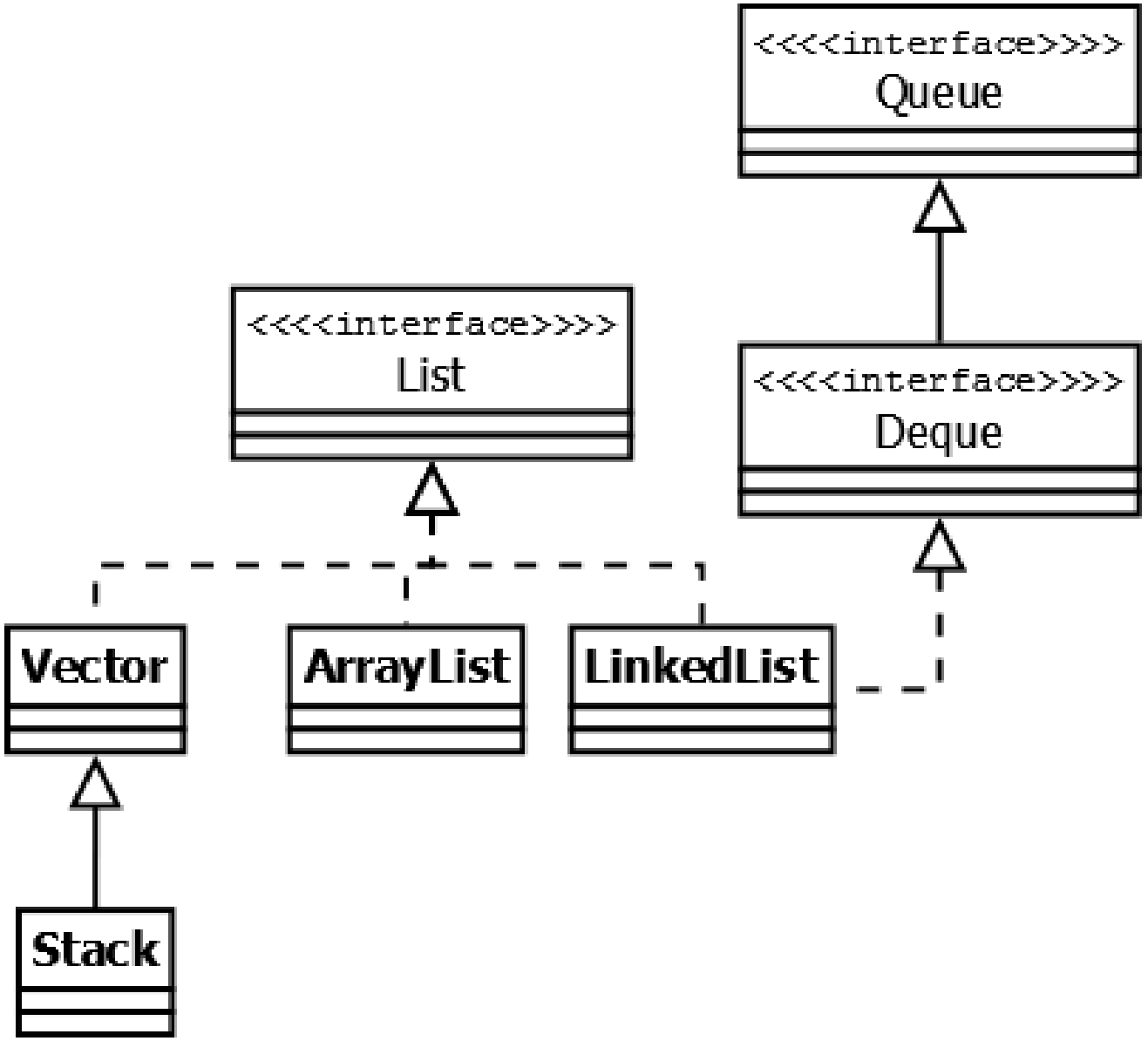
https://overcoder.net/q/35124/%D1%80%D0%B5%D0%B7%D1%8E%D0%BC%D0%B5-big-o-%D0%B4%D0%BB%D1%8F-%D1%80%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B9-java-collections-framework


<<<<interface>>>>
Collection
<<<<interface>>>>
Map
<<<<interface>>>>
Set
<<<<interface>>>>
List
<<<<interface>>>>
Queue

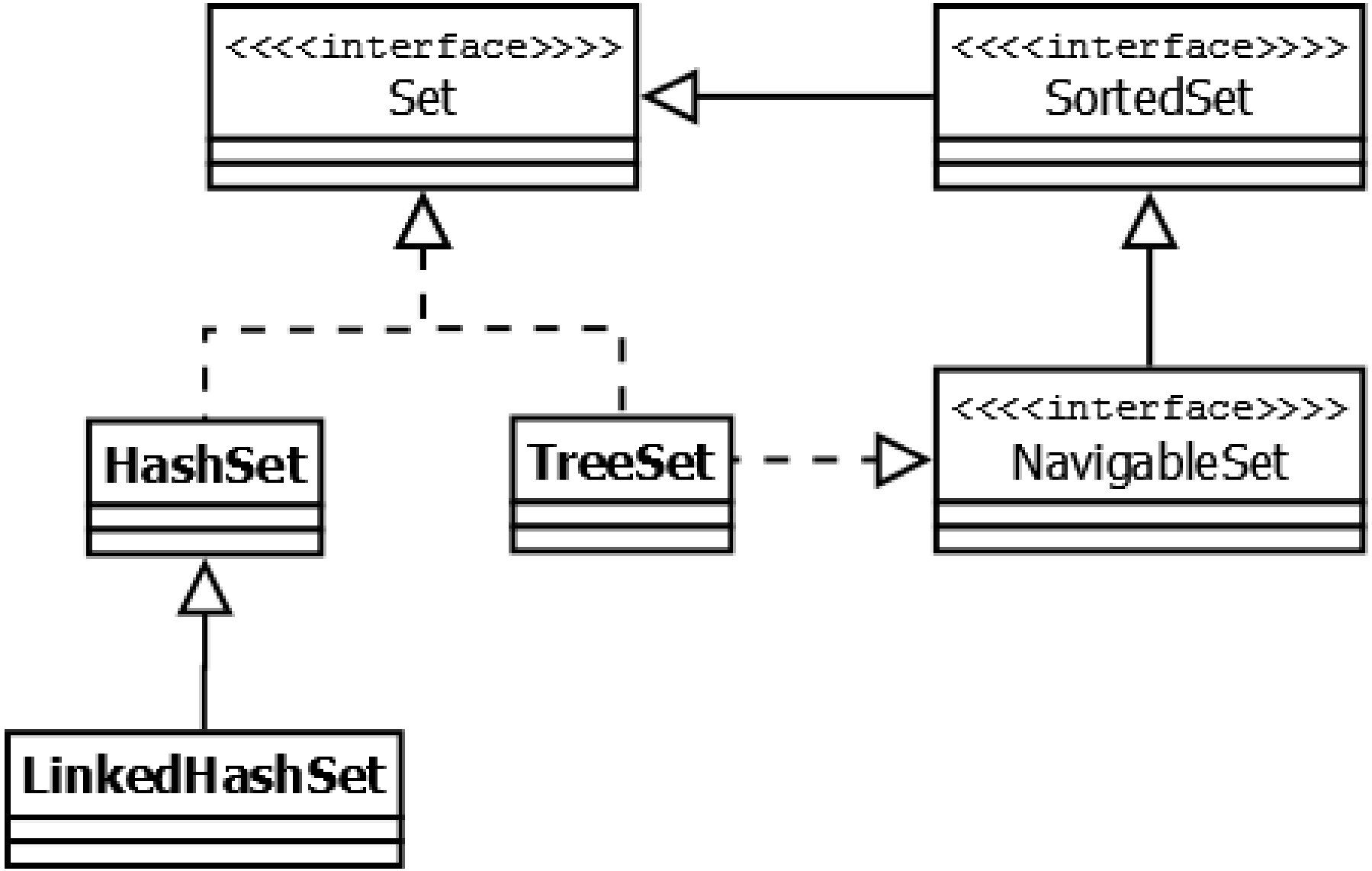
Интерфейс Map [doc]



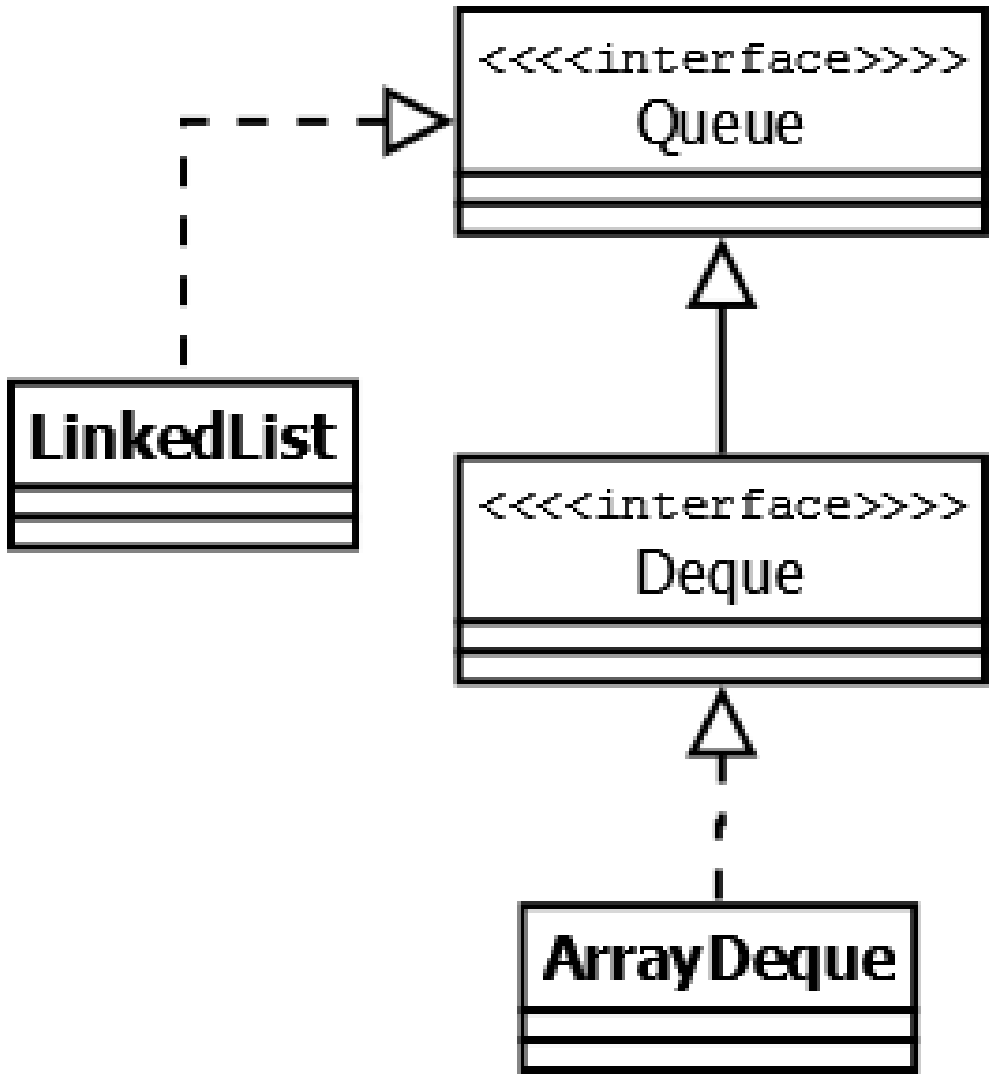
Интерфейс List [doc]



Интерфейс Set [doc]



Интерфейс Queue [doc]



Data Structures Comparison

Data Structures	Average Case			Worst Case		
	Search	Insert	Delete	Search	Insert	Delete
Array	O(n)	N/A	N/A	O(n)	N/A	N/A
Sorted Array	O(log n)	O(n)	O(n)	O(log n)	O(n)	O(n)
Linked List	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)
Doubly Linked List	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)
Stack	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)
Hash table	O(1)	O(1)	O(1)	O(n)	O(n)	O(n)
Binary Search Tree	O(log n)	O(log n)	O(log n)	O(n)	O(n)	O(n)
B-Tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
Red-Black tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
AVL Tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)

mpl	ADT	Data Structure	Performance (Big O notation)
ArrayList (sync)	List	Array of objects. A new array is created and populated whenever elements are added beyond the current length (capacity) of the underlying array.	**add(E element) method:** O(1) amortized. That is, adding n elements within capacity: constant time O(1). Adding an element beyond capacity: O(n) times. It's better to specify initial capacity at construction if known. **remove(int index)**: O(n - index), removing last is O(1). **All other operations including get(int index)** run in linear time O(1) . The constant factor of O(1) is low compared to that for the LinkedList implementation.
LinkedList (sync)	List, Deque	Doubly-linked list. Each element has memory addresses of the previous and next item used internally.	**get(int index), remove(int index)**: O(n) **add(E element) and others**: Constant time O(1).
Vector (sync) (Legacy)	List	Array of objects. Similar to ArrayList	Similar to ArrayList but slower because of synchronization.
Stack extends Vector (sync) (Legacy)	List	Array of objects. LIFO (Last in first out). It provides addition methods empty(), peek(), pop(), push(E e) and search(Object o)	Similar to Vector/ArrayList but slower because of synchronisation.

HashSet (sync)	Set	Backed by HashMap (a Hash table data structure). Elements of the set are populated as key of the HashMap. Allows at most one null.	**add, remove, contains, size**: O(1) **Iteration**: O(n + capacity). Better don't set initial capacity (size of backing hasMap) too high or load factor too low if iteration is frequently used.
LinkedHash Set (sync)	Set	Backed by LinkedHashMap where elements of this LinkedHashSet are populated as key of the Map. Maintains elements in insertion order. Allows at most one null.	**add, remove, contains, size**: O(1) **Iteration**: O(n), slightly slow that of HashSet, due to maintaining the linked list.
TreeSet (sync)	NavigableSet	Backed by TreeMap (a red-black tree data structure). The elements of this set are populated as key of the Map. Doesn't permit null.	**add, remove, contains**: O(log n) **Iteration**: O(n) slower than HashSet.
EnumSet (sync)	Set	Bit vectors All of the elements must come from a single enum type.	**All methods**: O(1). Very efficient
PriorityQue ue (sync)	Queue	Binary Heap Unbounded Elements are ordered to their natural ordering or by a provided Comparator.	**offer, poll, remove() and add**: O(log n) **remove(Object), contains(Object)** O(n) **peek, element, and size**: O(1)
ArrayDeque (sync)	Dequeue	Resizable-array (similar to ArrayList). Unbounded Nulls not permitted.	**remove, removeFirstOccurrence, removeLastOccurrence, contains, iterator.remove(), and the bulk operations**: O(n)

Comparison of Selected Interfaces					
Class or Interface	Superinterfaces	Notes	Allow Duplicates?	Allow null?	Retrieval
Collection	Iterable	The parent interface of the entire collections framework.	Depends	Depends	Through iterator
List	Collection	Allows you to put elements in a specific order. Each element is associated with an index.	Yes	Depends	Through iterator, or Random access by numerical index
Set	Collection	Prohibits duplicate elements. Easy test for membership. You can't order the elements.	No	Depends	Through iterator
Queue	Collection	Only the "head" (next out) is available.	Yes	Depends	Peeking or polling, or Through iterator
PriorityQueue	Queue	High priority elements get to the head first.	Yes	No	Peeking or polling
Deque	Queue	A "double-ended queue": insert and remove at both ends.	Yes	Depends	Peeking or polling, or Through iterator
Map	n/a	Maps keys to values	Keys: no Values: yes	Keys: (only 1 null key is allowed) Values: yes	Lookup by key object, or Through keySet, entrySet, values "views"

	Временная сложность							
	Среднее				*Худшее*			
	Индекс	Поиск	Вставка	Удаление	Индекс	Поиск	Вставка	Удаление
ArrayList	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Vector	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
LinkedList	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Hashtable	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
HashMap	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
LinkedHashMap	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
TreeMap	n/a	O(log(n))	O(log(n))	O(log(n))	n/a	O(log(n))	O(log(n))	O(log(n))
HashSet	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
LinkedHashSet	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
TreeSet	n/a	O(log(n))	O(log(n))	O(log(n))	n/a	O(log(n))	O(log(n))	O(log(n))

23. List ինտերֆեյս

- *ArrayList collection, ներդրված մեթոդները:*
- *LinkedList collection, ներդրված մեթոդները:*
- *ArrayList և LinkedList համեմատություն:*
- *List և ListIterator ինտերֆեյսներ:*

