Collections

Collections

- What is a collection?
- Collections and data structures
- Java implementation of collections - java.util.Collection
- Iterators
- Comparing

Overview of arrays

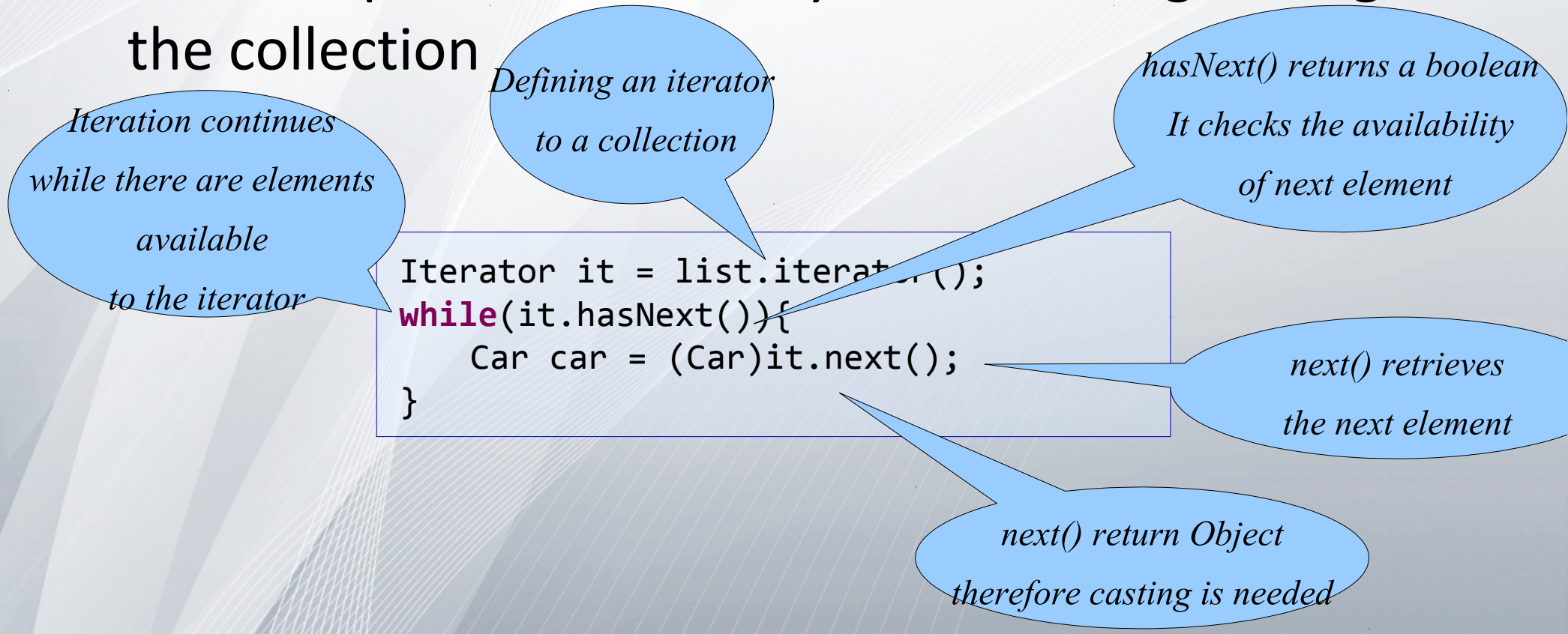
- Disadvantages
  - Arrays have a fixed size/length
  - Arrays can store objects/primitives of one type only
  - “Static” structure – can't change their size
- Advantages
  - Accessibility to every member of the array
  - Can hold more than one object/primitive

Basic Collections

- Dynamic data structure – can change its size
- Contains objects
  - Primitives can also be stored or retrieved
- Basic collection types:
  - List - java.util.List
  - Set - java.util.Set
  - Stack and Queue
  - Map – java.util.Map

Iterators

- Iterators are individual objects for each collection object
- Iterators provide the ability for traversing through the collection



List

- List features
  - Can add/remove new objects at any part of the list
  - Can hold equal object
  - Direct access to every object in the list
- java.util.List
- Most common implementations:
  - LinkedList
  - ArrayList
  - Vector

