CPE207 Object Oriented Programming

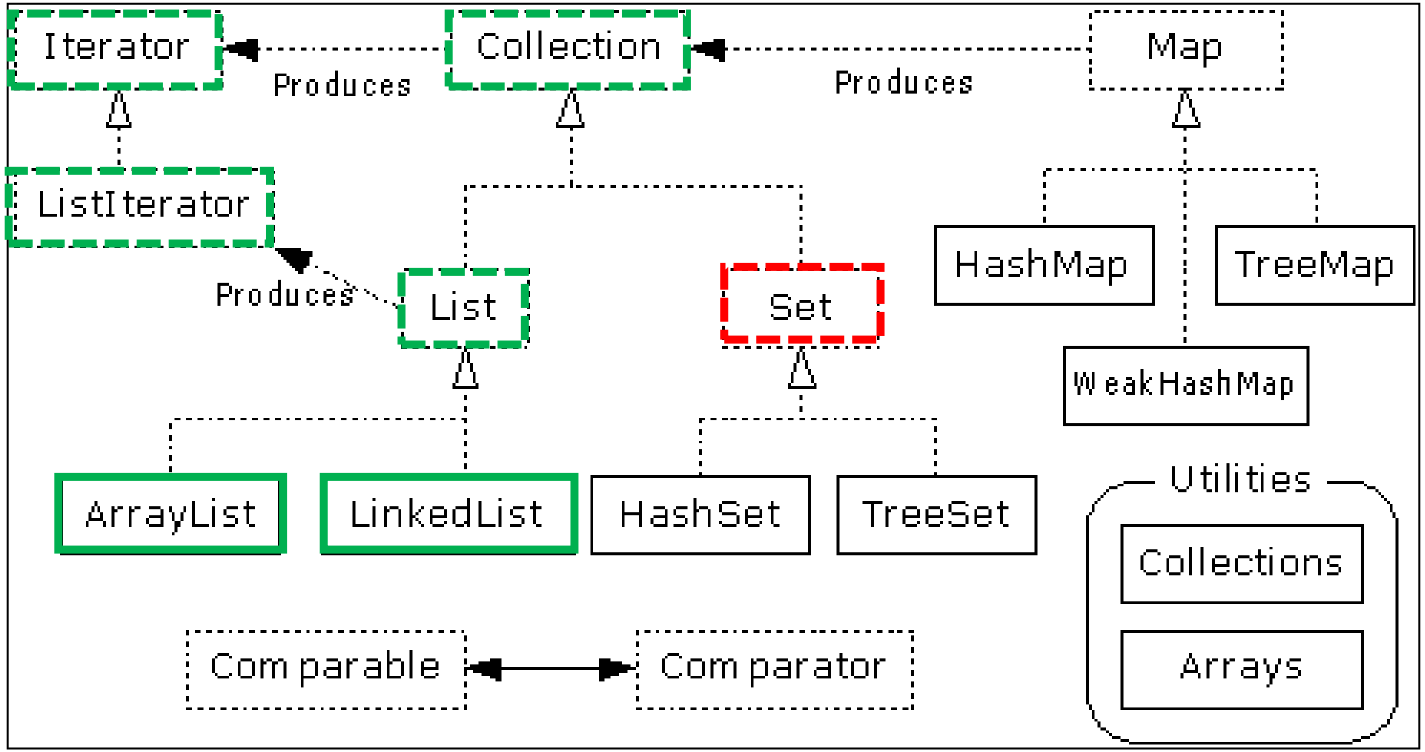
Week 13

Collections: Set & Map
Comparable & Comparator
File operations: Write/Read objects