Comparison of Lists		
	ArrayList	**LinkedList**
Description	Dynamic (a.k.a. growable, resizable) array	Doubly-linked list
Random Access with index (get())	O(1)	O(n)
Insertion (add()) / removal at the back	amortized O(1)	O(1)
Insertion / removal at the front	O(n)	O(1)
One step of iteration through an Iterator	O(1)	O(1)
Insertion / removal in the middle through an Iterator / ListIterator	O(n)	O(1)
Insertion / removal in the middle through the index	O(n)	O(n)
Search contains() / removal remove() by object	O(n)	O(n)

- https://www.javatpoint.com/java-arraylist
- https://www.javatpoint.com/java-linkedlist
- https://www.javatpoint.com/difference-between-arraylist-and-linkedlist

Աղյուսակ-HashTable

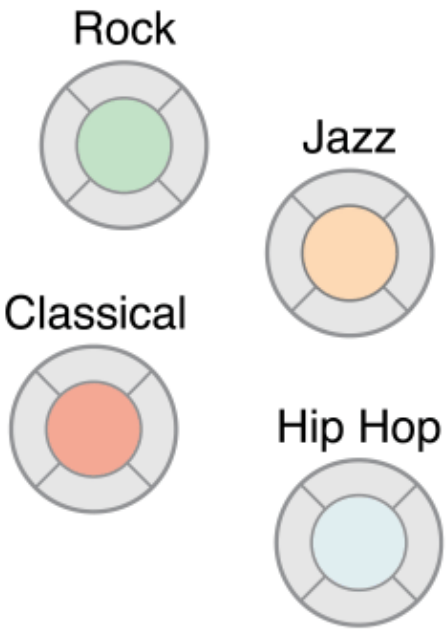
- *HashTable տվյալների կառուցվածք:*
- *Հեշավորման ֆունկցիա:*
- *Բախումների (collisions) վերացման մեթոդներ՝ linear prob, separate chaining:*
- *HashTable կլաս:*

Array

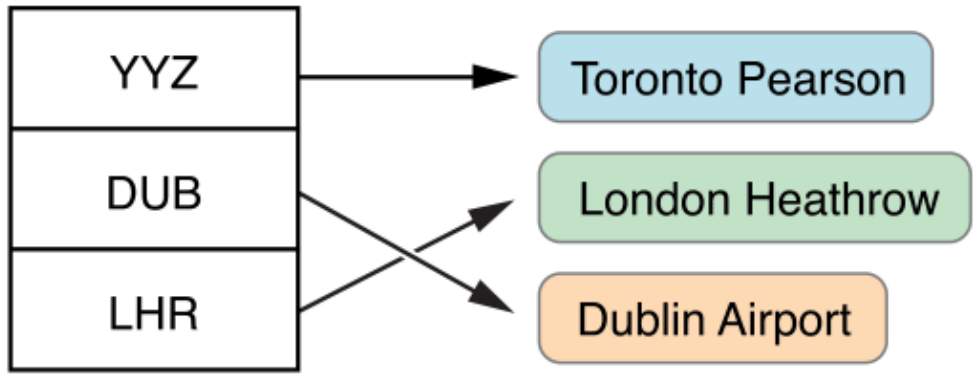
Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

Set

Values



Dictionary



Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Hashing

Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.

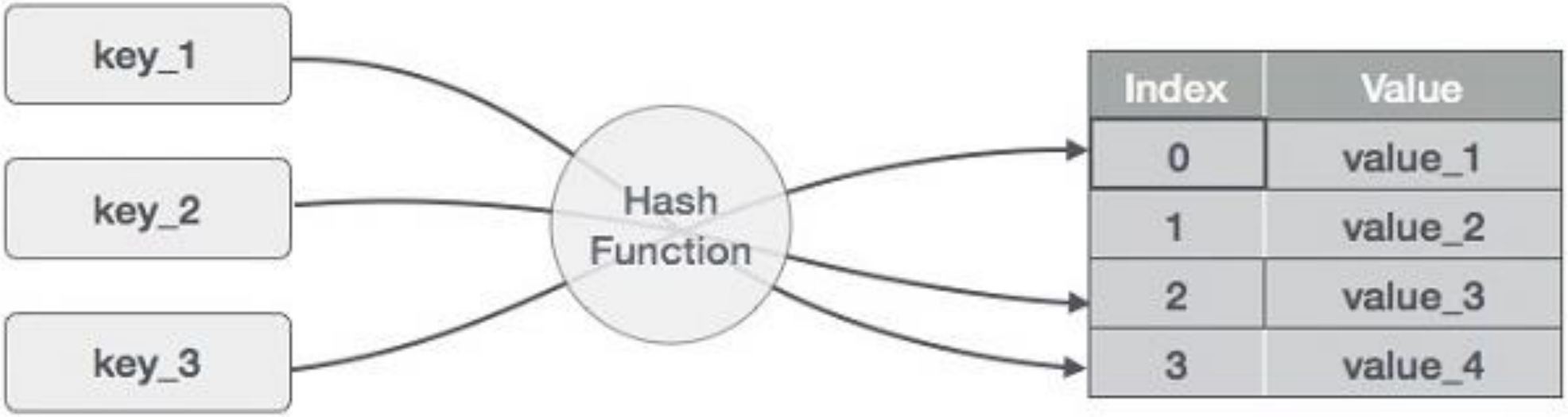
https://visualgo.net/bn/hashtable

https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm

https://www.hackerearth.com/ru/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

https://www.geeksforgeeks.org/hashing-set-1-introduction/

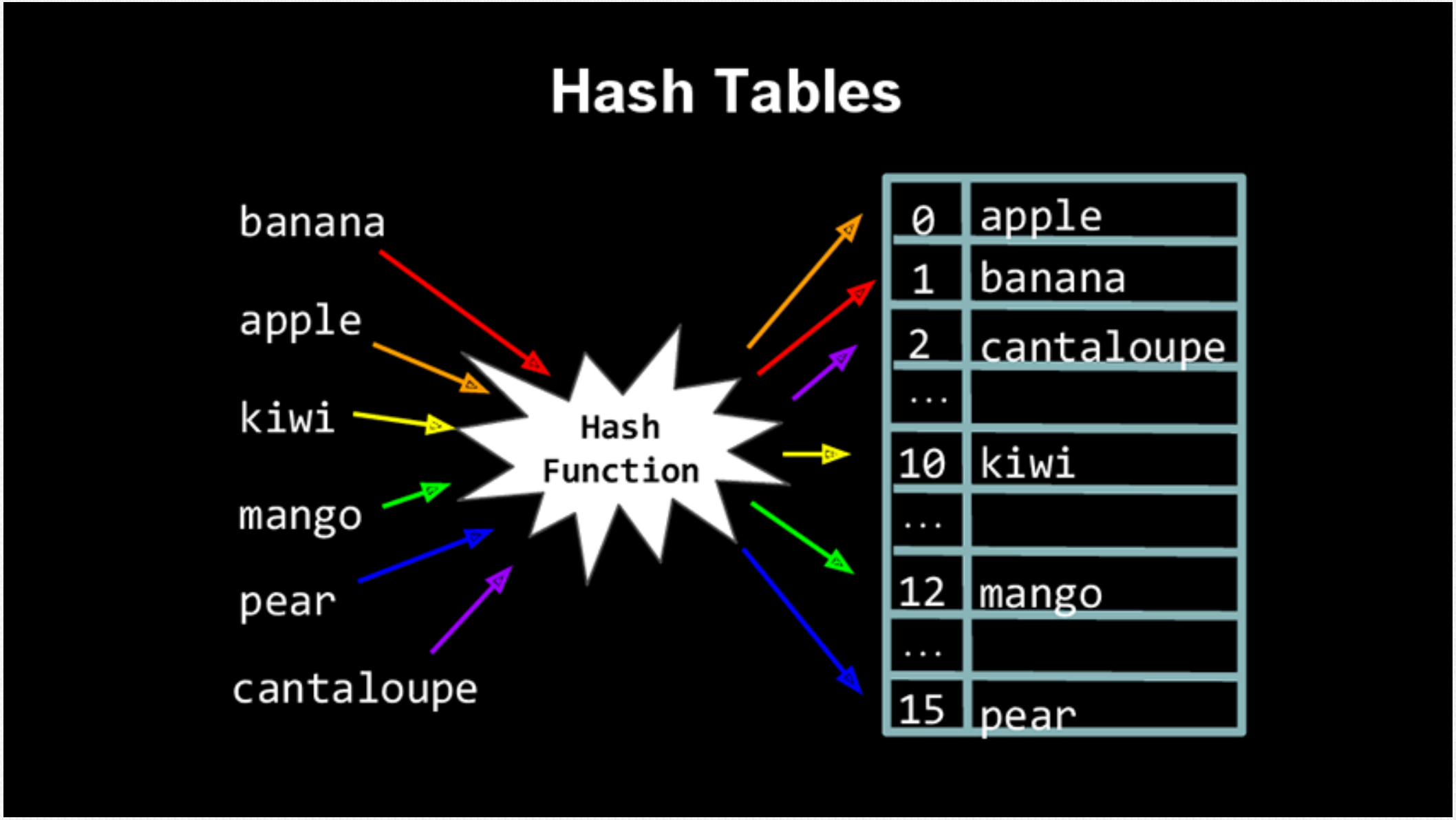
https://javarush.ru/quests/lectures/questharvardcs50.level05.lecture06

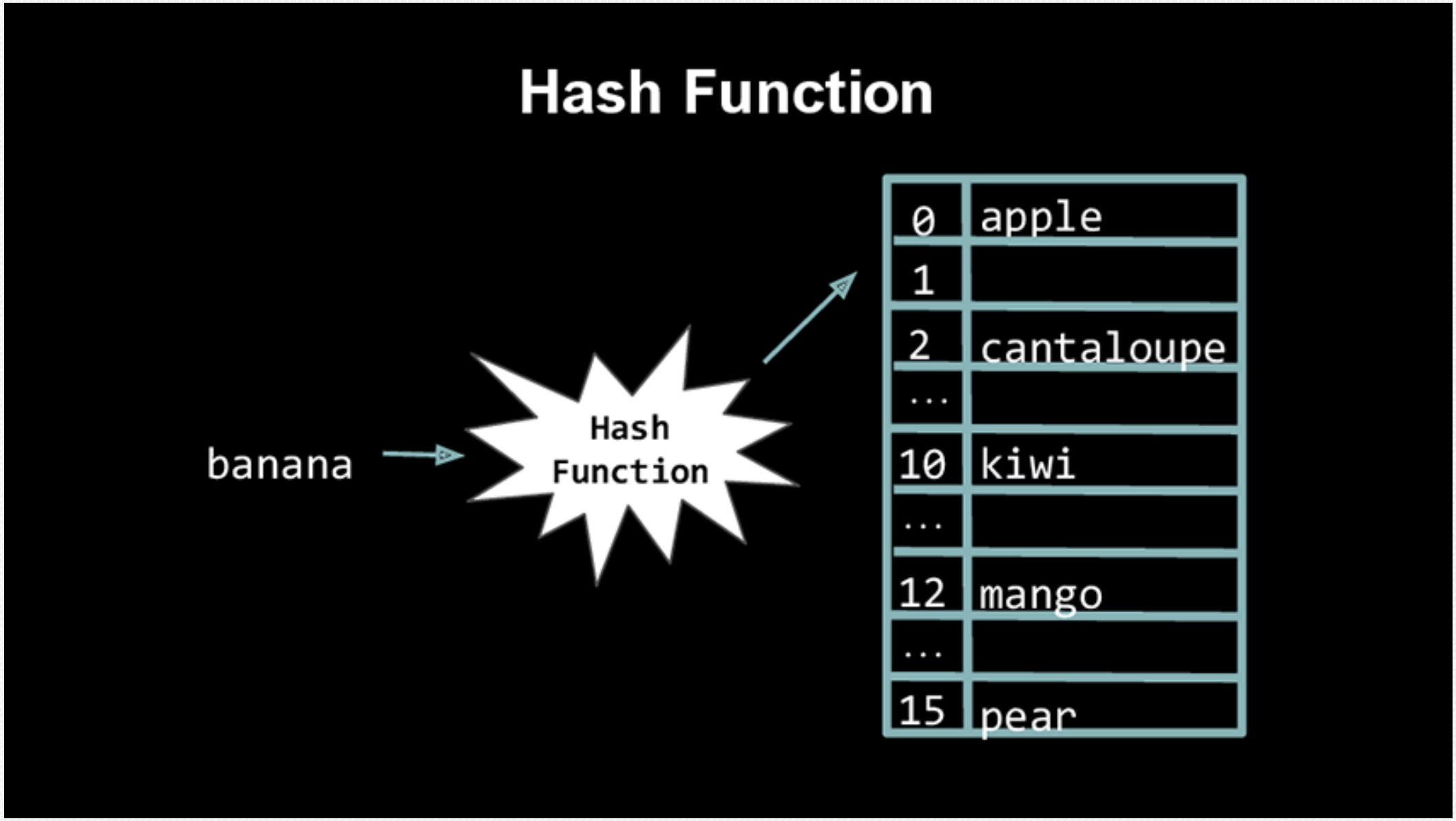


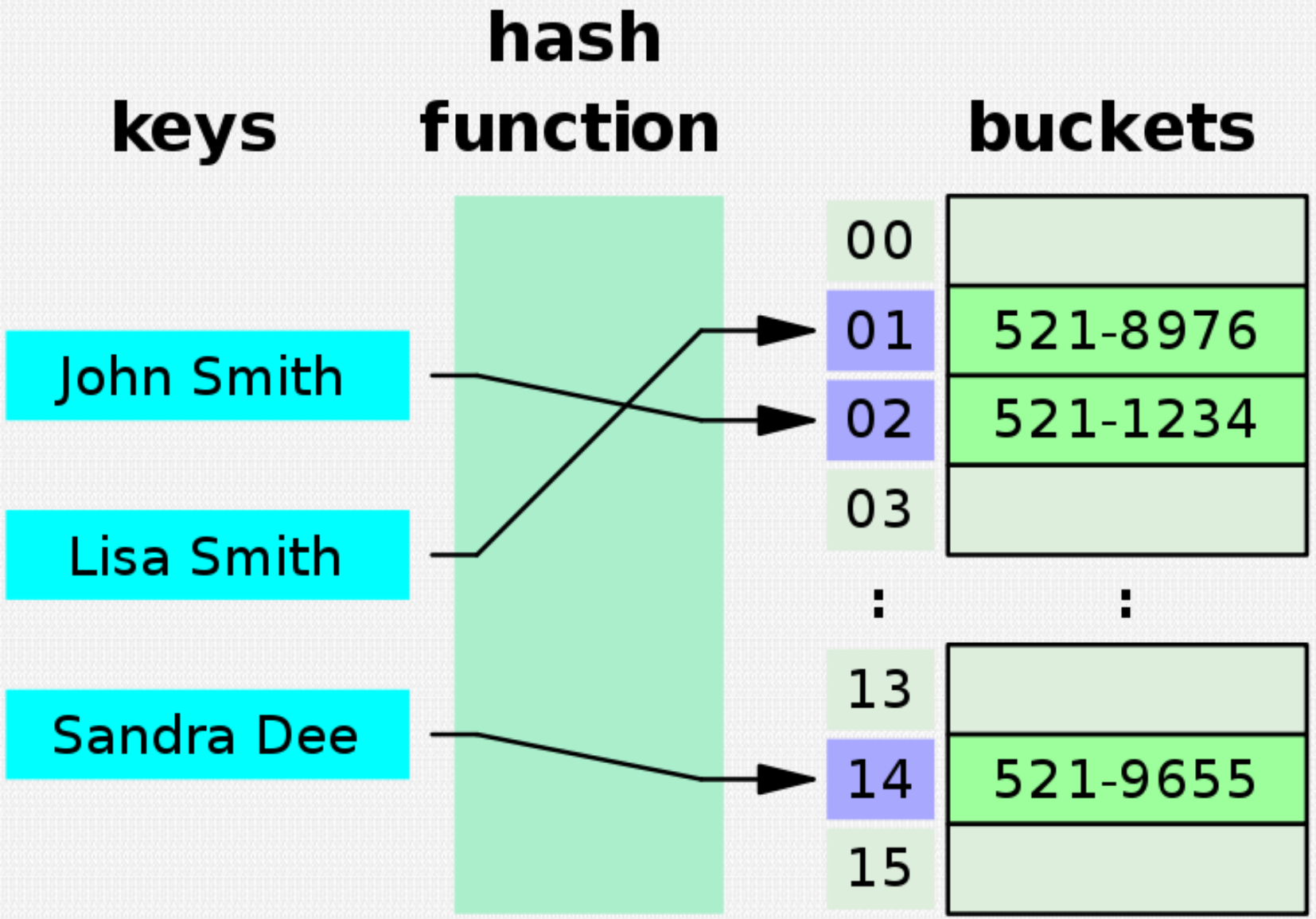
- (1,20)
- (2,70)
- (42,80)
- (4,25)
- (12,44)
- (14,32)
- (17,11)
- (13,78)
- (37,98)

