Лекція 6. Використання Java Collection Framework. Stream-API.

1. Колекції. Реалізація колекцій в Java.
2. Узагальнення. Реалізація Generic в Java.
3. Огляд ієрархії класів JCF.
4. Класи списків в Java.
5. Множини в JCF.
6. Відображення (словники).
7. Використання Java Stream API.

Деякі колекції Об’єктів

Vector - це здатний збільшувати число своїх елементів масив посилань на об'єкти. Всередині Vector реалізує стратегію динамічного розширення, що дозволяє мінімізувати невикористану пам'ять і кількість операцій з виділення пам'яті. Об'єкти можна або записувати в кінець об'єкта Vector за допомогою методу addElement, або вставляти в зазначену індексом позицію методом insertElementAt. Можна також записати в Vector масив об'єктів, для цього потрібно скористатися методом copyInto.

Методи Contains, indexOf, lastIndexOf, elementAt, firstElement і lastElement.

Деякі колекції Об’єктів

Stack - підклас класу Vector, який реалізує FIFO. На додаток до стандартних методів свого батьківського класу, Stack пропонує метод push для приміщення елемента в вершину стека і pop для вилучення з нього верхнього елементу. За допомогою методу peek ви можете отримати верхній елемент, не видаляючи його з стека. Метод empty служить для перевірки стека на наявність елементів - він повертає true, якщо стек порожній. Метод search шукає заданий елемент в стеці, повертаючи кількість операція pop, які потрібні для того щоб перевести шуканий елемент в вершину стека. Якщо заданий елемент в стеці відсутня, цей метод повертає -1.

Деякі колекції Об’єктів

Dictionary (словник) - абстрактний клас, що представляє собою сховище інформації типу "ключ-значення". Ключ - це ім'я, за яким здійснюється доступ до значення. Маючи ключ і значення, ви можете записати їх в словник методом put (key, value). Для отримання значення по заданому ключу служить метод get (key). І ключі, і значення можна отримати в формі перерахування (об'єкт Enumeration) методами keys і elements. Метод size повертає кількість пар "ключ-значення", записаних в словнику, метод isEmpty повертає true, якщо словник порожній. Для видалення ключа і пов'язаного з ним значення передбачений метод remove (key).

Arraylist

```
ArrayList array = new ArrayList();
        array.add(34);
        array.add("hello");
        array.add(new Cloneable(){});
array.add((array.get(2)).toString());

for (Object ob: array) {
System.out.println(ob);
}
int y = (int)array.get(0);
System.out.println(y);
```

Generics in Java

To update the Box class to use generics, you create a generic type declaration by changing the code

```
public class Box
```

to

```
public class Box<T>
```

This introduces the type variable, **T**, that can be used anywhere inside the class.

To instantiate this class, use the new keyword, as usual, but place <Integer> between the class name and the parenthesis:

```
Box<Integer> integerBox = new Box<Integer>();
```

Узагальнення

```
class Bank<T>{      }
public class MainClass {
public static void main(String[] args) {
    Bank<Integer> bank = new Bank(
        new Integer[] { 1, 2, 4, 5, 6 });
 Bank<String> bank2 = new Bank(
        new String[] {"13433", "342454", "21432"});
for (String s :bank2.clients) {
        System.out.println(s); }
Account[] accounts = new Account[] {
   new Account(1857), new Account(2225), new Account(33232)     };
Bank<Account> bank = new Bank(accounts);
bank.AccountsInfo(); }}
// з  обмеженнями на тип
class Bank<T extends IAccount>
```