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

These Slides mainly adopted from Assist. Prof. Dr. Ozacar Kasim lecture notes

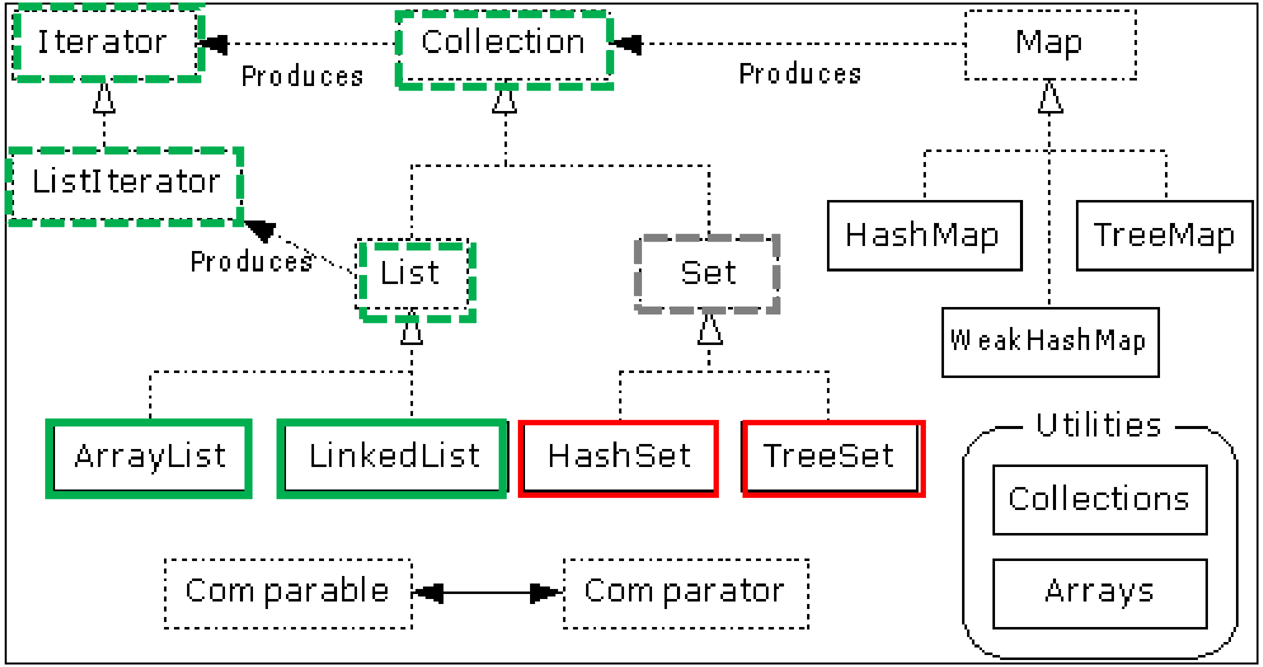
Set Interface



Set Interface

- Same methods as Collection
  - different contract – no duplicate entries
- Defines two fundamental methods
  - `boolean add(Object o)` – **reject duplicates**
  - `Iterator iterator()`
- Provides an Iterator to step through the elements in the Set
  - No guaranteed order in the basic Set interface