Sr.No.	Key	Hash	Array Index
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14
7	17	17 % 20 = 17	17
8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	17

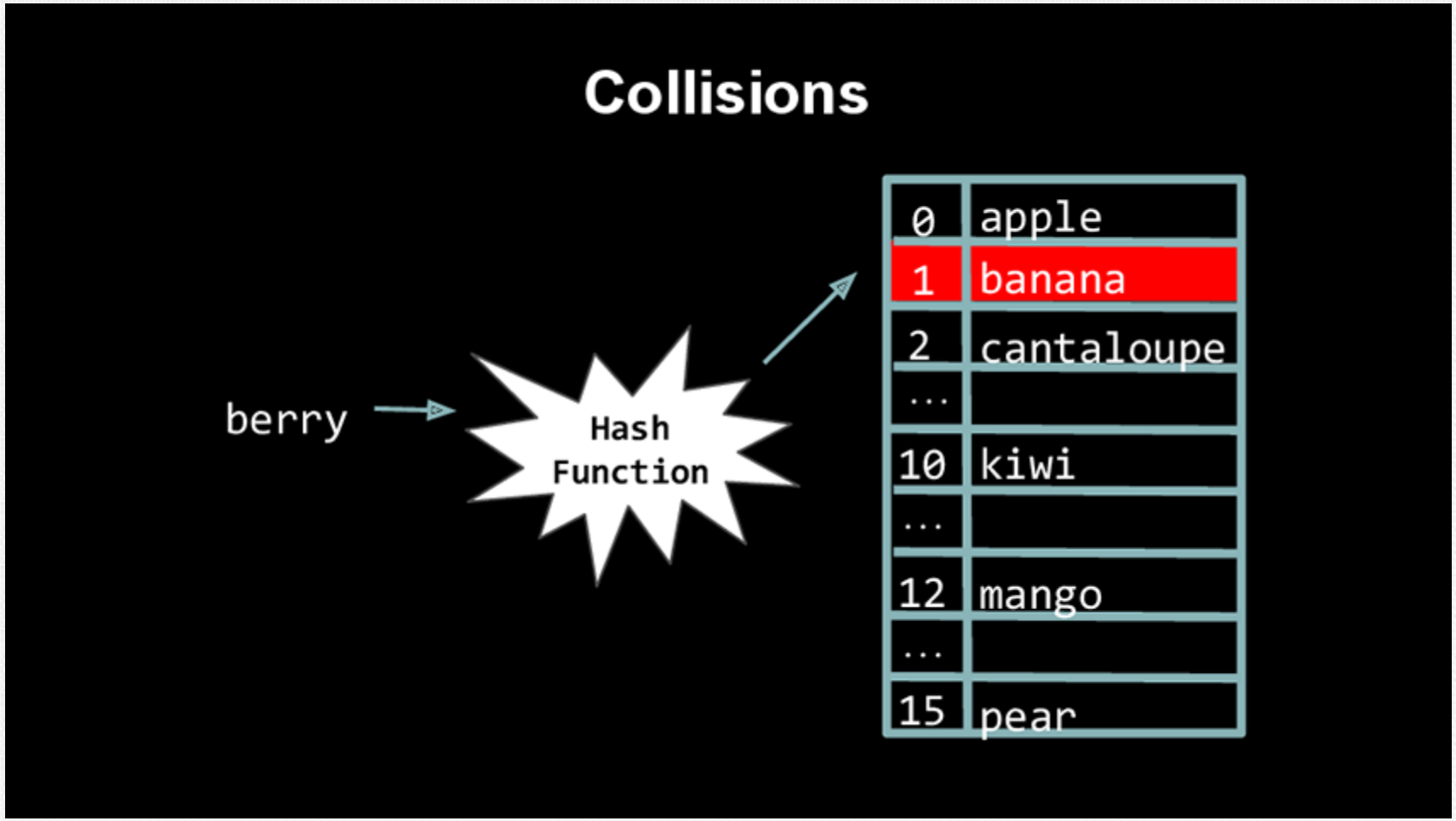
Hash Tables


banana
apple
kiwi
mango
pear
cantaloupe
Hash Function
0 apple
1 banana
2 cantaloupe
...
10 kiwi
...
12 mango
...
15 pear

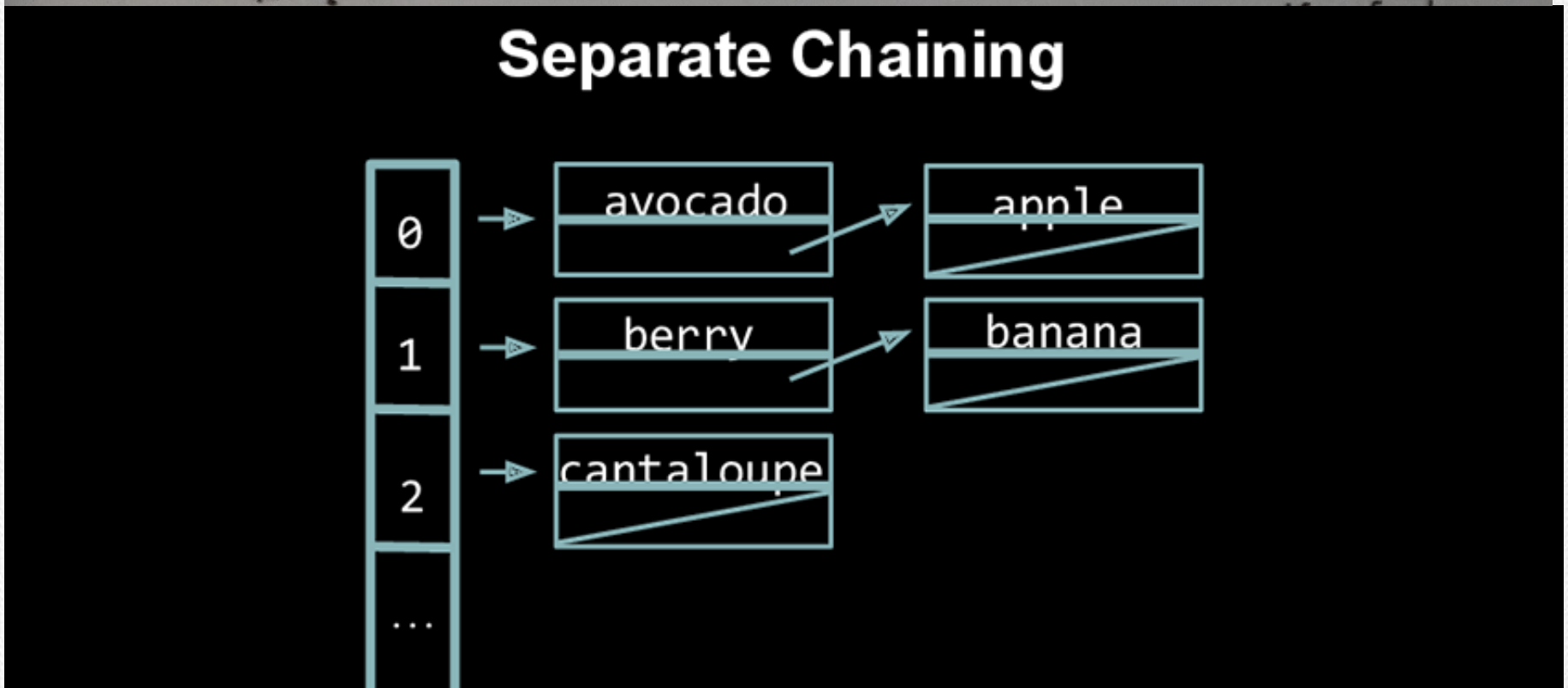

Hash Function
banana
Hash Function
0
apple
1
2
cantaloupe
...
10
kiwi
...
12
mango
...
15
pear

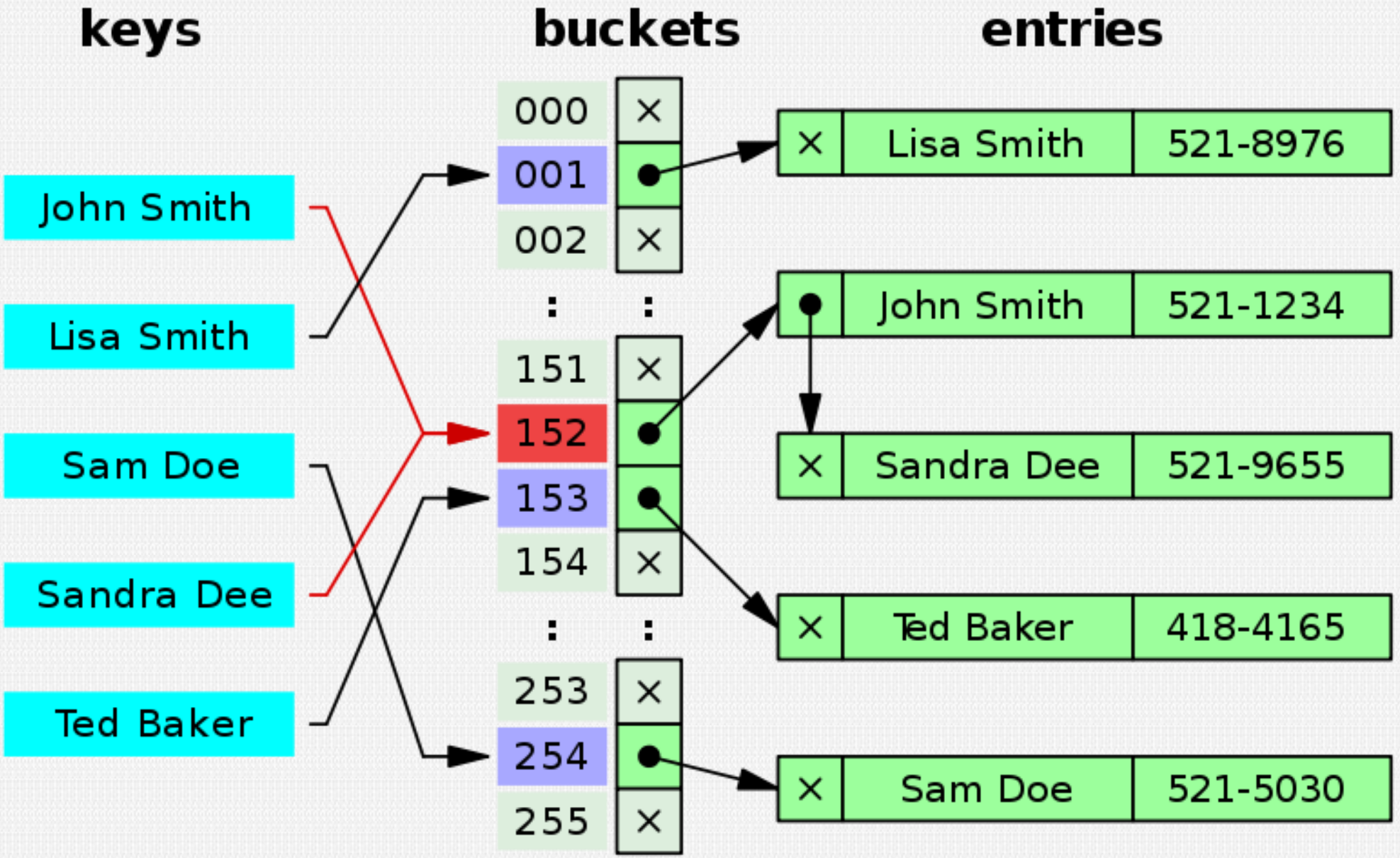

keys
hash function
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
13
14
15
521-8976
521-1234
521-9655

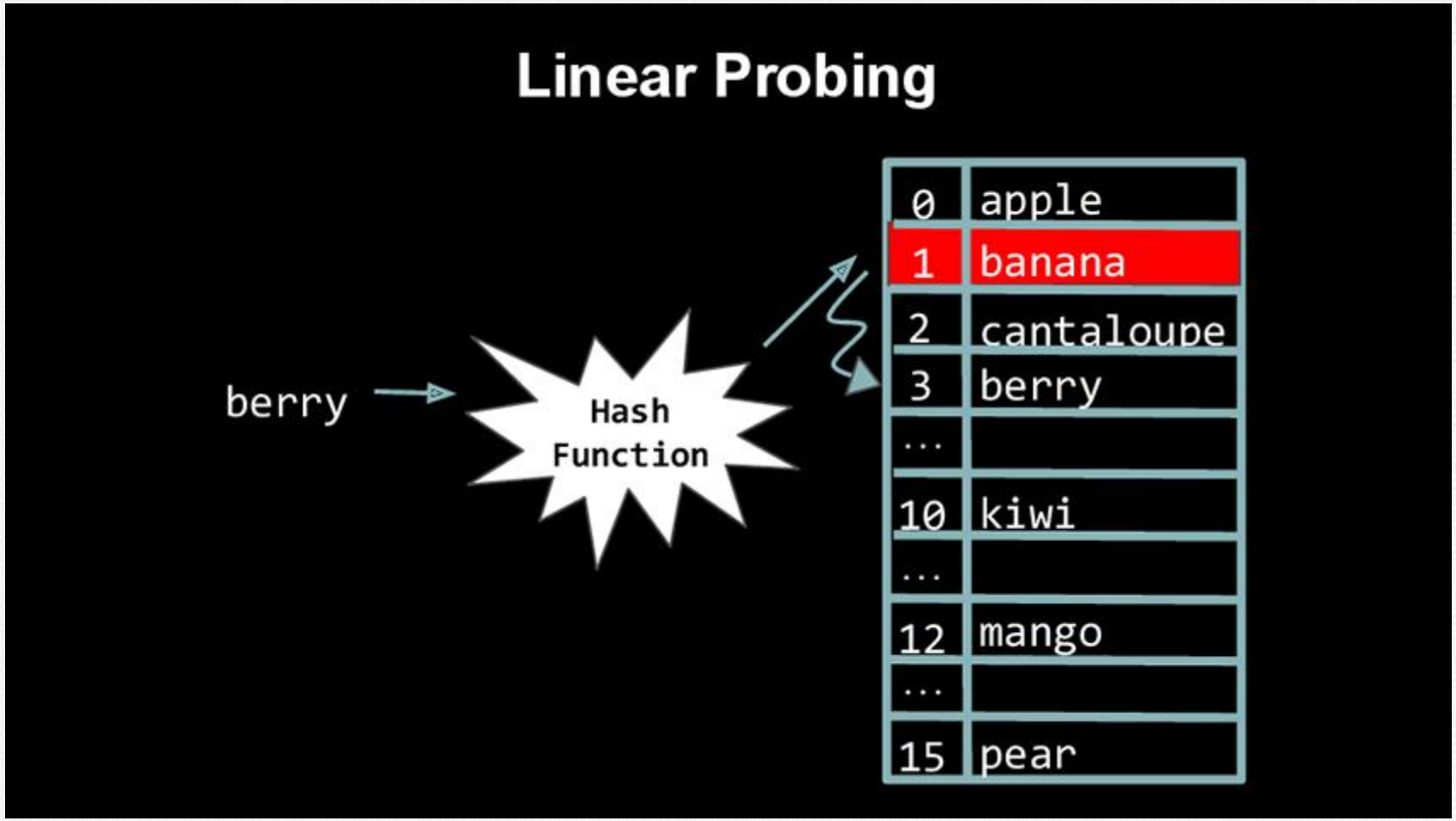
Բախումների (collisions) վերացման մեթոդներ՝
linear prob,
separate chaining:



Շղթաների մեթոդ: Ինչպես արդեն նշվել է, հեշավորման մեկ այլ մոտեցման դեպքում հեշ-աղյուսակը դիտարկվում է որպես կապակցված գծային ցուցակների զանգված: Ամեն մի ցուցակը կոչվում է սեգմենտ (բլոկ, bucket) և պարունակում է այն գրառումները, որոնց բանալիները հեշ-ֆունկցիայով արտապատկերվում են զանգվածի միևնույն հասցեին: Այս դեպքում x գրառումը, որի համար h(x)=i, ավելացվում է i-րդ կապակցված ցուցակին՝ նոր հանգույցի տեսքով:




keys
buckets
entries
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030


Linear Probing
berry
Hash Function
0 apple
1 banana
2 cantaloupe
3 berry
...
10 kiwi
...
12 mango
...
15 pear