Collections in Java

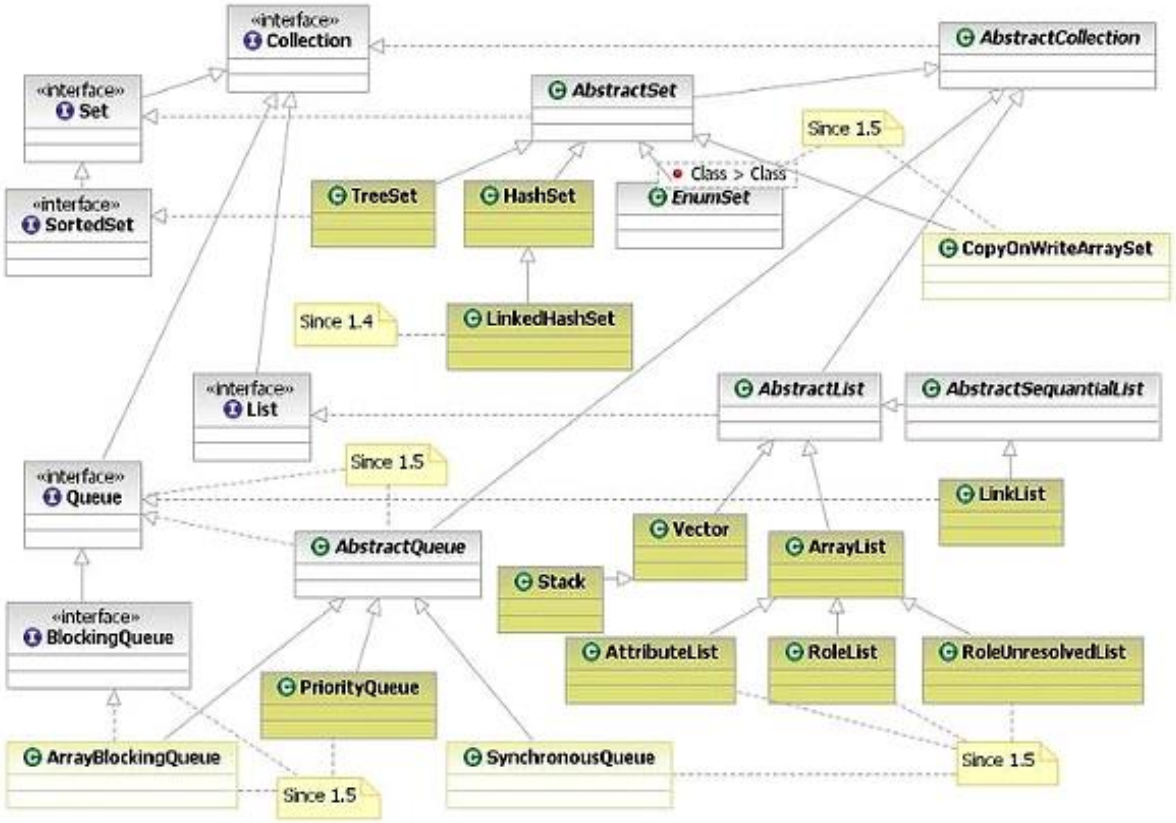
- Java Collection framework provides many interfaces (**`Set`**, **`List`**, **`Queue`**, **`Deque`**) and classes (**`ArrayList`**, **`Vector`**, **`LinkedList`**, **`PriorityQueue`**, **`HashSet`**, **`LinkedHashSet`**, **`TreeSet`**…).

- All collections frameworks contain the following:
  - interfaces
  - implementations
  - algorithms (there are the methods such as searching and sorting)

Java Collection Framework

- The **Collection** in Java is a framework that provides an architecture to ***store*** and ***manipulate*** the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as **searching**, **sorting**, **insertion**, **manipulation**, and **deletion**.
- Java Collection means a ***single unit of objects*** and its main advantage is ***grow as necessary***.
- Almost all collections and interfaces are **generics**.