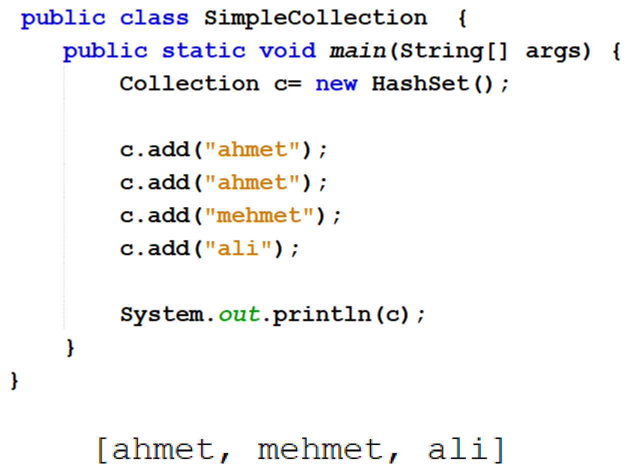
HashSet and TreeSet



```
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

HashSet

- Find and add elements very quickly
  - uses hashing implementation in HashMap
- Hashing uses an array of linked lists
- **HashSet doesn’t maintain any order.**
- HashSet internally uses HashMap for storing the data and retrieving it back.



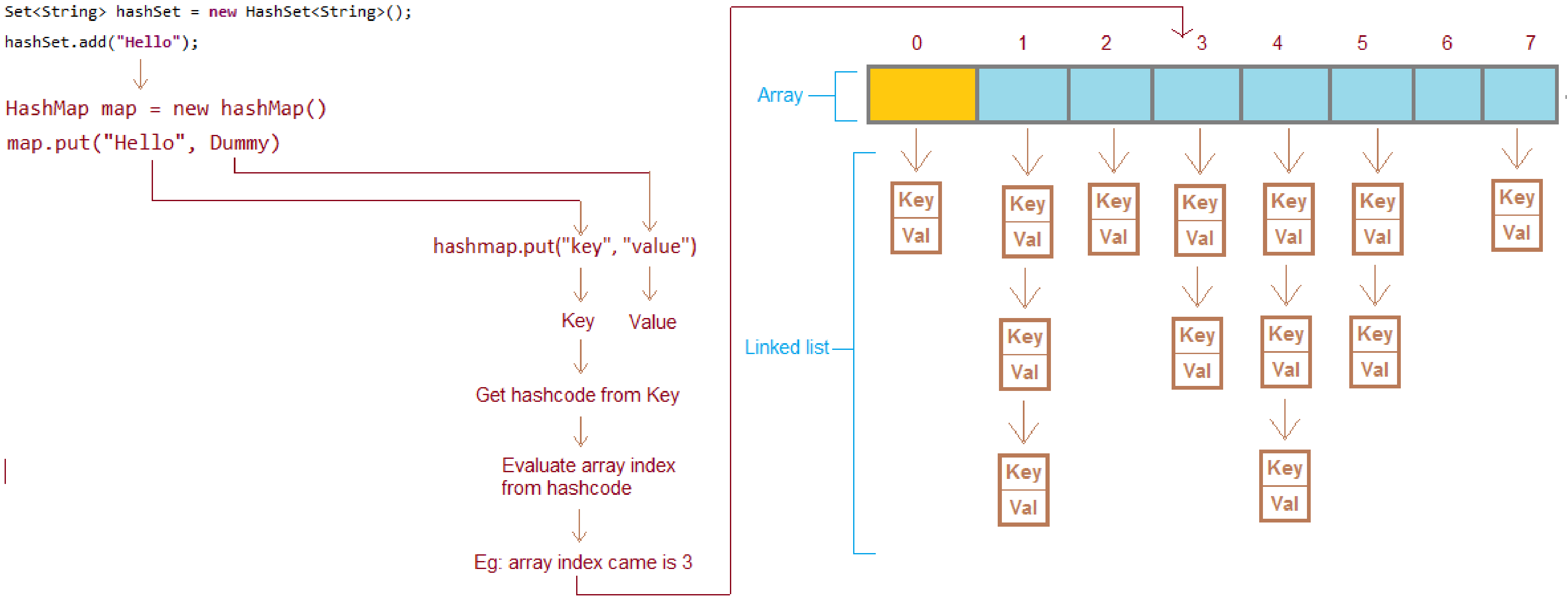
```java
public class SimpleCollection  {
   public static void main(String[] args) {
       Collection c= new HashSet();

       c.add("ahmet");
       c.add("ahmet");
       c.add("mehmet");
       c.add("ali");

       System.out.println(c);
   }
}
```

```
[ahmet, mehmet, ali]
```

HashSet

HashSet Internal Architecture

HashMap Internal Architecture