- A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
- Java Hashtable class contains **unique** elements.
- Java Hashtable class **doesn't allow null key** or value.
- Java Hashtable class is **synchronized**.
- The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

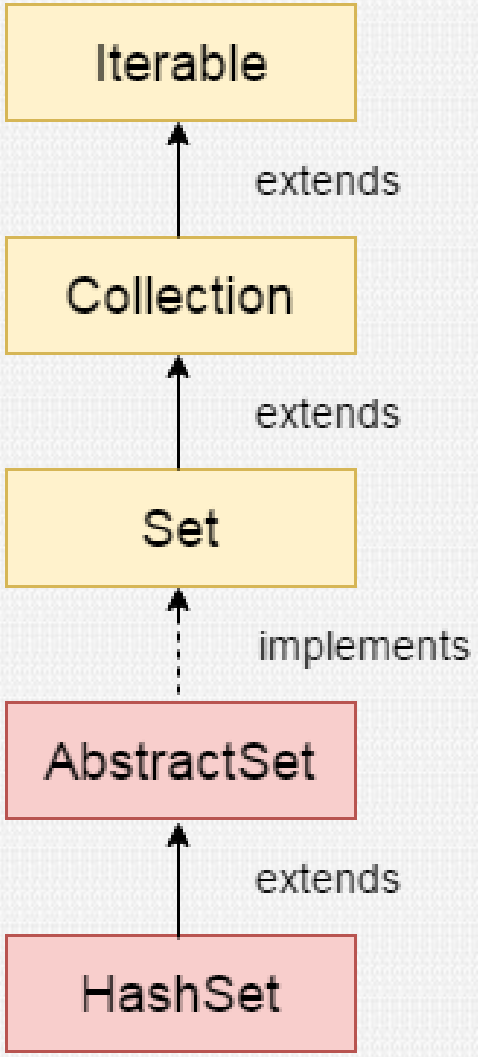
```
import java.util.*;
class Hashtable1{
public static void main(String args[]){
Hashtable<Integer,String> hm=new Hashtable<Integer,String>
();

hm.put(100,"Amit");
hm.put(102,"Ravi");
hm.put(101,"Vijay");
hm.put(103,"Rahul");

for(Map.Entry m:hm.entrySet()){
  System.out.println(m.getKey()+" "+m.getValue());
}
}
}
```

25. Set ինտերֆեյս

- *HashSet collection, ներդրված մեթոդները:*
- *LinkedHashSet collection, ներդրված մեթոդները:*
- *TreeSet collection, ներդրված մեթոդները:*

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains **unique elements** only.
- HashSet allows **null value**.
- HashSet class is **non synchronized**.
- HashSet doesn't maintain the **insertion order**. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for **search operations**.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.



public class HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Serializable

HashSet (sync)	Set	Backed by HashMap (a Hash table data structure). Elements of the set are populated as key of the HashMap. Allows at most one null.	**add, remove, contains, size**: O(1) **Iteration**: O(n + capacity). Better don't set initial capacity (size of backing hasMap) too high or load factor too low if iteration is frequently used.
LinkedHashSet (sync)	Set	Backed by LinkedHashMap where elements of this LinkedHashSet are populated as key of the Map. Maintains elements in insertion order. Allows at most one null.	**add, remove, contains, size**: O(1) **Iteration**: O(n), slightly slow that of HashSet, due to maintaining the linked list.

Constructors of Java HashSet class

SN	Constructor	Description
1)	HashSet()	It is used to construct a default HashSet.
2)	HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.
3)	HashSet(int capacity, float loadFactor)	It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.
4)	HashSet(Collection<? extends E> c)	It is used to initialize the hash set by using the elements of the collection c.

```java
import java.util.*;
class HashSet1{
 public static void main(String args[]){
  //Creating HashSet and adding elements
    HashSet<String> set=new HashSet();
          set.add("One");
          set.add("Two");
          set.add("Three");
          set.add("Four");
          set.add("Five");
          Iterator<String> i=set.iterator();
          while(i.hasNext())
          {
          System.out.println(i.next());
          }
 }
}
```

```
Five

One

Four

Two

Three
```

In this example, we see that HashSet doesn't allow duplicate elements.

```
import java.util.*;
class HashSet2{
 public static void main(String args[]){
  //Creating HashSet and adding elements
  HashSet<String> set=new HashSet<String>();
  set.add("Ravi");
  set.add("Vijay");
  set.add("Ravi");
  set.add("Ajay");
  //Traversing elements
  Iterator<String> itr=set.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

```
Ajay
Vijay
Ravi
```

Java HashSet example to remove elements

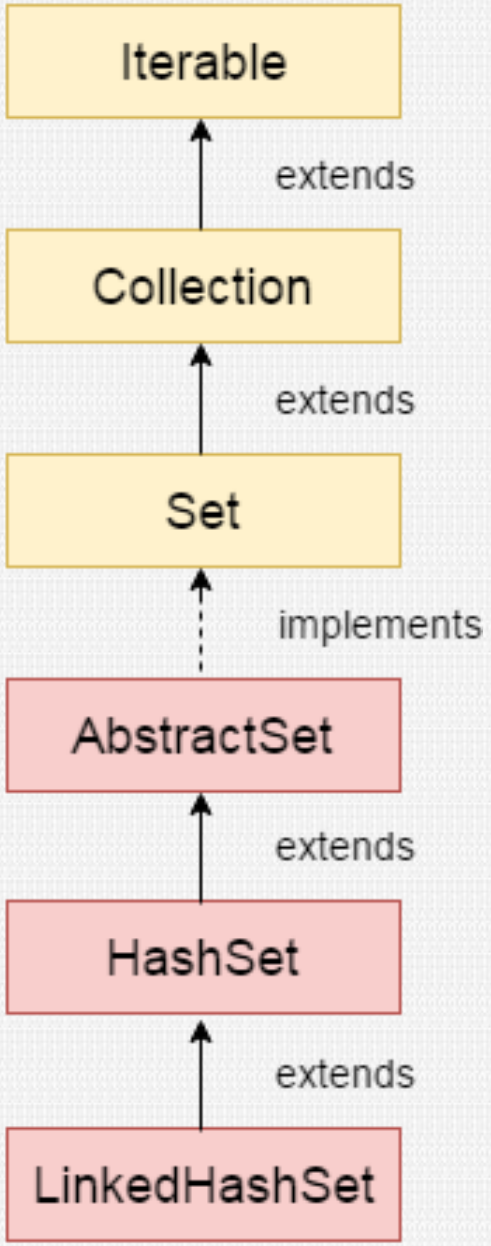
Here, we see different ways to remove an element.

```
import java.util.*;
class HashSet3{
 public static void main(String args[]){
  HashSet<String> set=new HashSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Arun");
        set.add("Sumit");
        System.out.println("An initial list of elements: "+set);
        //Removing specific element from HashSet
        set.remove("Ravi");
        System.out.println("After invoking remove(object) method: "+set);
        HashSet<String> set1=new HashSet<String>();
        set1.add("Ajay");
        set1.add("Gaurav");
        set.addAll(set1);
        System.out.println("Updated List: "+set);
        //Removing all the new elements from HashSet
        set.removeAll(set1);
        System.out.println("After invoking removeAll() method: "+set);
        //Removing elements on the basis of specified condition
        set.removeIf(str->str.contains("Vijay"));
        System.out.println("After invoking removeIf() method: "+set);
        //Removing all the elements available in the set
        set.clear();
        System.out.println("After invoking clear() method: "+set);
 }
```

Java LinkedHashSet class

Java LinkedHashSet class is a **Hashtable** and **Linked list** implementation of the set interface. It inherits HashSet class and implements Set interface.
The important points about Java LinkedHashSet class are:

- Java **LinkedHashSet** class contains **unique** elements only like HashSet.
- Java **LinkedHashSet** class provides all optional set operation and **permits null elements**.
- Java LinkedHashSet class is **non synchronized**.
- Java LinkedHashSet class maintains **insertion order**.

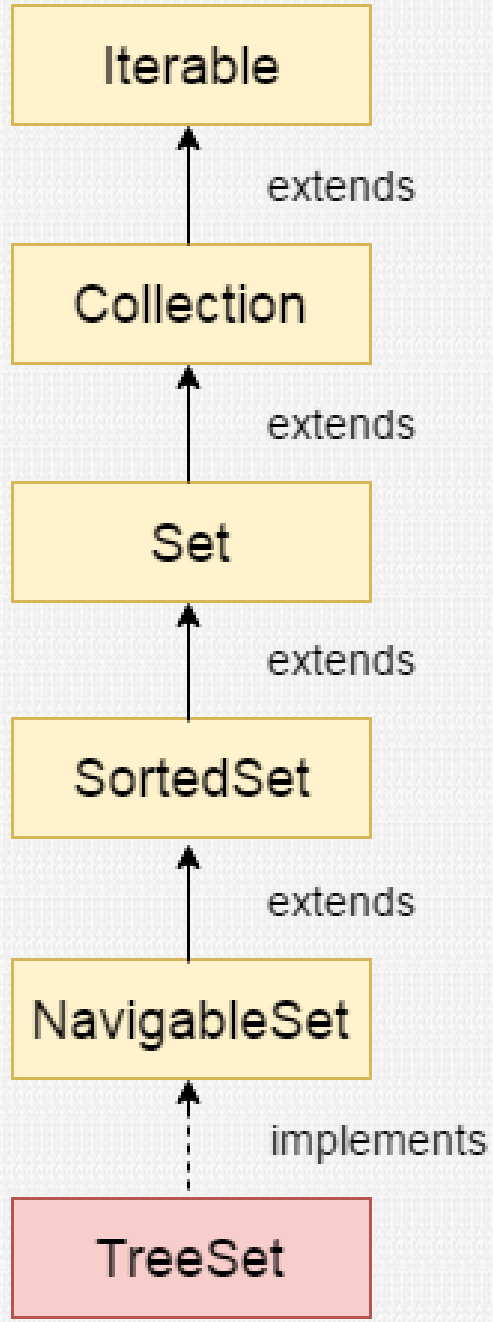


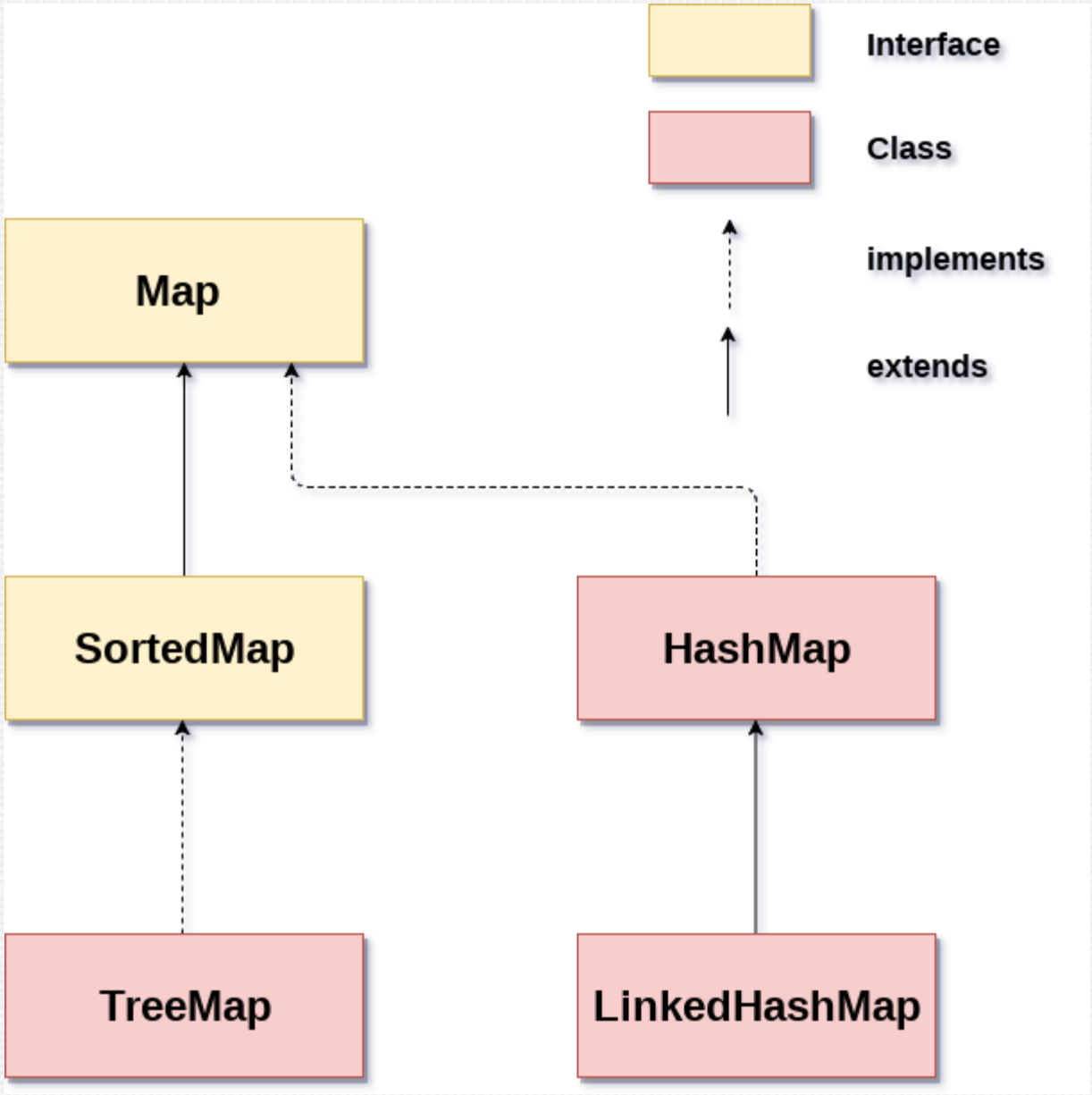
Java TreeSet class

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Java **TreeSet** class contains unique elements only like HashSet.
- Java TreeSet class **access and retrieval** times are quiet fast.
- Java TreeSet class doesn't allow **null element**.
- Java TreeSet class is **non synchronized**.
- Java TreeSet class maintains **ascending order**.




Interface
Class
implements
extends
Map
SortedMap
HashMap
TreeMap
LinkedHashMap

Java Map Example: Generic (New Style)

```
import java.util.*;
class MapExample2{
 public static void main(String args[]){
  Map<Integer,String> map=new HashMap<Integer,String>();
  map.put(100,"Amit");
  map.put(101,"Vijay");
  map.put(102,"Rahul");
  //Elements can traverse in any order
  for(Map.Entry m:map.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```

Output:

```
102 Rahul
100 Amit
101 Vijay
```

Java Map Example: comparingByKey()

```
import java.util.*;
class MapExample3{
 public static void main(String args[]){
Map<Integer,String> map=new HashMap<Integer,String>();
      map.put(100,"Amit");
      map.put(101,"Vijay");
      map.put(102,"Rahul");
      //Returns a Set view of the mappings contained in this map
      map.entrySet()
      //Returns a sequential Stream with this collection as its source
      .stream()
      //Sorted according to the provided Comparator
      .sorted(Map.Entry.comparingByKey())
      //Performs an action for each element of this stream
      .forEach(System.out::println);
 }
}
```

Java Map Example: comparingByKey() in Descending Order

```
import java.util.*;
class MapExample4{
 public static void main(String args[]){
Map<Integer,String> map=new HashMap<Integer,String>();
      map.put(100,"Amit");
      map.put(101,"Vijay");
      map.put(102,"Rahul");
      //Returns a Set view of the mappings contained in this map
      map.entrySet()
      //Returns a sequential Stream with this collection as its source
      .stream()
      //Sorted according to the provided Comparator
      .sorted(Map.Entry.comparingByKey(Comparator.reverseOrder()))
      //Performs an action for each element of this stream
      .forEach(System.out::println);
 }
}
```

```
102=Rahul
101=Vijay
100=Amit
```

Java Map Example: comparingByValue()

```
import java.util.*;
class MapExample5{
 public static void main(String args[]){
Map<Integer,String> map=new HashMap<Integer,String>();
      map.put(100,"Amit");
      map.put(101,"Vijay");
      map.put(102,"Rahul");
      //Returns a Set view of the mappings contained in this map
      map.entrySet()
      //Returns a sequential Stream with this collection as its source
      .stream()
      //Sorted according to the provided Comparator
      .sorted(Map.Entry.comparingByValue())
      //Performs an action for each element of this stream
      .forEach(System.out::println);
 }
}
```