jCF

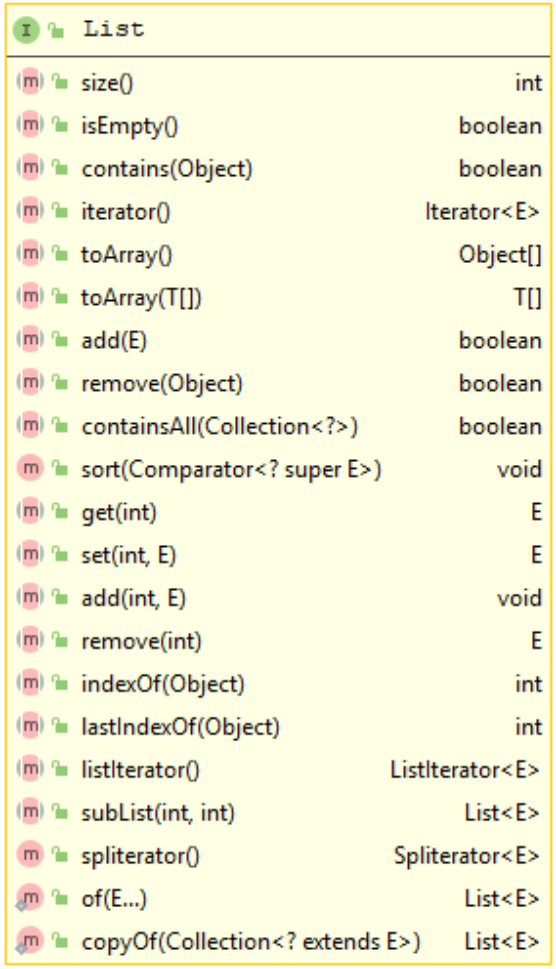

«interface»
Collection
AbstractCollection
«interface»
Set
AbstractSet
Since 1.5
Class > Class
«interface»
SortedSet
TreeSet
HashSet
EnumSet
CopyOnWriteArraySet
Since 1.4
LinkedHashSet
«interface»
List
AbstractList
AbstractSequantialList
«interface»
Queue
Since 1.5
LinkList
Vector
AbstractQueue
ArrayList
Stack
«interface»
BlockingQueue
AttributeList
RoleList
RoleUnresolvedList
PriorityQueue
ArrayBlockingQueue
SynchronousQueue
Since 1.5
Since 1.5

Interfaces

- **List** – a list of objects. Objects can be added to the list (the method add()), replace the list (method set()), removed from the list (the method remove()), extract (method get()). There is the ability to pass on the list of organizations with an *iterator*.
- **Set** – a set of objects. The same features as that of the List, but the object can be part of set only once. Double addition of one and the same object in the set is not change the set.
- **Map** – a map or associative array. In Map we add pair of objects (key, value). Accordingly, the search operation is a key value. Adding a couple with an existing key in the Map leads to the replacement, not to upload it. From Map can be obtain key and a list of values.

Interface List

- A **Lists** are **Ordered Collections** (sometimes called a ***sequence***).
- Lists can contain ***duplicate elements***.
- The user of a List generally has ***precise control*** over where in the list each element is inserted and can access elements by their **integer index** (position).



Interface List

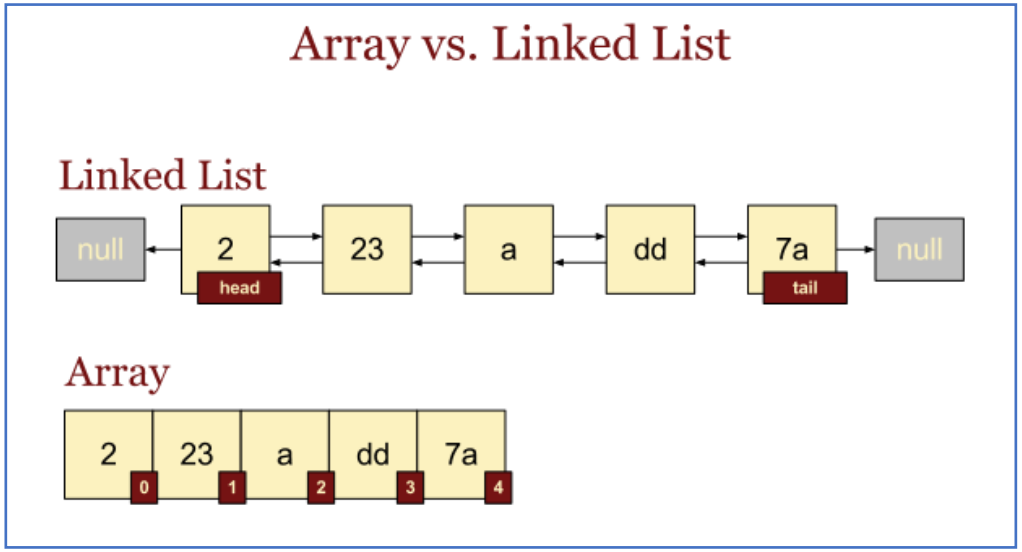
- Since List is an interface, you need to instantiate a concrete implementation of the interface in order to use it.
- There are two general-purpose List implementations – java.util.ArrayList and java.util.LinkedList