```
Set<String> hashSet = new HashSet<String>();
hashSet.add("Hello");
```

```
HashMap map = new hashMap()
map.put("Hello", Dummy)
```

hashmap.put("key", "value")
Key
Value
Get hashcode from Key
Evaluate array index from hashcode
Eg: array index came is 3
Array
0
1
2
3
4
5
6
7
Linked list
Key
Val

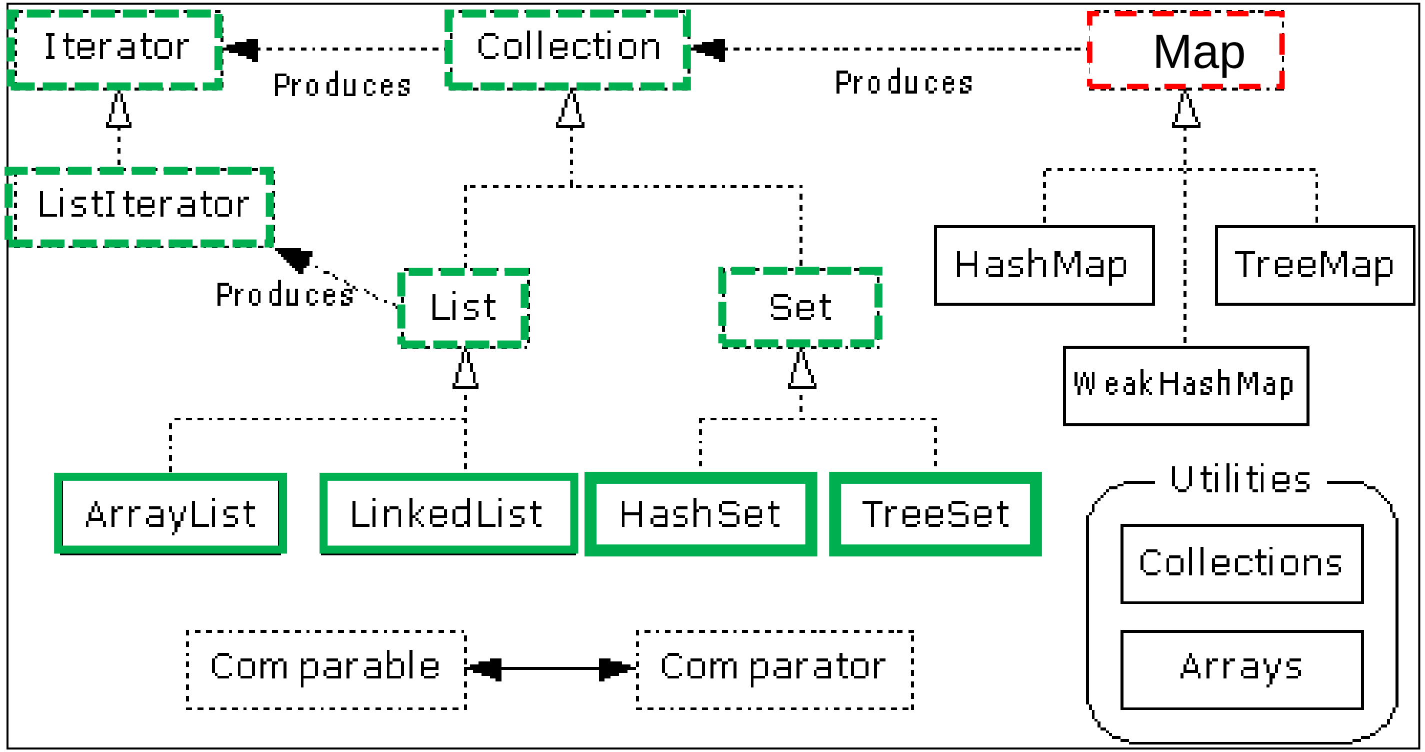
TreeSet

- Elements can be inserted in any order
- **The TreeSet stores them in order**
- An iterator always presents them in order
- Default order is defined by natural order

```java
public class SimpleCollection  {
    public static void main(String[] args) {
        Collection c= new TreeSet();

        c.add("ahmet");
        c.add("ahmet");
        c.add("mehmet");
        c.add("ali");