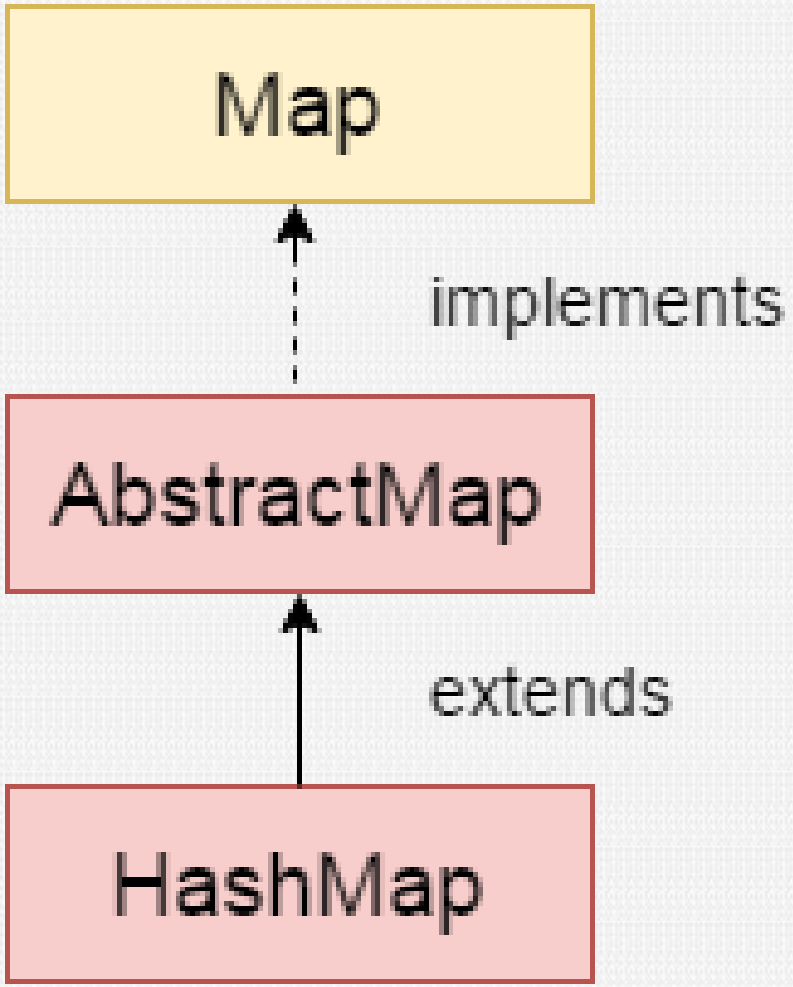
Output:

```
100=Amit
```

Java Map Example: comparingByValue() in Descending Order

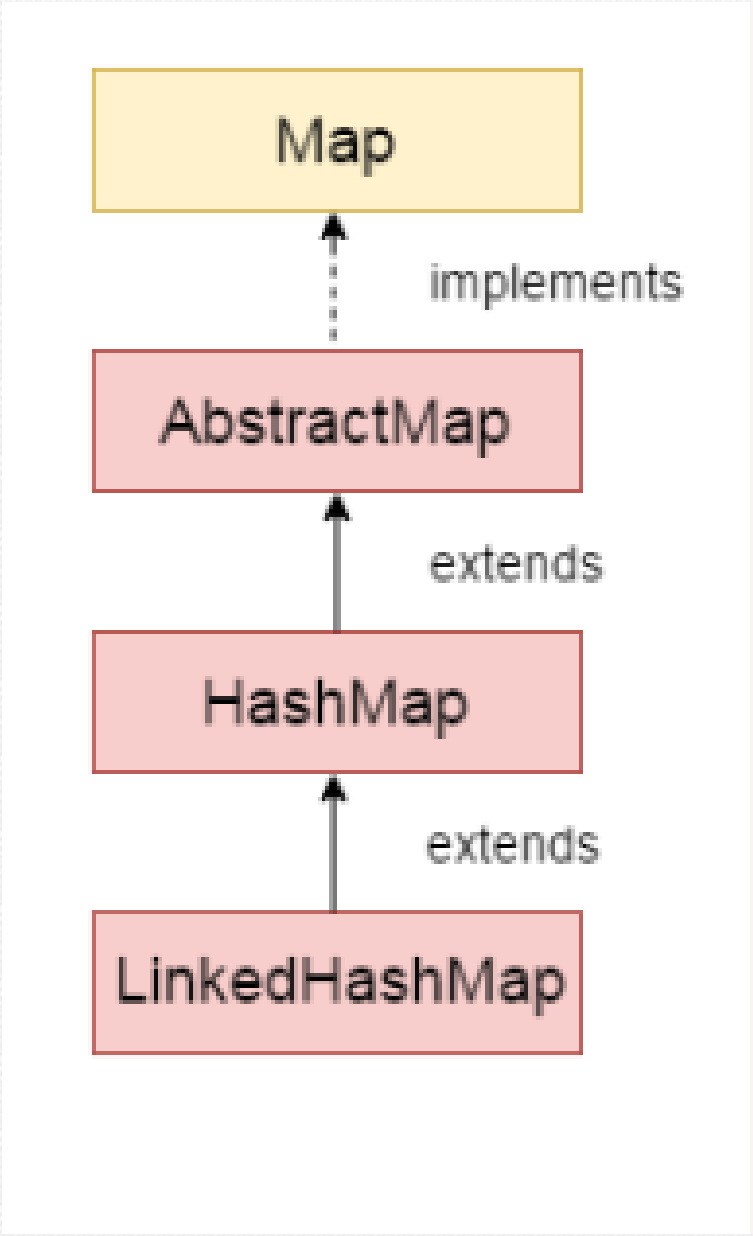
```
import java.util.*;
class MapExample6{
 public static void main(String args[]){
Map<Integer,String> map=new HashMap<Integer,String>();
      map.put(100,"Amit");
      map.put(101,"Vijay");
      map.put(102,"Rahul");
     //Returns a Set view of the mappings contained in this map
     map.entrySet()
     //Returns a sequential Stream with this collection as its source
     .stream()
     //Sorted according to the provided Comparator
     .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
     //Performs an action for each element of this stream
     .forEach(System.out::println);
 }
}
```

- Java HashMap class contains values based on the key.
- Java HashMap class contains only unique keys.
- Java HashMap class may have one null key and multiple null values.
- Java HashMap class is non synchronized.
- Java HashMap class maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.



```
import java.util.*;
class HashMap1{
 public static void main(String args[]){
  HashMap<Integer,String> hm=new HashMap<Integer,String>();
   System.out.println("Initial list of elements: "+hm);
    hm.put(100,"Amit");
    hm.put(101,"Vijay");
    hm.put(102,"Rahul");

    System.out.println("After invoking put() method ");
    for(Map.Entry m:hm.entrySet()){
     System.out.println(m.getKey()+" "+m.getValue());
    }

    hm.putIfAbsent(103, "Gaurav");
    System.out.println("After invoking putIfAbsent() method ");
    for(Map.Entry m:hm.entrySet()){
        System.out.println(m.getKey()+" "+m.getValue());
       }
    HashMap<Integer,String> map=new HashMap<Integer,String>();
    map.put(104,"Ravi");
    map.putAll(hm);
    System.out.println("After invoking putAll() method ");
    for(Map.Entry m:map.entrySet()){
        System.out.println(m.getKey()+" "+m.getValue());
       }
 }
}
```

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains **unique** elements.
- Java LinkedHashMap may have **one null key** and **multiple null** values.
- Java LinkedHashMap is non synchronized.
- Java LinkedHashMap maintains **insertion order**.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75



Java LinkedHashMap Example

```
import java.util.*;
class LinkedHashMap1{
 public static void main(String args[]){

  LinkedHashMap<Integer,String> hm=new LinkedHashMap<Integer,String>();

  hm.put(100,"Amit");
  hm.put(101,"Vijay");
  hm.put(102,"Rahul");

for(Map.Entry m:hm.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```

```
Output:100 Amit
       101 Vijay
       102 Rahul
```

The important points about Java TreeMap class are:

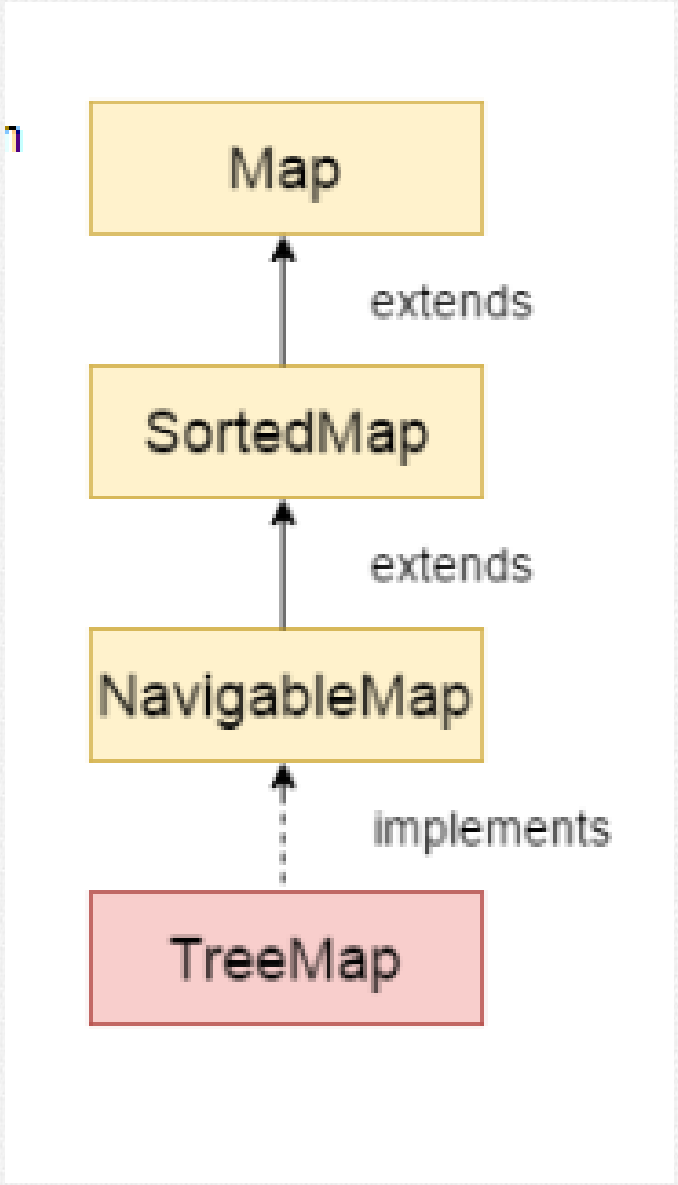
•Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.

•Java TreeMap contains only **unique elements**.

•Java TreeMap **cannot** have a **null key but can have multiple null values**.

•Java TreeMap is non synchronized.

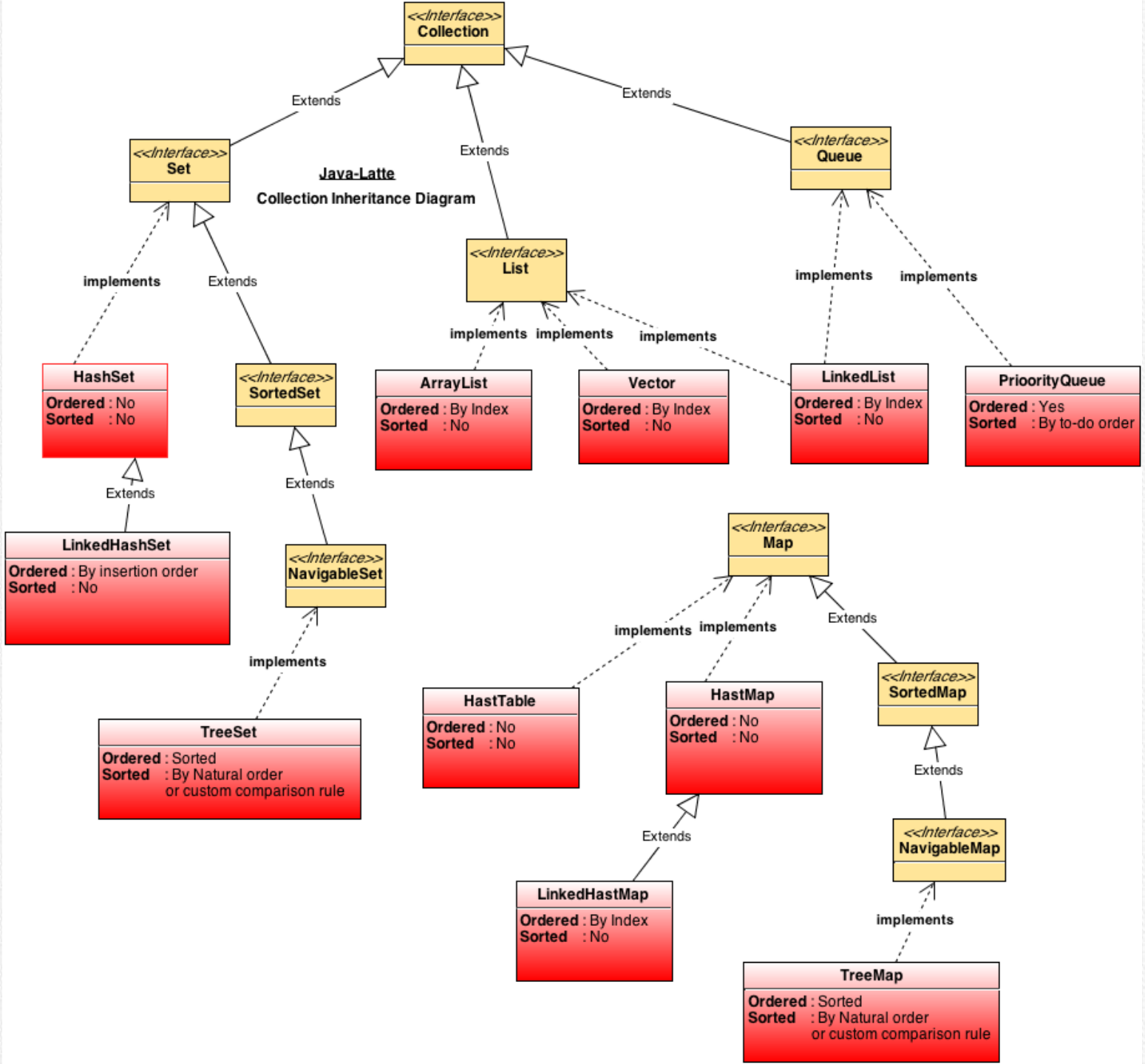
•Java TreeMap **maintains ascending order**.



HashMap	Hashtable
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value.**
3) HashMap is a **new class introduced in JDK 1.2.**	Hashtable is a **legacy class.**
4) HashMap is **fast.**	Hashtable is **slow.**
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator.**	Hashtable is **traversed by Enumerator and Iterator.**
7) Iterator in HashMap is **fail-fast.**	Enumerator in Hashtable is **not fail-fast.**
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

Class	Map	Set	List	Ordered	Sorted
HashMap	X			No	No
Hashtable	X			No	No
TreeMap	X			Sorted	By *natural order* or custom comparison rules
LinkedHashMap	X			By insertion order or last access order	No
HashSet		X		No	No
TreeSet		X		Sorted	By *natural order* or custom comparison rules
LinkedHashSet		X		By insertion order or last access order	No
ArrayList			X	By index	No
Vector			X	By index	No
LinkedList			X	By index	No

Метод	Тип	Время
get, set	ArrayList	O(1)
add, remove	ArrayList	O(n)
contains, indexOf	ArrayList	O(n)
get, put, remove, containsKey	HashMap	O(1)
add, remove, contains	HashSet	O(1)
add, remove, contains	LinkedHashSet	O(1)
get, set, add, remove (с любого конца)	LinkedList	O(1)
get, set, add, remove (по индексу)	LinkedList	O(n)
contains, indexOf	LinkedList	O(n)
peek	PriorityQueue	O(1)
add, remove	PriorityQueue	O(log n)
remove, get, put, containsKey	TreeMap	O(log n)
add, remove, contains	TreeSet	O(log n)


Java-Latte
Collection Inheritance Diagram
<<Interface>>
Collection
<<Interface>>
Set
<<Interface>>
List
<<Interface>>
Queue
<<Interface>>
SortedSet
<<Interface>>
NavigableSet
<<Interface>>
Map
<<Interface>>
SortedMap
<<Interface>>
NavigableMap
HashSet
Ordered : No
Sorted : No
LinkedHashSet
Ordered : By insertion order
Sorted : No
TreeSet
Ordered : Sorted
Sorted : By Natural order
or custom comparison rule
ArrayList
Ordered : By Index
Sorted : No
Vector
Ordered : By Index
Sorted : No
LinkedList
Ordered : By Index
Sorted : No
PrioorityQueue
Ordered : Yes
Sorted : By to-do order
HastTable
Ordered : No
Sorted : No
HastMap
Ordered : No
Sorted : No
LinkedHastMap
Ordered : By Index
Sorted : No
TreeMap
Ordered : Sorted
Sorted : By Natural order
or custom comparison rule
Extends
implements

1.Comparable, comparator ինտերֆեյսներ

1.Comparable ինտերֆեյս:

2.Comparator ինտերֆեյս:

3.Comparable և comparator համեմատություն:

2.Collections կլաս

1.Collections կլասի մեթոդներ:

2.Կարգավորում (sorting) Collections կլասի միջոցով:

3.EnumSet և EnumMap կլասներ

Java Comparable interface

- compareTo(Object obj) method

public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

File: Student.java