`listArr` and `listLink` have objects of type Object

Here are a few examples of how to create a List instance:

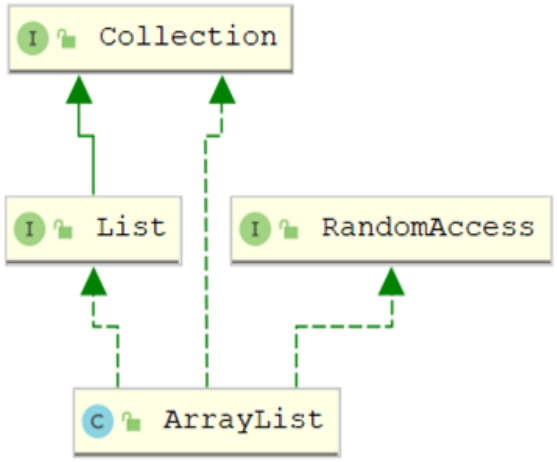
```
List<> listArr = new ArrayList(); or List listArr = new ArrayList(15);
List listLink = new LinkedList();
```

Interface List



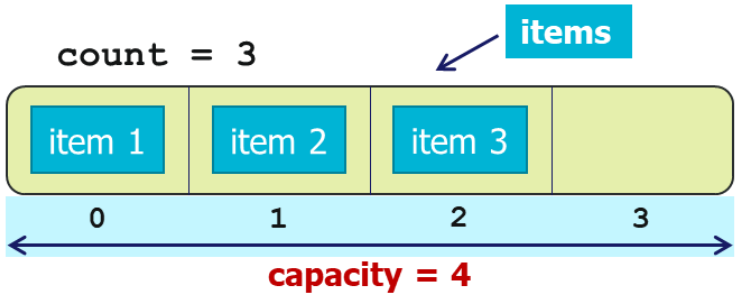
ArrayList is a list implemented based on an array, while **LinkedList** is a classic linked list based on objects with links between them

Class **`ArrayList`**



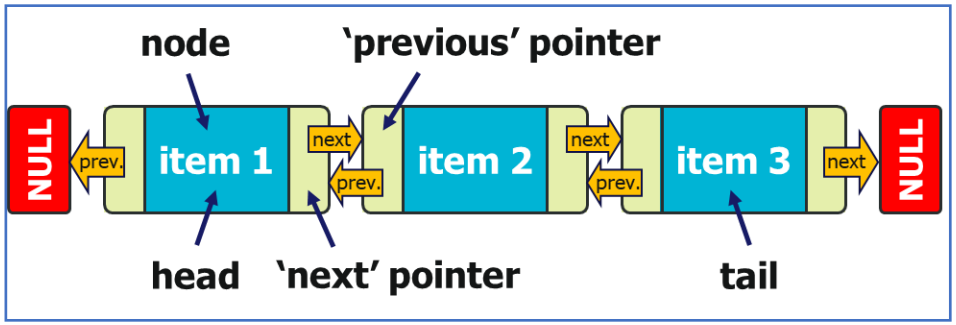
- **Array Lists** use a ***dynamic array*** to store the duplicate element of ***different data types***
- The Array List maintains the ***insertion order***
- The elements stored in the Array List can be ***randomly accessed***
- Capacity of ArrayList by default capacity = 10