        System.out.println(c);
    }
}
```

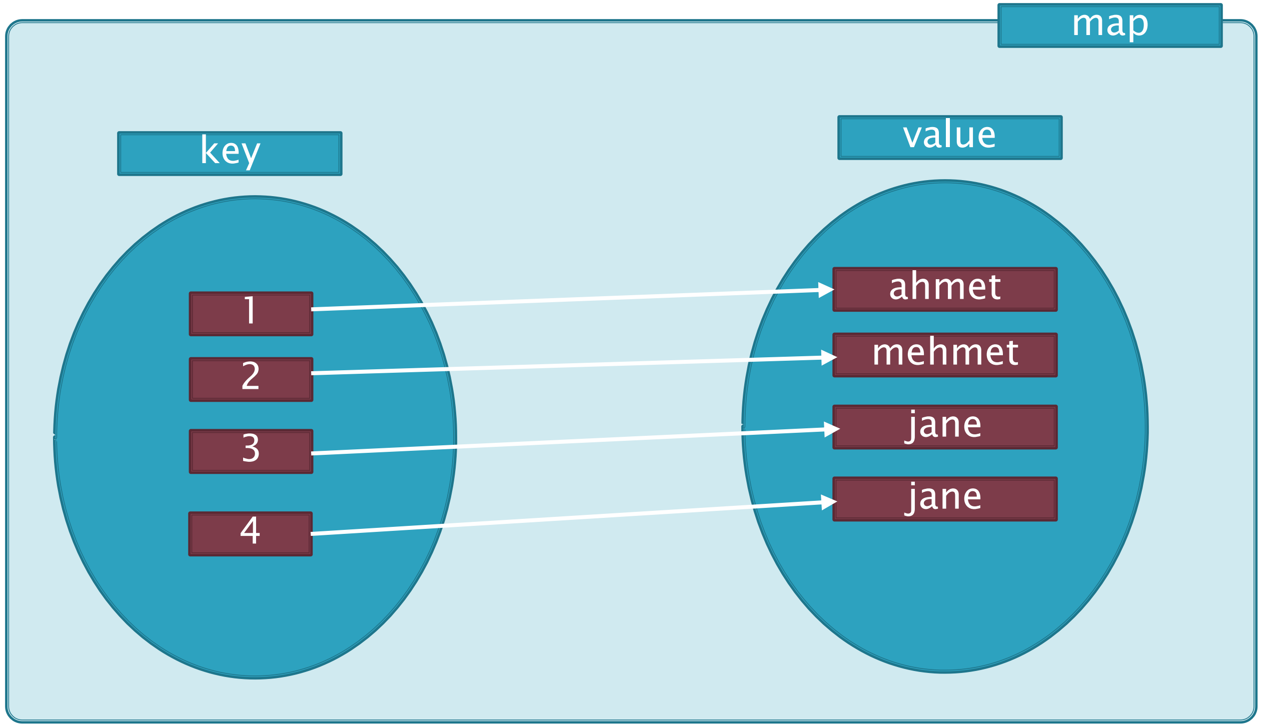
```
[ahmet, ali, mehmet]
```

Map Interface Context



Map Interface

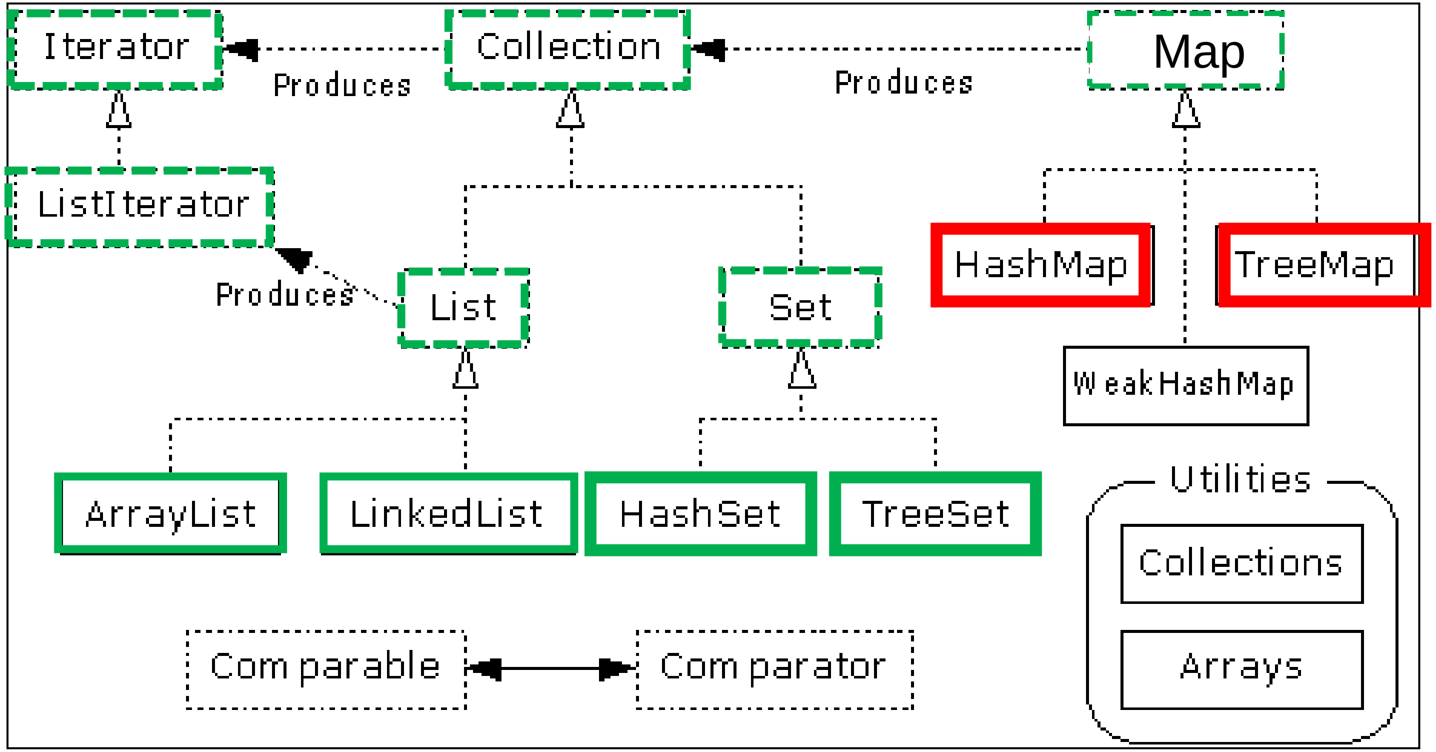
- Stores key/value pairs
- Maps from the key to the value
- Keys are unique
  - a single key only appears once in the Map
  - a key can map to only one value
- Values do not have to be unique


map
key
value
1
2
3
4
ahmet
mehmet
jane
jane

Map methods