```
class Student implements Comparable<Student>{
int rollno;
String name;
int age;
Student(int rollno,String name,int age){
this.rollno=rollno;
this.name=name;
this.age=age;
}

public int compareTo(Student st){
if(age==st.age)
return 0;
else if(age>st.age)
return 1;
else
return -1;
}
}
```

File: TestSort1.java

```
import java.util.*;
public class TestSort1{
public static void main(String args[]){
ArrayList<Student> al=new ArrayList<Student>();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

Collections.sort(al);
for(Student st:al){
System.out.println(st.rollno+" "+st.name+" "+st.age);
}
}
}
```

```
105 Jai 21
101 Vijay 23
106 Ajay 27
```

File: Student.java

```
class Student implements Comparable<Student>{
 int rollno;
 String name;
 int age;
 Student(int rollno,String name,int age){
 this.rollno=rollno;
 this.name=name;
 this.age=age;
 }

 public int compareTo(Student st){
 if(age==st.age)
 return 0;
 else if(age<st.age)
 return 1;
 else
 return -1;
 }
 }
```

File: TestSort2.java

```
import java.util.*;
public class TestSort2{
public static void main(String args[]){
ArrayList<Student> al=new ArrayList<Student>();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

Collections.sort(al);
for(Student st:al){
System.out.println(st.rollno+" "+st.name+" "+st.age);
}
}
}
```

```
106 Ajay 27
101 Vijay 23
105 Jai 21
```

Comparator interface

Java Comparator Example (Generic)

Student.java

```
class Student{
int rollno;
String name;
int age;
Student(int rollno,String name,int age){
this.rollno=rollno;
this.name=name;
this.age=age;
}
}
```

AgeComparator.java

```
import java.util.*;
class AgeComparator implements Comparator<Student>{
public int compare(Student s1,Student s2){
if(s1.age==s2.age)
return 0;
else if(s1.age>s2.age)
return 1;
else
return -1;
}
}
```

NameComparator.java

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

```
import java.util.*;
class NameComparator implements Comparator<Student>{
public int compare(Student s1,Student s2){
return s1.name.compareTo(s2.name);
}
}
```

```
import java.util.*;
import java.io.*;
class Simple{
public static void main(String args[]){

ArrayList<Student> al=new ArrayList<Student>();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

System.out.println("Sorting by Name");

Collections.sort(al,new NameComparator());
for(Student st: al){
System.out.println(st.rollno+" "+st.name+" "+st.age);
}

System.out.println("Sorting by age");

Collections.sort(al,new AgeComparator());
for(Student st: al){
System.out.println(st.rollno+" "+st.name+" "+st.age);
}
}
}
```

```
Sorting by Name
106 Ajay 27
105 Jai 21
101 Vijay 23

Sorting by age
105 Jai 21
101 Vijay 23
106 Ajay 27
```

Տարբերություններ

Comparable	Comparator
1) Comparable provides a **single sorting sequence.** In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences.** In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

Java EnumSet class

Java EnumSet class is the specialized Set implementation for use with enum types. It inherits AbstractSet class and implements the Set interface.

EnumSet class hierarchy

The hierarchy of EnumSet class is given in the figure given below.


EnumSet class hierarchy

EnumSet class declaration

Let's see the declaration for java.util.EnumSet class.

```
public abstract class EnumSet<E extends Enum<E>> extends AbstractSet<E> implements Cloneable, Seria
```

Java EnumSet Example

```
import java.util.*;
enum days {
  SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
public class EnumSetExample {
  public static void main(String[] args) {
    Set<days> set = EnumSet.of(days.TUESDAY, days.WEDNESDAY);
    // Traversing elements
    Iterator<days> iter = set.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());
  }
}
```

Output:

```
TUESDAY
WEDNESDAY
```

Java EnumSet Example: allOf() and noneOf()

```
import java.util.*;
enum days {
  SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
public class EnumSetExample {
  public static void main(String[] args) {
    Set<days> set1 = EnumSet.allOf(days.class);
      System.out.println("Week Days:"+set1);
      Set<days> set2 = EnumSet.noneOf(days.class);
      System.out.println("Week Days:"+set2);
  }
}
```

Output:

```
Week Days:[SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY]
Week Days:[]
```

EnumMap class declaration

Let's see the declaration for java.util.EnumMap class.

```
public class EnumMap<K extends Enum<K>,V> extends AbstractMap<K,V> implements Serializable, Cloneal
```

EnumMap class Parameters

Let's see the Parameters for java.util.EnumMap class.

- **K:** It is the type of keys maintained by this map.
- **V:** It is the type of mapped values.

Constructors of Java EnumMap class

Constructor	Description
EnumMap(Class<K> keyType)	It is used to create an empty enum map with the specified key type.
EnumMap(EnumMap<K,? extends V> m)	It is used to create an enum map with the same key type as the specified enum map.
EnumMap(Map<K,? extends V> m)	It is used to create an enum map initialized from the specified map.

```
import java.util.*;
public class EnumMapExample {
  // create an enum
  public enum Days {
  Monday, Tuesday, Wednesday, Thursday
  };
  public static void main(String[] args) {
  //create and populate enum map
  EnumMap<Days, String> map = new EnumMap<Days, String>(Days.class);
  map.put(Days.Monday, "1");
  map.put(Days.Tuesday, "2");
  map.put(Days.Wednesday, "3");
  map.put(Days.Thursday, "4");
  // print the map
  for(Map.Entry m:map.entrySet()){
     System.out.println(m.getKey()+" "+m.getValue());
    }
  }
}
```

Output:

```
Monday 1
Tuesday 2
Wednesday 3
Thursday 4
```

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class EnumMapExample {
// Creating enum
    public enum Key{
          One, Two, Three
          };
public static void main(String[] args) {
    EnumMap<Key, Book> map = new EnumMap<Key, Book>(Key.class);
    // Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    // Adding Books to Map
      map.put(Key.One, b1);
      map.put(Key.Two, b2);
      map.put(Key.Three, b3);
    // Traversing EnumMap
      for(Map.Entry<Key, Book> entry:map.entrySet()){
          Book b=entry.getValue();
          System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
       }
}
```

Զուգահեռ մշակումներ, Multithreading

- ԶՈՒԳԱՀԵՌ ՄՇԱԿՈՒՄՆԵՐԻ ԿԻՐԱՌՈՒԹՅՈՒՆԸ, SUPER COMPUTING:
- ԶՈՒԳԱՀԵՌ ՀԱՄԱԿԱՐԳՉԱՅԻՆ ՃԱՐՏԱՐԱՊԵՏՈՒԹՅՈՒՆՆԵՐ:
- Thread Scheduler, running, Join, Naming:

HIGH PERFORMANCE COMPUTING

Բարձր արտադրողականության հաշվարկը (HPC) բարձր արագությամբ տվյալների հաշվարկման և բարդ հաշվարկներ կատարելու հնարավորություն է:

Բարձր արտադրողականության հաշվարկները կամ HPC-ն սուպերհամակարգիչների կիրառումն է համակարգչային առաջադրանքների ժամանակ, որոնք կամ չափազանց մեծ են ստանդարտ համակարգիչների համար, կամ չափազանց շատ ժամանակ են պահանջում:

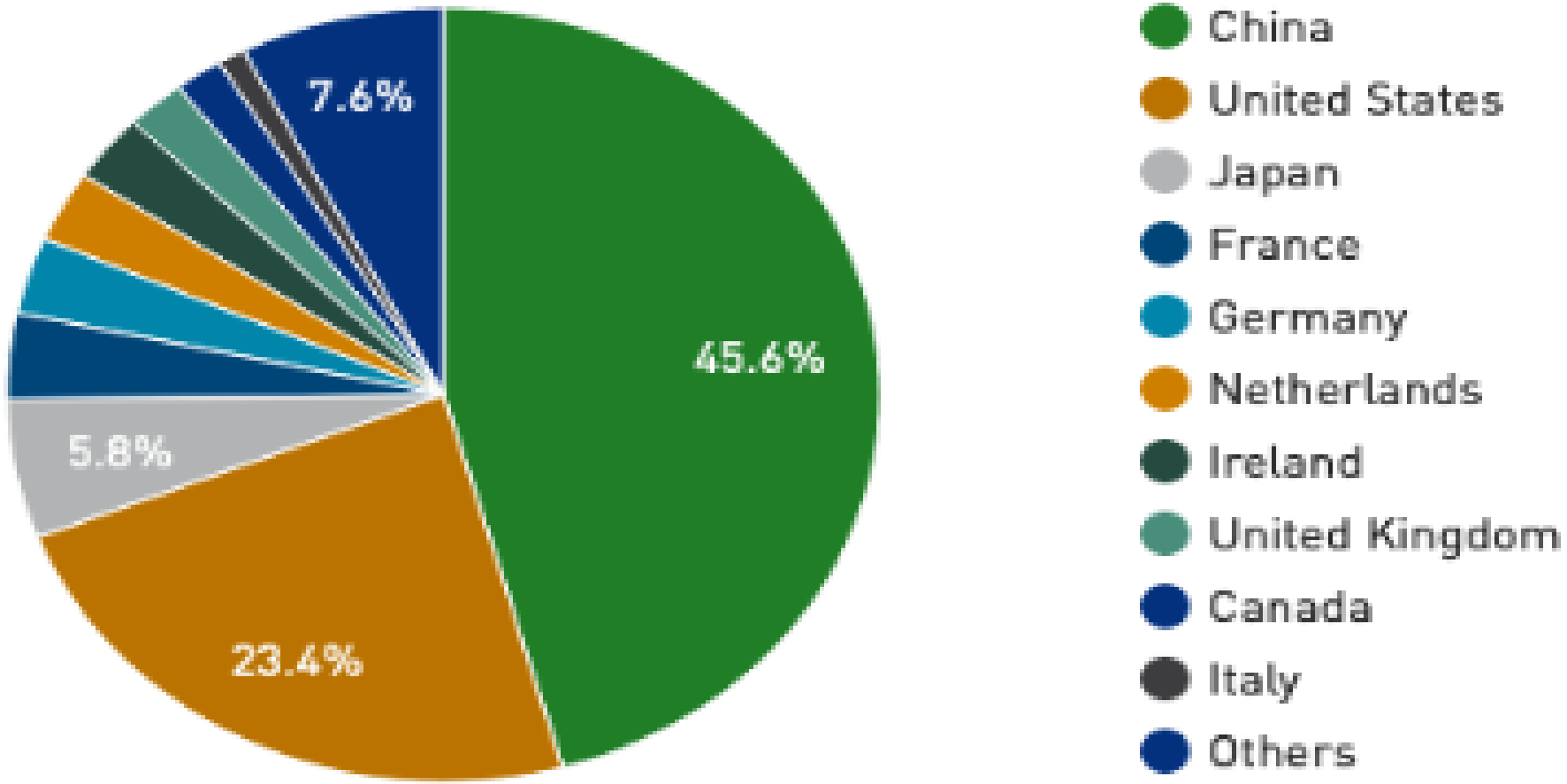
Սուպերհամակարգիչները դասակարգվում են Top500-ի սանդղակավորման աղյուսակի:Այս սանդղակավորման աղյուսակը պարունակում է 500 գերհամակարգիչներ, որոնք ներկայացված են նվազման կարգով:: Նախագիծը մեկնարկել է 1993 թվականին և 1993 թվականի հունիսից Top500-ը կազմվում է տարին երկու անգամ՝ հունիսին և նոյեմբերին հրապարակվում է սուպերհամակարգիչների թարմացված ցանկը և հիմնված է միայն ցանցի հանգույցների և արտադրողների տեղեկատվության վրա:

Top500 սանդղակավորման աղյուսակ՝

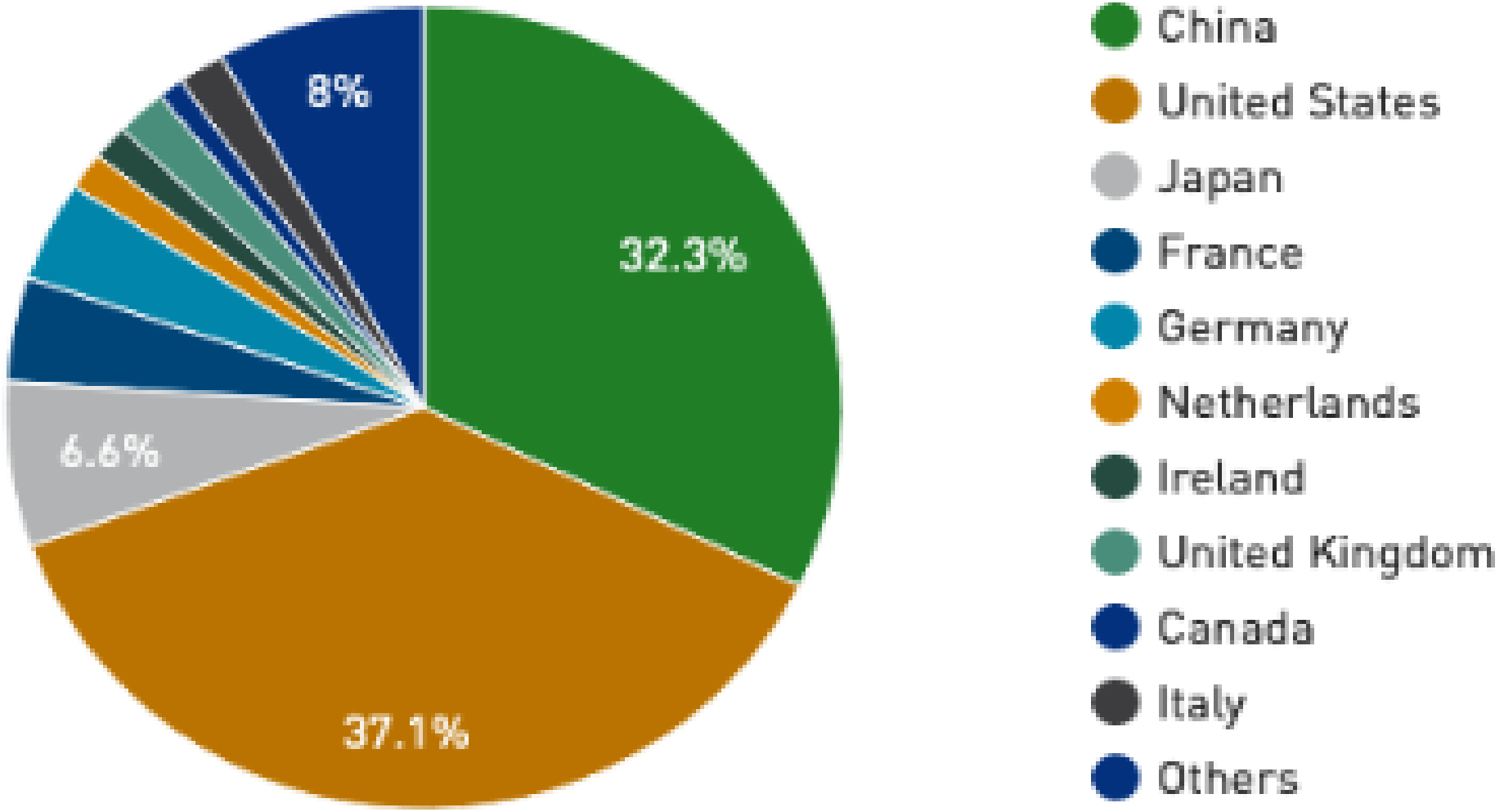
Rank	Rmax Rpeak (PFLOPS)	Name	Model	Processor	Interconnect	Vendor	Site country, year	Operating system
1 —	148.600 200.795	Summit	IBM Power System AC922	POWER9, Tesla V100	InfiniBand EDR	IBM	Oak Ridge National Laboratory United States, 2018	Linux (RHEL)
2 —	94.640 125.712	Sierra	IBM Power System S922LC	POWER9, Tesla V100	InfiniBand EDR	IBM	Lawrence Livermore National Laboratory United States, 2018	Linux (RHEL)
3 —	93.015 125.436	Sunway TaihuLight	Sunway MPP	SW26010	Sunway[26]	NRCPC	National Supercomputing Center in Wuxi China, 2016[26]	Linux (Raise)
4 —	61.445 100.679	Tianhe-2A	TH-IVB-FEP	Xeon E5–2692 v2, Matrix-2000[27]	TH Express-2	NUDT	National Supercomputing Center in Guangzhou China, 2013	Linux (Kylin)
5 ▲	23.516 38.746	Frontera	Dell C6420	Xeon Platinum 8280 (subsystems with e.g. POWER9 CPUs and Nvidia GPUs were added after official benchmarking[11])	InfiniBand HDR	Dell EMC	Texas Advanced Computing Center United States, 2019	Linux (CentOS)
6 ▼	21.230 27.154	Piz Daint	Cray XC50	Xeon E5-2690 v3, Tesla P100	Aries	Cray	Swiss National Supercomputing Centre Switzerland, 2016	Linux (CLE)
7 ▼	20.159 41.461	Trinity	Cray XC40	Xeon E5–2698 v3, Xeon Phi 7250	Aries	Cray	Los Alamos National Laboratory United States, 2015	Linux (CLE)
8 ▼	19.880 32.577	AI Bridging Cloud Infrastructure[28]	PRIMERGY CX2550 M4	Xeon Gold 6148, Tesla V100	InfiniBand EDR	Fujitsu	National Institute of Advanced Industrial Science and Technology Japan, 2018	Linux
9 —	19.477 26.874	SuperMUC-NG [29]	ThinkSystem SD530	Xeon Platinum 8174 (plus not benchmarked e.g. 32 cloud GPU nodes with Tesla V100[30])	Intel Omni-Path	Lenovo	Leibniz Supercomputing Centre Germany, 2018	Linux (SLES)
10 ▲	18.200 23.047	Lassen	IBM Power System S922LC	POWER9, Tesla V100	InfiniBand EDR	IBM	Lawrence Livermore National Laboratory United States, 2018	Linux (RHEL)