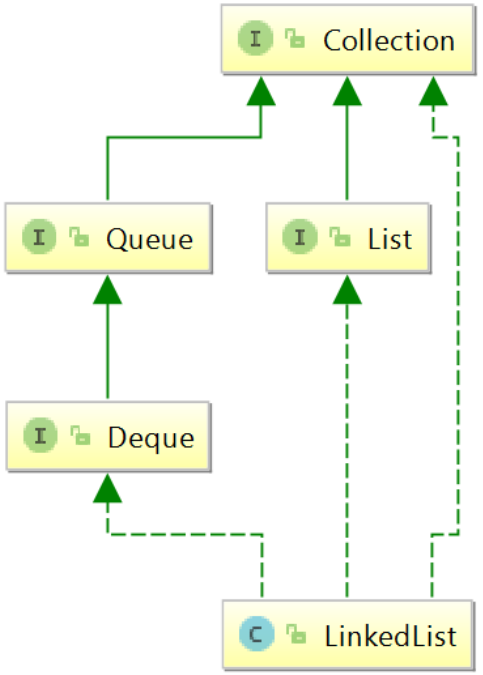
  For new arrayList capacity = (oldCapacity * 3) / 2 + 1



Class `LinkedList`

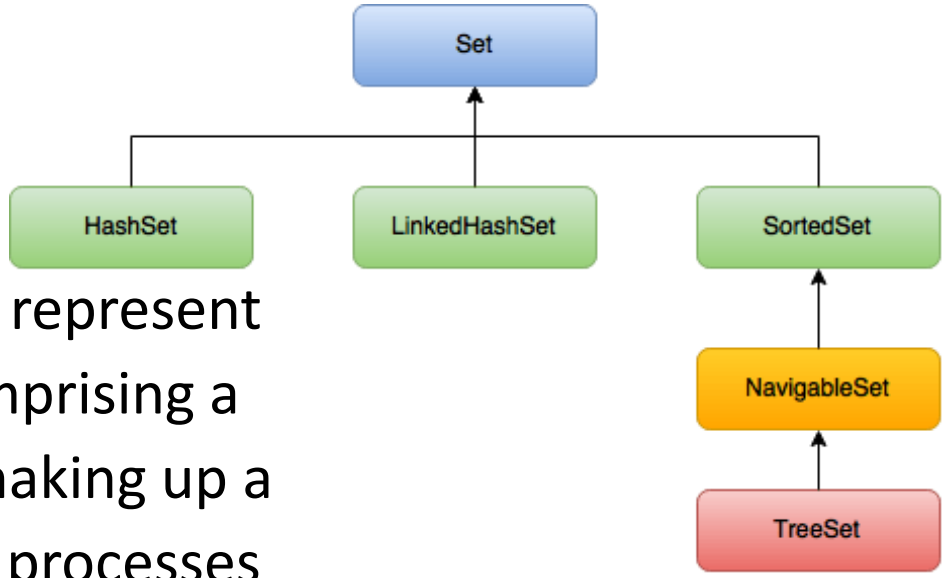
- **Linked Lists** use a ***references*** to maintain an ordered lists of "**nodes**".
- The "**head**" of the list references the first node.
- Each node has a ***value*** and a ***reference*** to the ***next*** and ***previous node***.