```
Object put(Object key, Object value)
Object get(Object key)
Object remove(Object key)
boolean containsKey(Object key)
boolean containsValue(Object value)
int size()
boolean isEmpty()
```

HashMap and TreeMap



HashMap and TreeMap

- HashMap
  - The keys are a set – unique, unordered
  - Fast
- TreeMap
  - The keys are a set – unique, ordered

When to use List, Set and Map in Java?

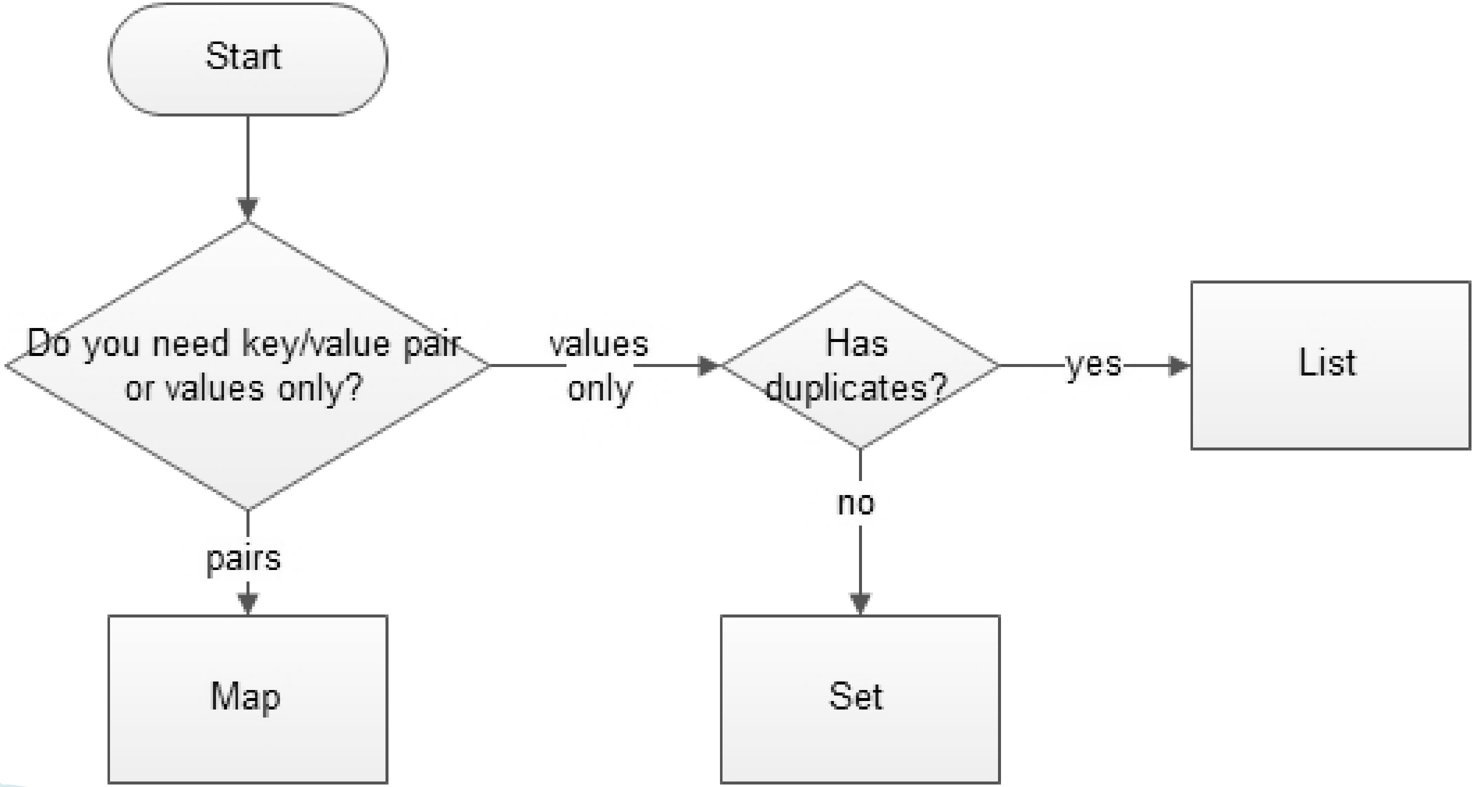
1) If there is a need of frequent search operations based on the index values (random access) then List (ArrayList) is a better choice. O(1)

2) If there is a need of maintaining the insertion order, then also the List(LinkedList) is a preferred collection interface.

3) If you do not want to have duplicate values in the database, then Set should be your first choice as all its classes do not allow duplicates.

4) If the requirement is to have the key & value mappings in the database, then Map is your best bet.

Choose the Right Java Collection



Comparing objects

- The == operator can be used to check if two object references point to the same object.