List

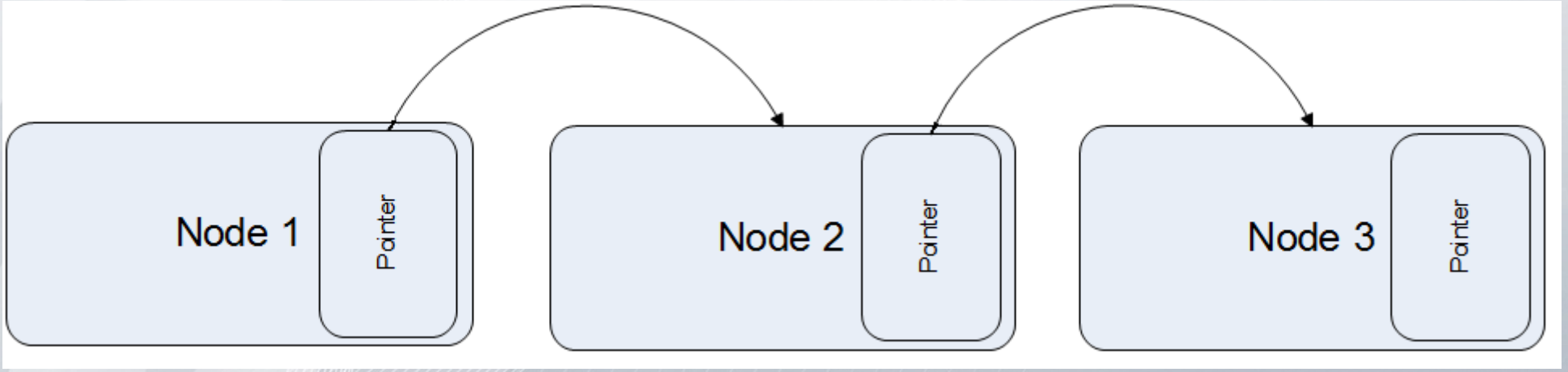
- Basic methods
  - add()
  - get()
  - clear()
  - remove()
  - contains()
  - toArray()

ArrayList

- Class which contains an array of objects
- Each time a new object is being added to the list a check for the array length is made. If there isn't sufficient space, a new larger array is created and the elements of the old one are copied into the new one.
- ensureCapacity(int n) – ensures capacity of n

LinkedList

- Each node of the structure contains a pointer which points to next node of the structure
- Not Synchronized



How to use Lists in Java

- Constructor
  - List arrayList = new ArrayList();
  - List linkedList = new LinkedList();
- Add object
  - list.add(o1); - Adds object at the end of the list
  - list.addFirst(o1) – adds an object at the beginning
- Remove object
  - list.remove(o1) – removes object
  - list.remove(3) – removes the 4^{th} element

How to use Lists in Java

- Contain object
  - list.contains(o1);
- Iterator
  - list.iterator();
- Get an Object
  - list.get(3) – gets the 4th element. Returns an object and casting is needed
  - iterator.next() - Returns an object and casting is needed

Comparing

- Java objects can be compared
  - equals() or „==“ return the equality of a reference but not the logical equality meant by the programmer
- Two types of comparing in java
- Comparable = java.lang.Comparable
  - Requires overriding of **compareTo(Object o1)**
  - CompareTo returns (by Convention)
    - -1 if o1>this
    - 0 if o1==this
    - 1 if o1<this

Comparing

- Comparator = java.util.Comparator
  - Requires overriding of **compare(Object o1, Object o2)**
  - Compare returns (by Convention)
    - 1 if o1>o2
    - 0 if o1==o2
    - -1 if o1<o2

How to compare

Comparable should be implemented and the class that should be compared against is specified in <>

```
public class CarCompare implements Comparable<CarCompare>{
    //some code here
    public int compareTo(CarCompare car){
        if(this.getMaxSpeed() > car.getMaxSpeed()){
            return 1;
        }
        else{
            if(this.getMaxSpeed() < car.getMaxSpeed()){
                return -1;
            }
        }
        return 0;
    }
}
```

compareTo() is used for comparing

Returning 1 if this is greater -1 if this is less and 0 if equals

How to compare

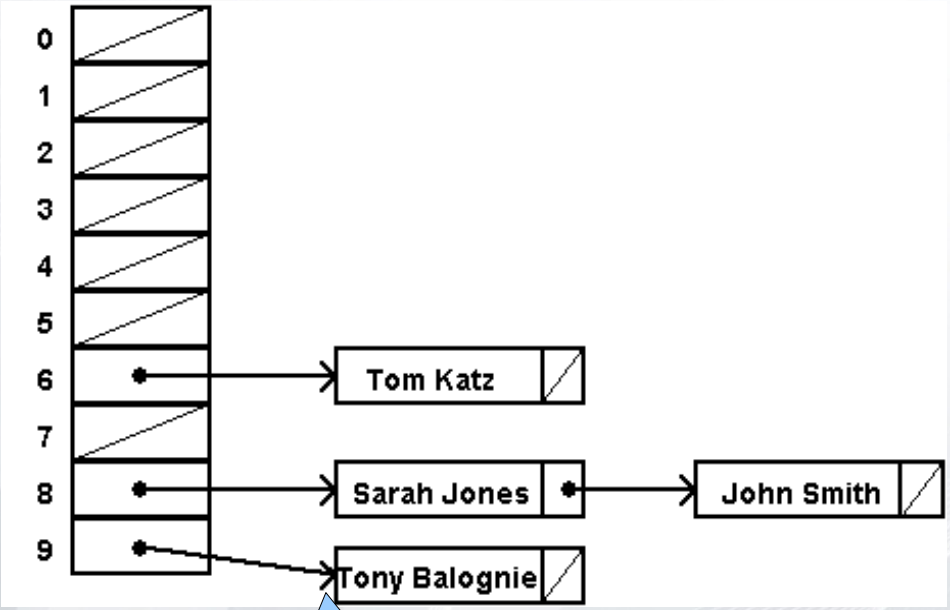
Comparator should be implemented and the class that should be compared against is specified in <>