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Countries System Share

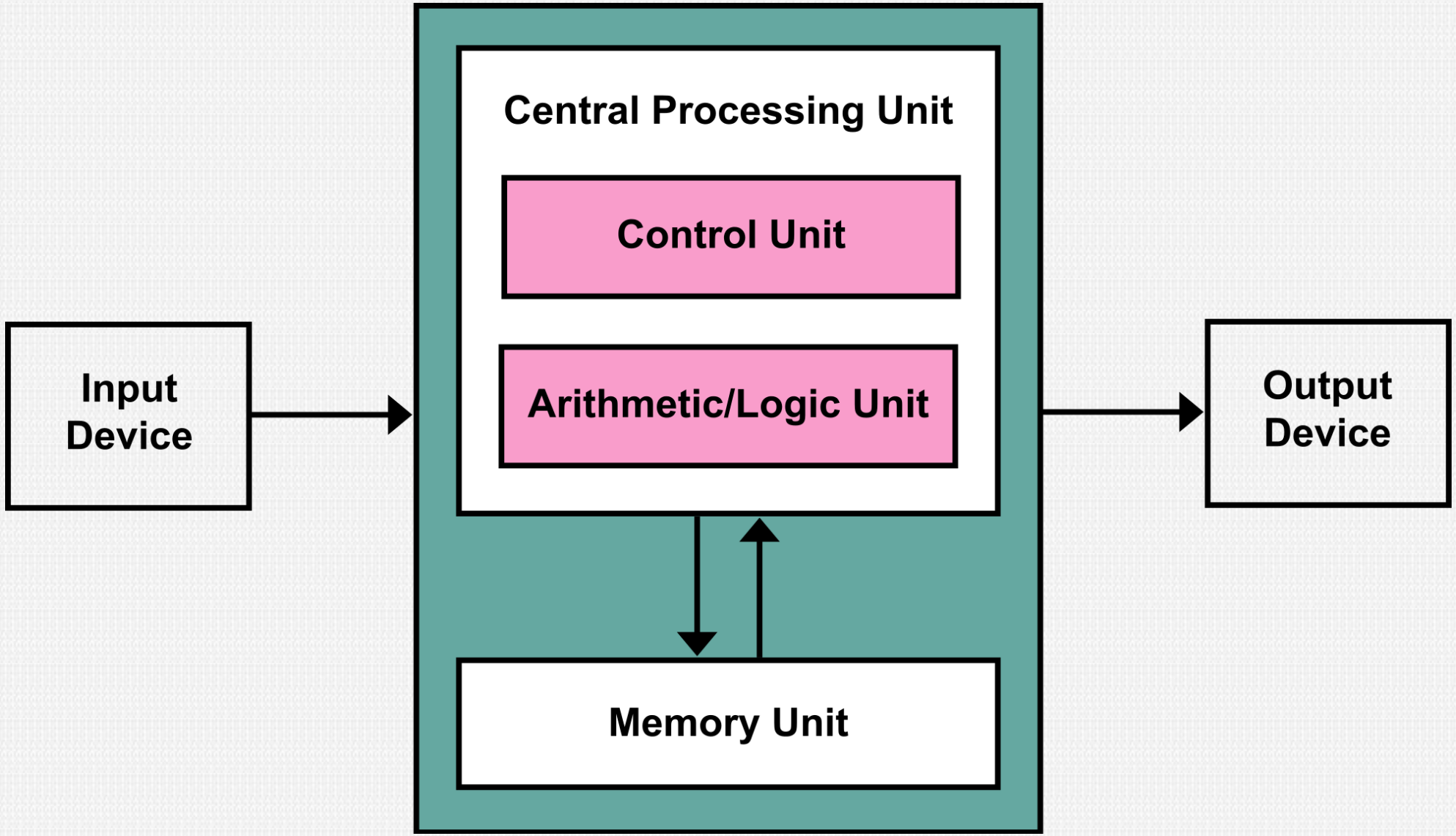


Countries Performance Share



Հաշվողական համակարգերի Ֆլինի դասակարգումը՝ SISD, SIMD, MISD, MIMD:

«ՋՈՆ ՆԵՅՄԱՆԻ ՃԱՐՏԱՐԱՊԵՏՈՒԹՅՈՒՆԸ»

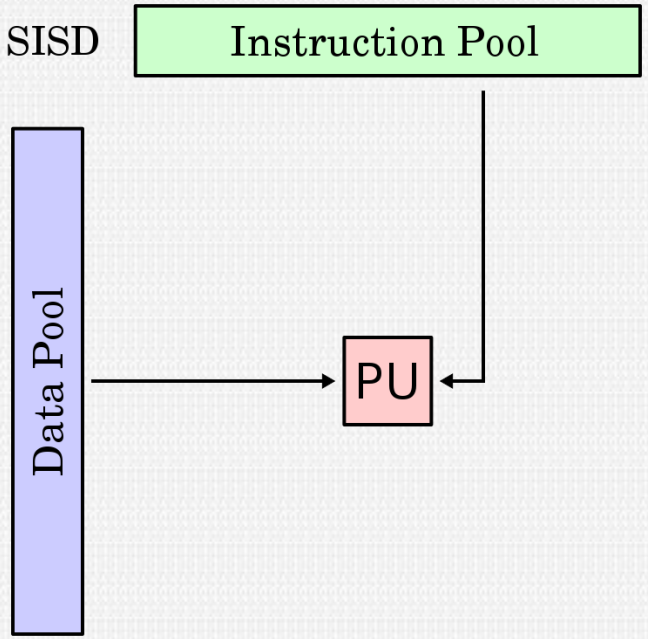


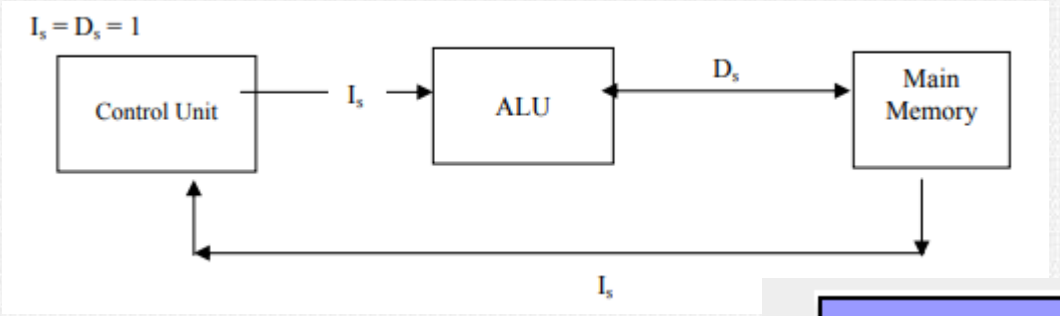
Ֆլինի դասակարգումն համակարգչային ճարտարապետության ընդհանուր դասակարգումն է՝ հիմնված հրամանի և տվյալների հոսքերի մեջ զուգահեռության առկայության վրա: Այն առաջարկվել է Մայքլ Ֆլինի կողմից 1966 թվականին և ընդլայնվել 1972 թվականին:

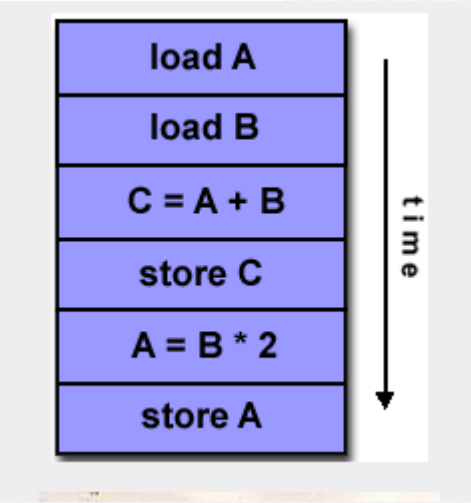
Ֆլինի դասակարգումն բաժանվում է 4 դասի՝

- SISD (single instruction stream over a single data stream)- Հաշվարկային համակարգ՝ հրամանների մեկ հոսքով և տվյալների մի հոսքով :
- SIMD (single instruction, multiple data) -Հաշվարկային համակարգ՝ հրամանների մեկ հոսքով և տվյալների բազմակի հոսքով:
- MISD (multiple instruction, single data) -Հաշվարկային համակարգ՝ հրամանների մի քանի հոսքով և տվյալների մի հոսքով:
- MIMD(multiple instruction, multiple data)-Հաշվարկային համակարգ՝ հրամանների բազմակի հոսքով և տվյալների բազմակի հոսքով:

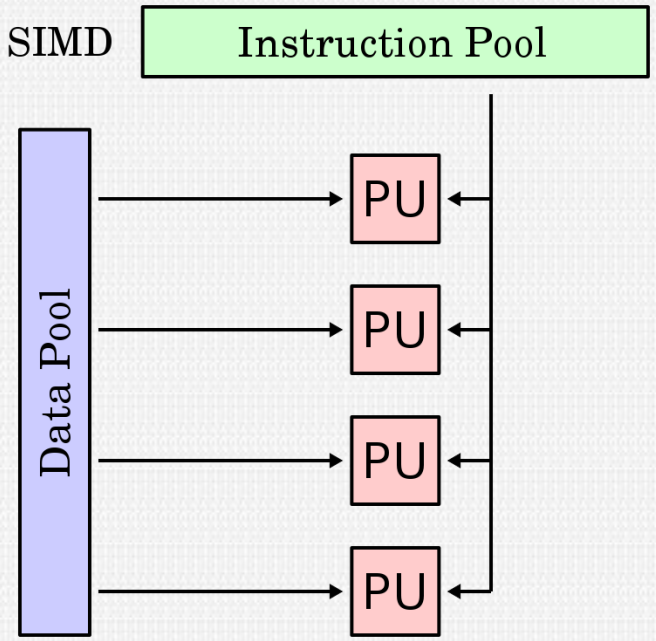
SISD

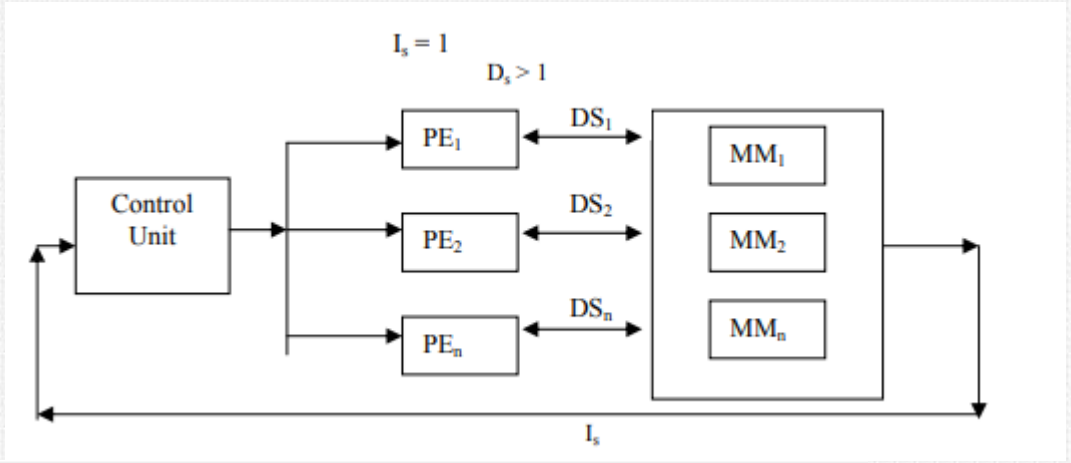

SISD
Instruction Pool
Data Pool
PU

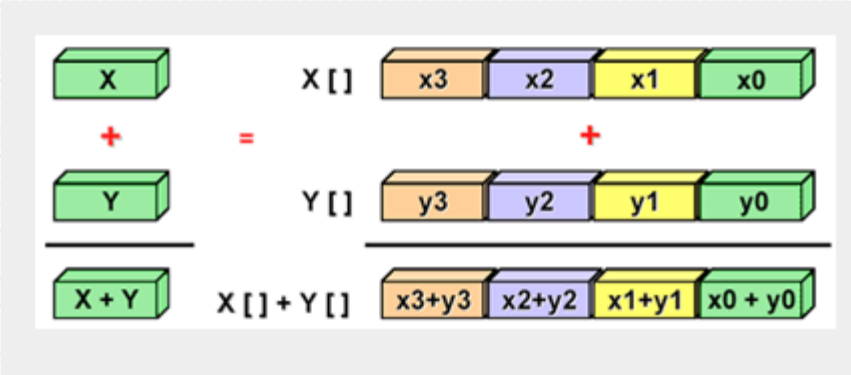

$I_s = D_s = 1$
Control Unit
I_s
ALU
D_s
Main Memory
I_s

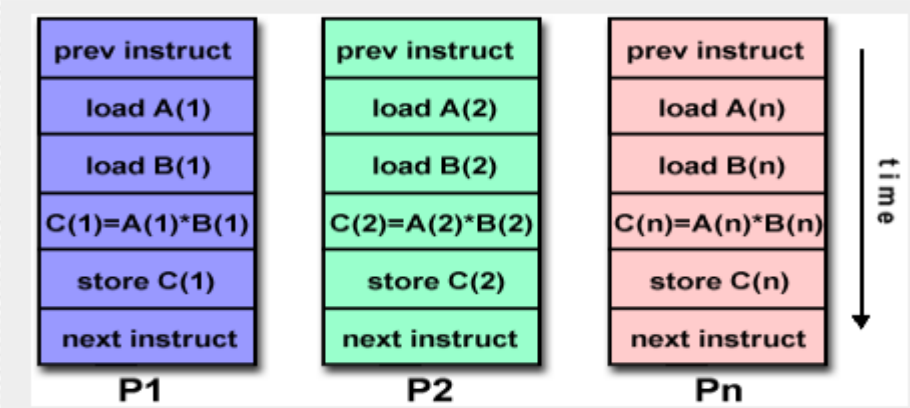

load A
load B
C = A + B
store C
A = B * 2
store A
time

SIMD

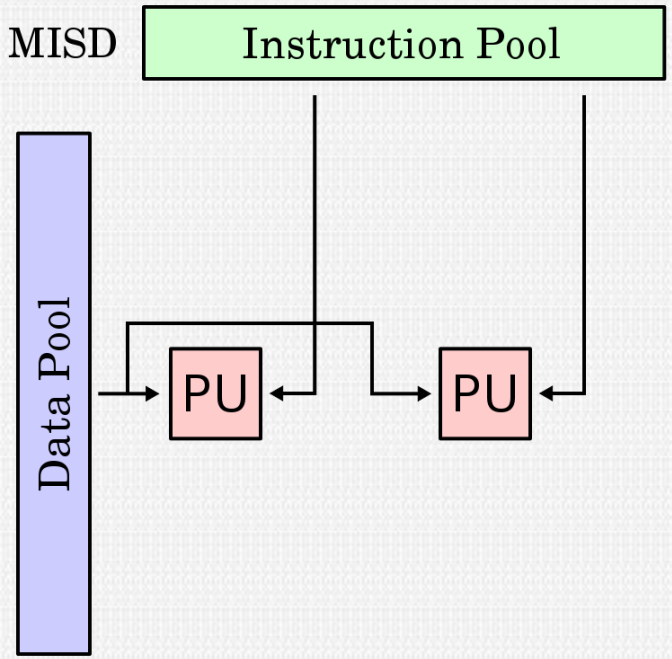

SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU

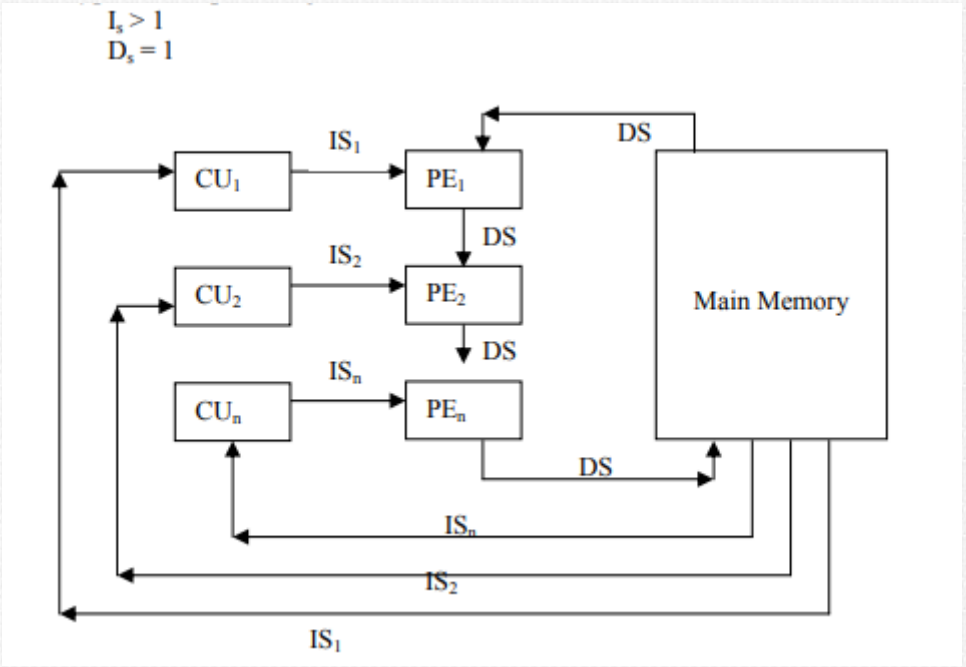

$I_s = 1$
$D_s > 1$
Control Unit
PE_1
PE_2
PE_n
DS_1
DS_2
DS_n
MM_1
MM_2
MM_n
I_s


X
Y
X + Y
X [ ]
Y [ ]
X [ ] + Y [ ]
x3
x2
x1
x0
y3
y2
y1
y0
x3+y3
x2+y2
x1+y1
x0 + y0

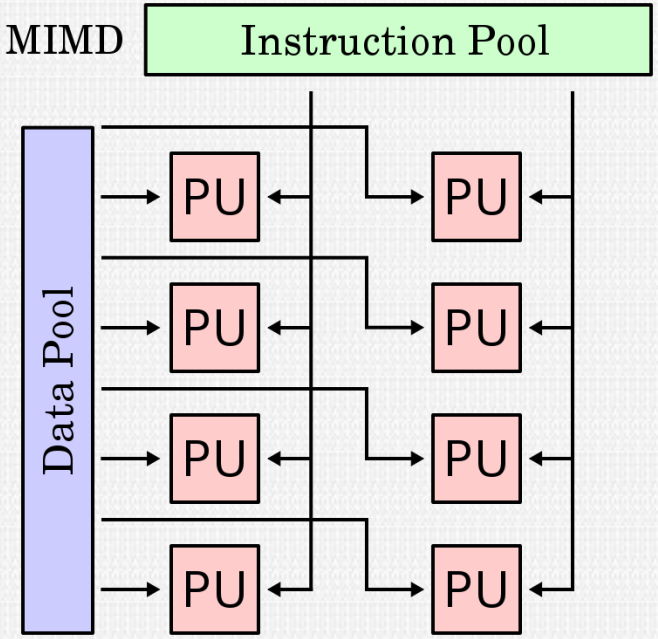

prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time

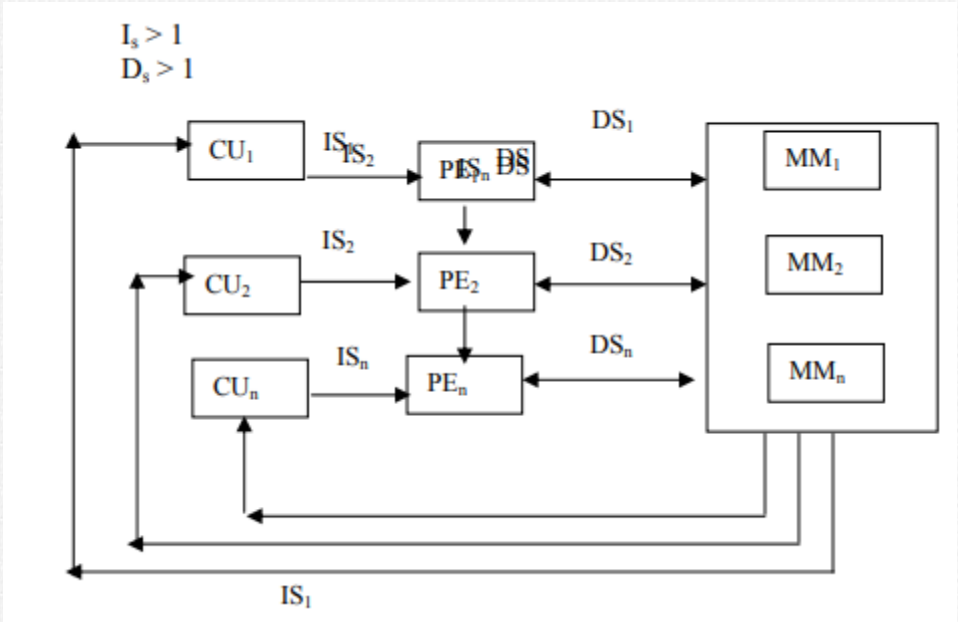
MISD


MISD
Instruction Pool
Data Pool
PU
PU

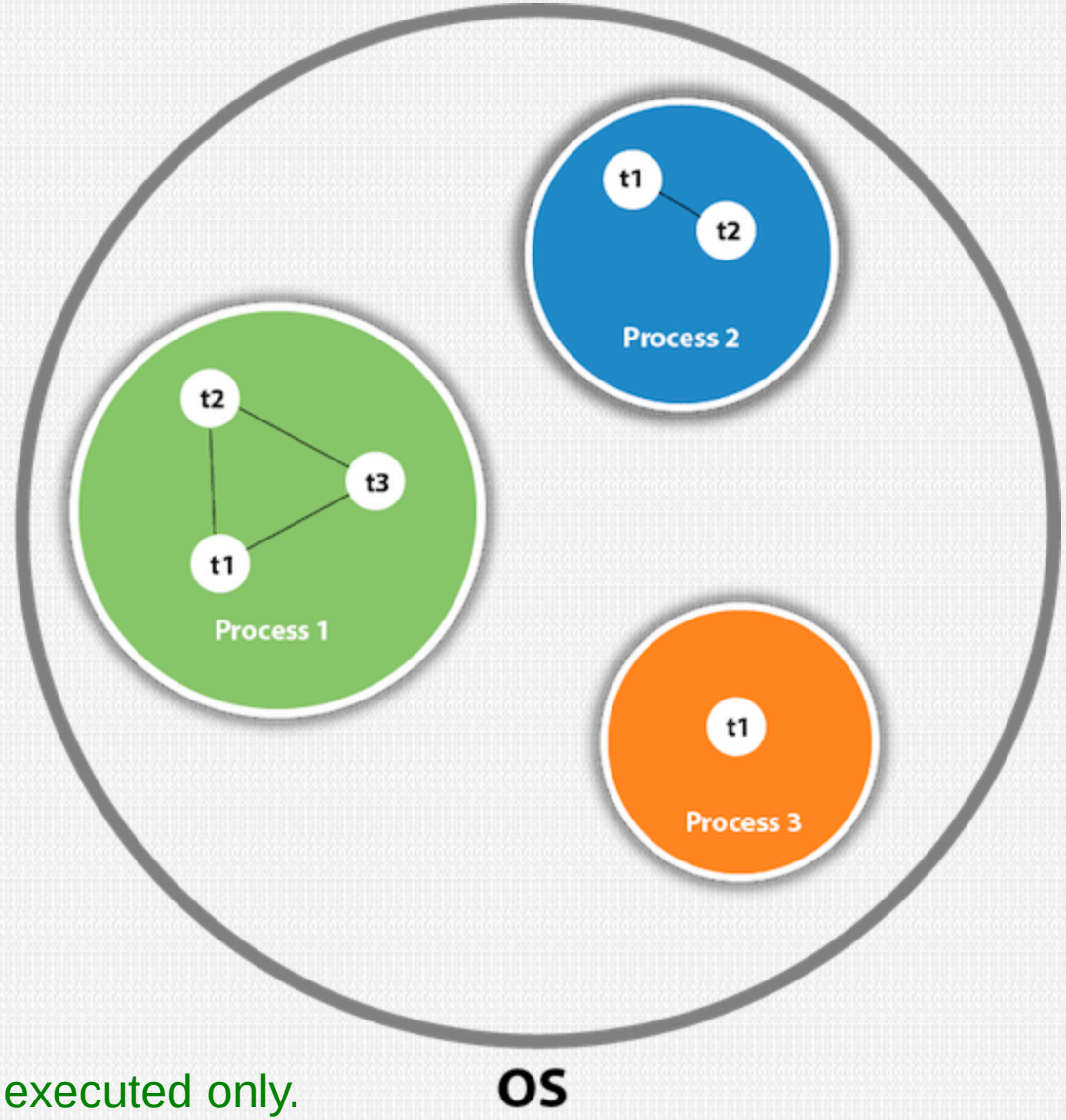

$I_s > 1$
$D_s = 1$
IS_1
CU_1
PE_1
DS
IS_2
CU_2
PE_2
DS
IS_n
CU_n
PE_n
DS
Main Memory
DS
IS_n
IS_2
IS_1

MIMD


MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU

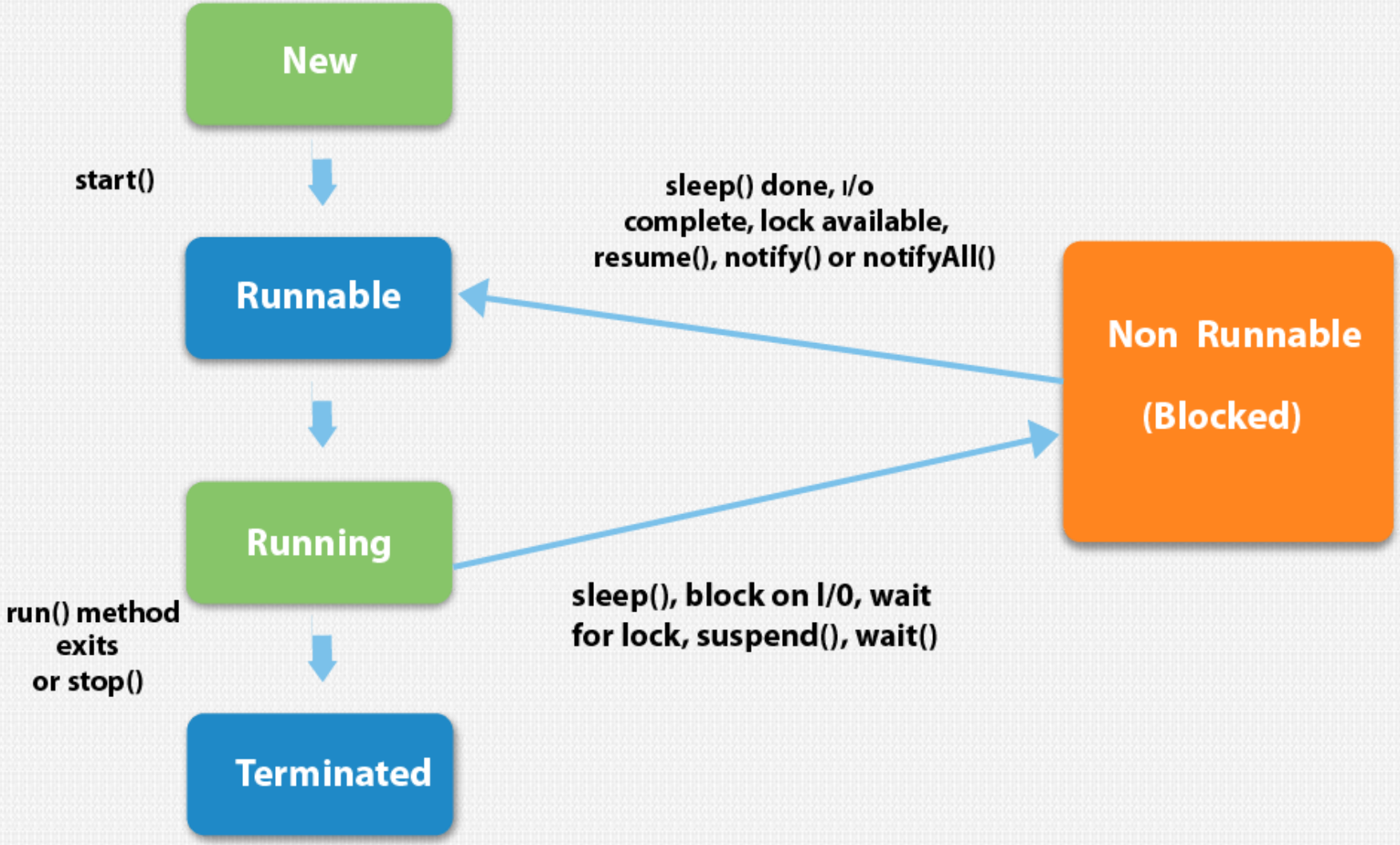

$I_s > 1$
$D_s > 1$
CU_1
IS_1
IS_2
PE_1
DS_1
MM_1
CU_2
IS_2
PE_2
DS_2
MM_2
CU_n
IS_n
PE_n
DS_n
MM_n
IS_1

Thread in Java



Note: At a time one thread is executed only.

- New
- Runnable
- Running
- Non-Runnable (Blocked)
- Terminated



1) Java Thread Example by extending Thread class

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

```
Output:thread is running...
```

2) Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

```
Output:thread is running...
```

Example of sleep method in java

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();
  t2.start();
 }
}
```

Output:

```
1
1
2
2
3
3
4
4
```

```
public class TestThreadTwice1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestThreadTwice1 t1=new TestThreadTwice1();
  t1.start();
  t1.start();
 }
}
```

Test it Now

```
running
Exception in thread "main" java.lang.IllegalThreadStateException
```

```
class TestCallRun1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestCallRun1 t1=new TestCallRun1();
  t1.run();//fine, but does not start a separate call stack
 }
}
```

Test it Now

```
Output:running...
```

Stack
(main thread)

Problem if you direct call run() method

```
class TestCallRun2 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestCallRun2 t1=new TestCallRun2();
  TestCallRun2 t2=new TestCallRun2();

  t1.run();
  t2.run();
 }
}
```

Test it Now

```
Output:1
       2
       3
       4
       5
       1
       2
       3
       4
       5
```

Example of join() method

```
class TestJoinMethod1 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try{
  t1.join();
 }catch(Exception e){System.out.println(e);}

 t2.start();
 t3.start();
 }
}
```