```
if (objRef1 == objRef2) {
    // The two object references point to the same object
}
```

- To be able to compare two Java objects of the same class the boolean **equals(Object obj)** method must be overridden and implemented by the class.

- The implementor decides which values must be equal to consider two objects to be equal. For example in the below class, the name and the address must be equal but not the description.

Example

```
class Worker {
        private String name;
        private int age;
        private int weight;

        public Worker(String name, int age, int weight) {
            this.name = name;
            this.age = age;
            this.weight = weight;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
               if (o == null || getClass() != o.getClass()) return false;
                Worker worker = (Worker) o;
             return age == worker.age && weight == worker.weight && name.equals(worker.name);
        }
    }
```

equals() versus ==

- the == operator and equals() method may appear to do the same thing, but in truth they work differently. The == operator compares whether two object references point to the same object. For example:

```
Worker w1 = new Worker("Homer", 35, 120)
Worker w2 = new Worker("Homer", 35, 120)

System.out.println(w1 == w2);            //false
System.out.println(w1.equals(w2)); //true
```

Sorting/Ordering objects: Comparable Interface

- Java Comparable interface is used to **<u>order the objects of the user-defined class</u>**
- you can sort the elements based on single data member only such as rollno, name, age or anything else.

- **compareTo(Object obj) method**
- compares 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.

Comparable: Example

```
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{
    return age – st.age;
   }
public void toString(){
  return st.rollno+" "+st.name+" "+st.age
  }
}
```

```
public static void main(String args[]){

ArrayList<Student> students=new ArrayList<Student>();

students.add(new Student(101,"Vijay",23));
students.add(new Student(106,"Ajay",27));
students.add(new Student(105,"Jai",21));

Collections.sort(students);