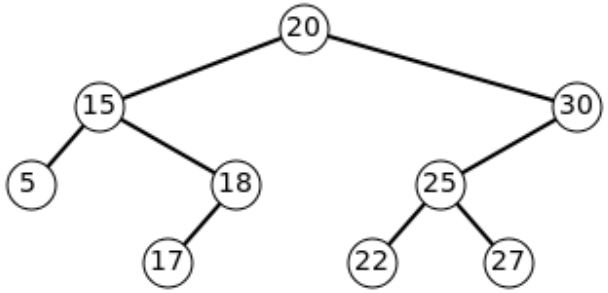




Interface Set

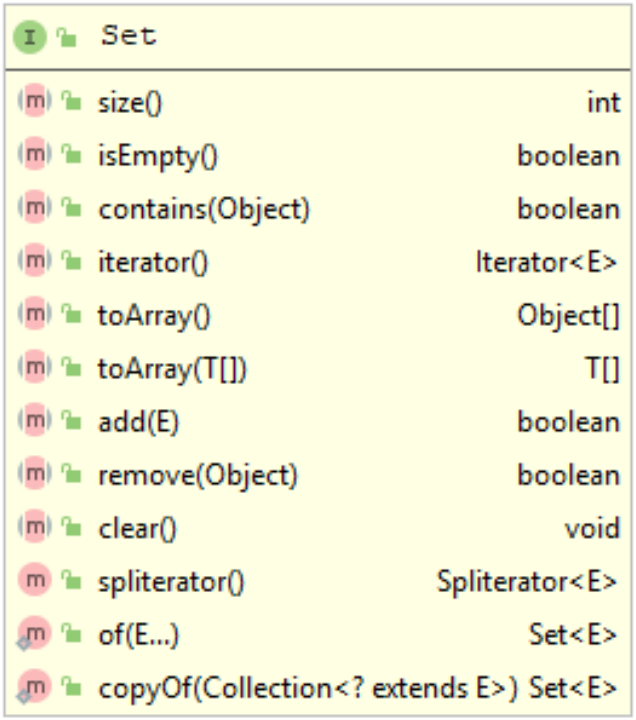
- The Set interface defines a set of elements (Set)
- This interface models the ***mathematical set abstraction*** and is used to represent sets, such as the cards comprising a poker hand, the courses making up a student's schedule, or the processes running on a machine.
- A **Sets** are **Unordered Collection** that ***cannot contain duplicate elements***.





Interface Set

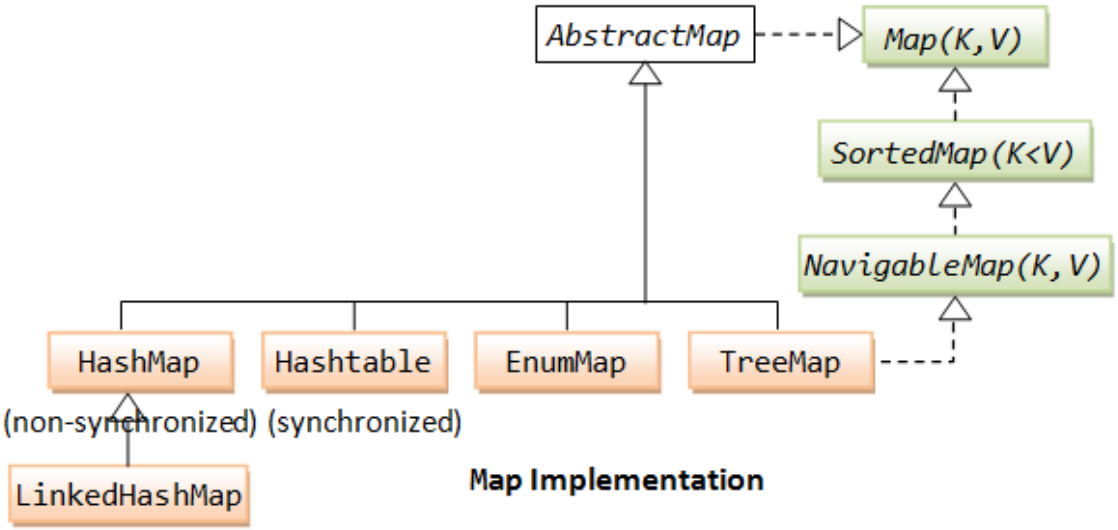
- The interface extends from **Collection** and does not define any additional methods of its own.
- Uniqueness is determined by the implementation of the **equals()** method - an override of the **equals()** method is required