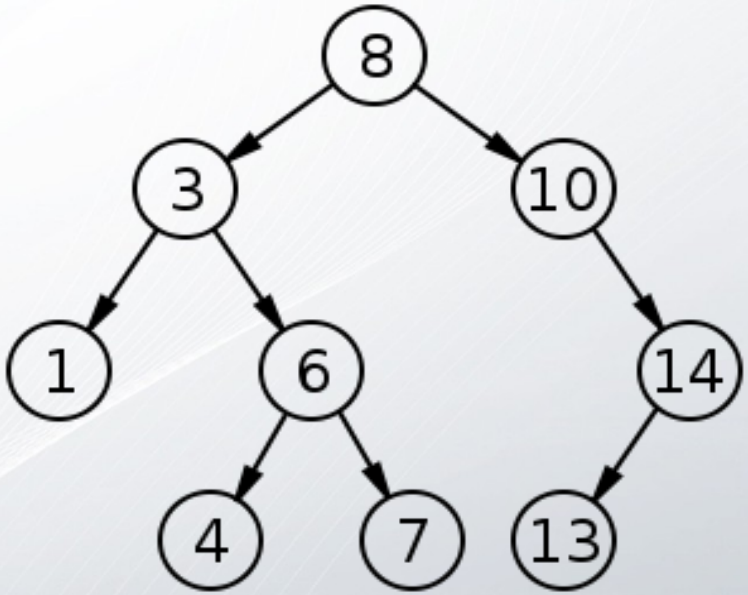
```
public class CarComparator implements Comparator<Car>{
    @Override
    public int compare(Car car1, Car car2) {
        if(car1.getMaxSpeed() > car2.getMaxSpeed()){
            return 1;
        }
        else{
            if(car1.getMaxSpeed() < car2.getMaxSpeed()){
                return -1;
            }
        }
        return 0;
    }
}
```

compare is used for comparing Two objects are passe

Returning 1 if first object is greater -1 if first objects is less and 0 if equals

Trees & Hash Tables





HashSet – stores the values for particular key. Values are retrieved by the key

Tree- hierarchical structure

Set

- Contains only unique values
  - What is uniqueness?
- Implementations – TreeSet, HashSet, LinkedHashSet
  - TreeSet
    - The elements in the set are sorted
    - Not synchronized
    - Objects in TreeSet MUST implement Comparable
    - Uniqueness is granted by Comparable

Set

- HashSet
  - No guarantee the order of the elements will be kept
  - Not synchronized
  - hashCode() is used for keeping track of the uniqueness

How to use Set in Java

- Constructors
  - HashSet hashSet = new HashSet();
  - TreeSet treeSet = new TreeSet();
- Methods
  - set.add(o1);
  - set.add(o1);
  - set.remove(o1);
  - set.size();
  - set.iterator;

Stack

- Stack is a LIFO structure
- All elements are added at the top
- All elements are extracted from the top
- In Java Stack extends Vector
- Methods
  - push() - adds an element
  - pop() - removes an element

Queue

- FIFO structure
  - All new elements are added at the beginning
  - All elements are got from the end
- java.util.Queue
- Methods
  - offer(Object o1) -adds a new element
  - remove() - retrieves and removes the head
  - peek() - retrieves but does not remove the head
  - poll() - same as peek, returns null if the queue is empty

Understanding Set, List, Queue and Stack

- Use List as a Set
- Use Set as a List?
- Use List as a Queue
- Use List as a Stack
- Use Queue and Stack as a List?

Map

- Contains key-value pairs
- Keys are unique
- Most common implementations
  - HashMap – HashSet of keys i.e. order of elements may change.
  - SortedMap – Elements are sorted by key
  - TreeMap – Red-black tree

How to use Map

- Constructor
  - Map map = new HashMap();
  - Map map = new TreeMap();
- Methods
  - map.put(key,value);
  - map.remove(key);
  - map.containsKey(key);
  - map.containsValue(value);
  - Set keysSet = map.keySet();
  - Collection values = map.values();

Collections class

- addAll()
- fill()
- max(), min()
- shuffle()
- sort, swap()
- unmodifiableList()

Custom implementation of LinkedList

- Create a class Node
  - Create a field next of type Node
  - Create a field element for the values
- Create a class LinkedList
  - Implement List
  - Create a field head of type Node
  - Implement all methods from List
  - Create methods get
  - Rearrange pointers when adding/removing a node

Custom implementation of Stack and Queue

- Each node points to the next one – define a Node class as for LinkedList
- Create class Stack/Queue
- Create a field head/top of type Node
- Create methods push() and pop() for stack
- Create methods pop() and poll() for queue
- For queue – change the head when removing
- For stack – change the top when adding

Summary

- Collections are dynamic structures
- Basic data structures – linked list, binary tree, hash table
- Basic collection types – list, set, map, stack, queue
- Iterators
- Comparing