for(Student st: students){
  System.out.println(st);
  }
}
```

Change Sorting/Ordering: Comparator Interface

- Sometimes we may want to change the ordering of a collection of objects from the same class. We may want to order descending or ascending order. We may want to sort by name or by score, price etc.
- We need to create a class for each way of ordering. It must implement the Comparator interface.
- **Java Comparator interface** is used to order the objects of a user-defined class.
- **int compare(Object o1, Object o2)** 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.

Comparator Example

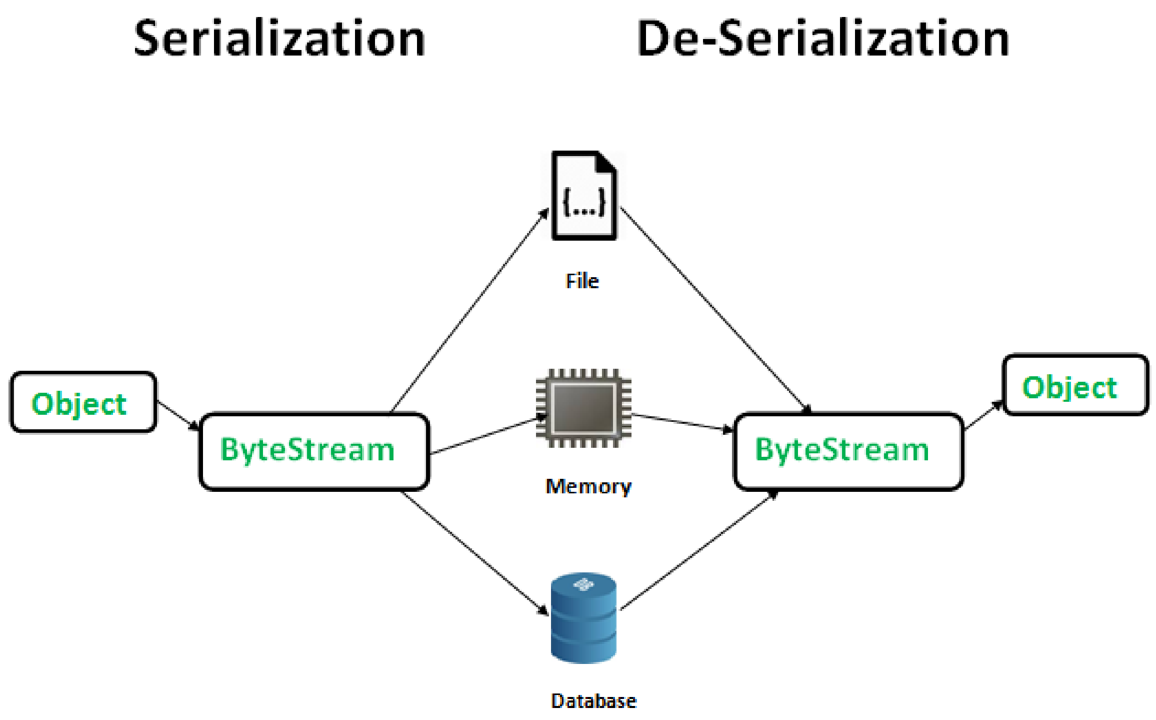
```
class AgeComparator implements Comparator<Student>{
public int compare(Student  s1, Student  s2){
 return s1.age-s2.age;   }

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

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

Usage

```
public static void main(String args[]){

    ArrayList al=new ArrayList();

    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());
        Iterator itr=al.iterator();

        while(itr.hasNext()){
           Student st=(Student)itr.next();
           System.out.println(st.rollno+" "+st.name+" "+st.age);   }

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

        Collections.sort(al,new AgeComparator());
        Iterator itr2=al.iterator();

        while(itr2.hasNext()){
           Student st=(Student)itr2.next();
           System.out.println(st.rollno+" "+st.name+" "+st.age);   }
}
```

Saving Objects/ Loading objects



Saving objects to File

```
 FileOutputStream fileOut = new FileOutputStream(filepath);
//FileOutputStream class is an output stream for writing data to a File

 ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
//The Java ObjectOutputStream class (java.io.ObjectOutputStream) enables you
to write Java objects to an OutputStream instead of just raw bytes

 objectOut.writeObject(object);
//writes object to a file
```

Get saved objects from a file

```
FileInputStream fileIn = new FileInputStream(filepath);
 ObjectInputStream objectIn = new ObjectInputStream(fileIn);
 YourType obj = (YourType)objectIn.readObject();
```

Save and Load

```
public static  void saveToFile(YourDataType s, String path) throws Exception{
       FileOutputStream fileOut = new FileOutputStream(path);
       ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
       objectOut.writeObject(s);
       objectOut.close();
   }

   public  static YourDataType getFromFile(String path)throws Exception{
       FileInputStream fileIn = new FileInputStream(path);
       ObjectInputStream objectIn = new ObjectInputStream(fileIn);
       YourDataType s = (YourDataType)objectIn.readObject();
       objectIn.close();
       return s;
   }
```

Thanks ☺