TreeSet and **HashSet** are most popular implementation of Set interface

Interface Map

- The **uniqueness** of the keys is determined by the implementation of the **equals()** method. To work correctly with maps, you must override the **equals()** and **hashCode()** methods



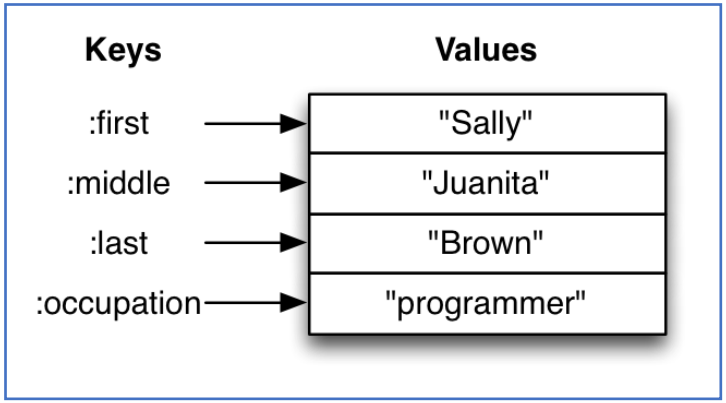
The Map interface does not extend the Collection interface

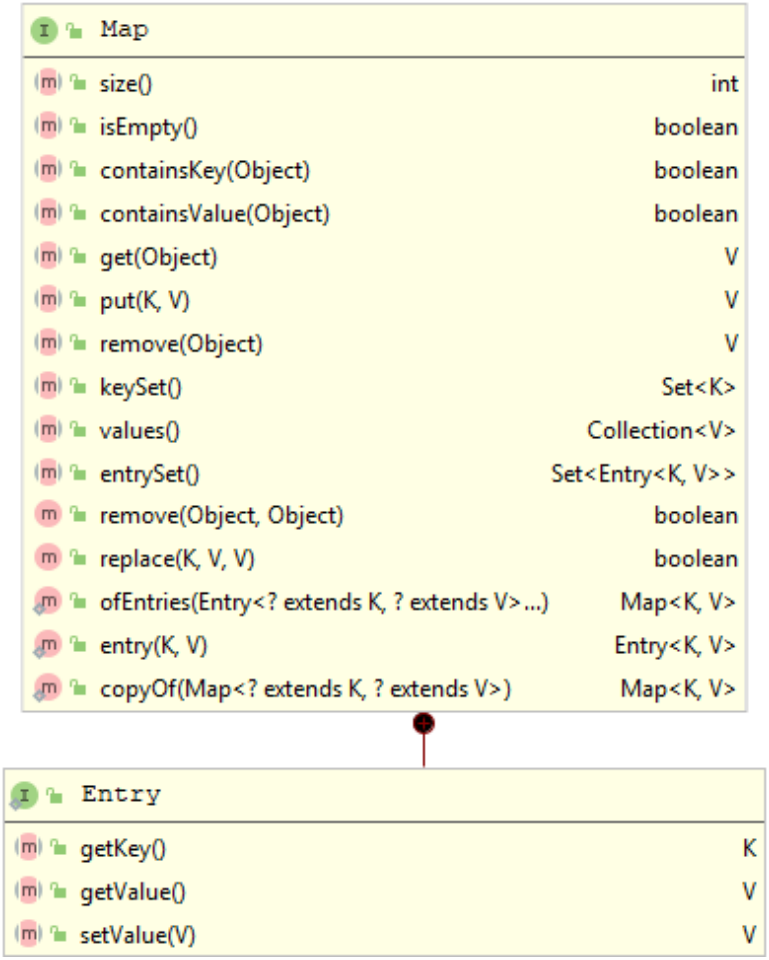
- The **Map** is an object that **maps keys** to **values**
- Map cannot contain **duplicate keys** and each key **can map** to at **most one value**
- Both key and value are objects

Interface Map

A Map stores ***keys***, ***values***, and the ***association between them***.

- **Keys** - provides an easy way to represent an object
- **Values** - the actual object that is associated with the key





The most commonly used Map implementations are **HashMap** and **TreeMap.**

Class HashMap

HashMap class inherits the AbstractMap class and implements the Map interface which allows to store key and value pair, where **keys** should be **unique.**

```
public class Program {
        public static void main(String[] args) {
                Map<Integer, String> hashMap = new HashMap<>();

                 //add elements to the map
                hashMap.put(1, "Mike");
                hashMap.put(2, "Nick");
                hashMap.put(3, "Sara");

                //get object by key 2
               String element = hashMap.get(2);
               System.out.println("Element by key 2: ” + element);

               //returns a Set containing all map values
              Set<Integer> keys = hashMap.keySet();
```

Stream-API

Посинаючи з JDK 8 в Java запровадили Stream API з метою вдосконалити роботу з наборами даних, зокрема, операції фільтрації, сортування та інші маніпуляції з даними. Функціональність даного API міститься в пакеті java.util.stream.

Поточний інтерфейс Stream API допускає використання лямбда-виразів, які покращують запис дій з даними

В порівнянні з C # та .Net функціонал Stream-API можна вважати приблизним аналогом технології LINQ.

Stream-API

Ключовим поняттям Stream API є потік даних. Дані для потоку можуть зберігатися в колекції, масиві, файлі або бути згенеровані програмно. При цьому важливо розуміти відмінності колекцій від потоків:

- потоки не зберігають елементів;
- операції з потоками не змінюють джерела даних;
- для потоку характерним є відкладене виконання.

Створення потоку

1. З колекції

```
Collection<String> collection =
                                Arrays.asList("a1", "a2", "a3");
        Stream<String> streamFromCollection =
                                        collection.stream();
```

2. З набору значень

```
Stream<String> streamFromValues =
                                 Stream.of("a1", "a2", "a3");
```

Створення потоку

3. З масиву

```
String[] array = {"a1","a2","a3"};
Stream<String> streamFromArrays =
                                    Arrays.stream(array);
```

4. З файлу

```
Stream<String> streamFromFiles =
                                Files.lines(Paths.get("file.txt"))
```

Створення потоку

5. Масив символів зі стрічки

```
IntStream streamFromString = "123".chars();
```

6. За допомогою build-ера

```
Stream.builder().add("a1").add("a2").add("a3").build();
```

7. Паралельний стрім

```
Stream<String> stream = collection.parallelStream();
```

Створення потоку

8. Нескінченний потік за допомогою ітерації

```
Stream<Integer> streamFromIterate =
                                    Stream.iterate(1, n -> n + 1);
```

9. Нескінченний потік за допомогою генерації

```
Stream<String> streamFromGenerate =
                                    Stream.generate(() -> "a1");
```

Методи для опрацювання потоку

Java Stream API пропонує два види методів:
1. Конвеєрні - повертають інший stream, тобто працюють як builder,
2. Термінальні - повертають інший об'єкт, такий як колекція, примітиви, об'єкти, Optional і т.д.
У stream'а може бути скільки завгодно викликів конвеєрних викликів і в кінці один термінальний, при цьому всі конвеєрні методи виконуються ліниво і поки не буде викликаний термінальний метод ніяких дій насправді не відбувається

Методи для опрацювання потоку

Приклади конвеєрних методів:

filter, skip, distinct, map, peek, limit, sorted, mapToInt, flatMap, flatMapToDouble

Приклади термінальних методів:

findFirst, findAny, collect, count, anyMatch, min, max, forEach, forEachOrdered, toArray