Collection framework – Generics

Object Oriented Programming with Java

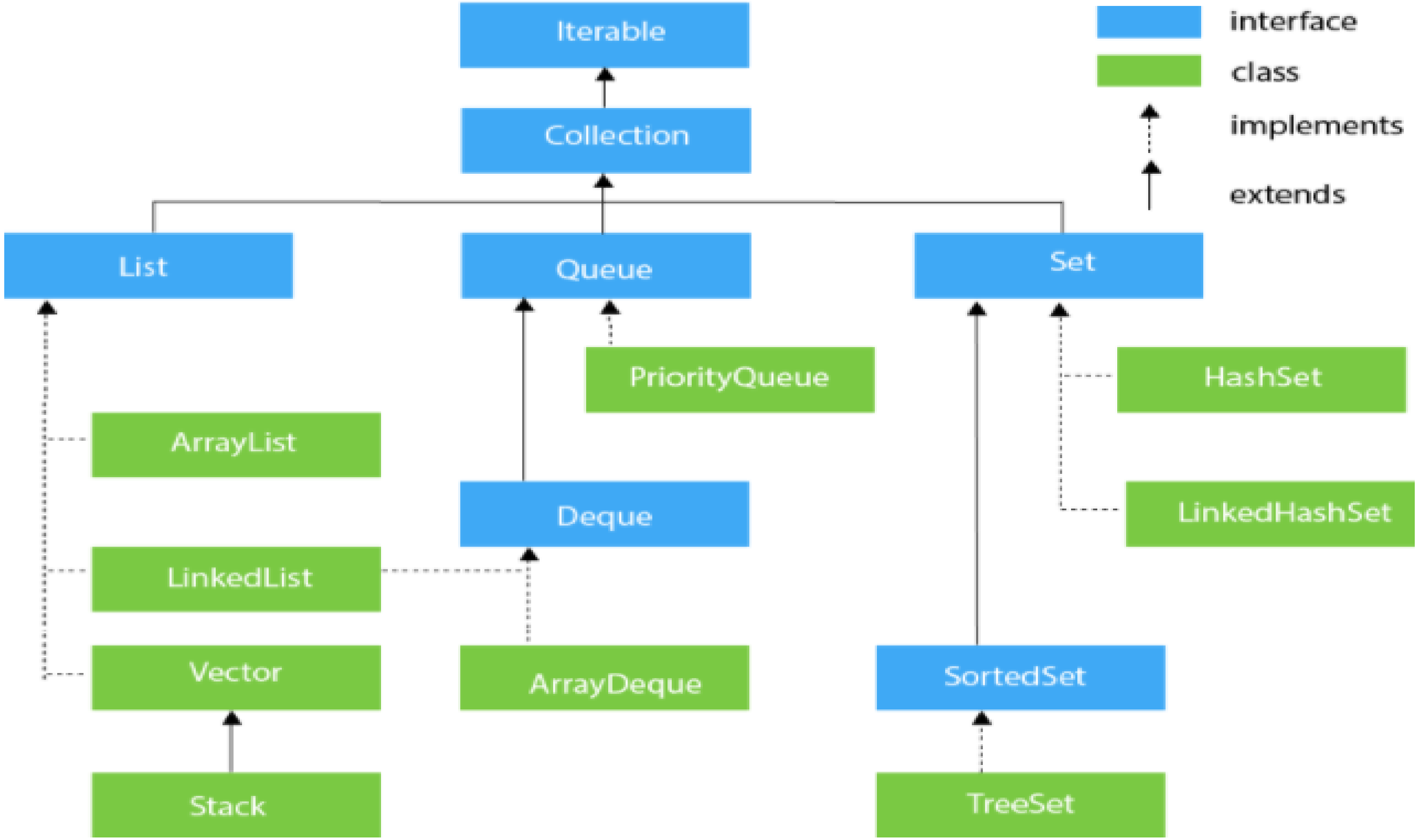
Chapter 6

FPTU Da Nang – IT Department

Objectives

- Introduction of Collections framework
- Collections framework
  - List : ArrayList, LinkedList, Stack, Queue
  - Set : HashSet, TreeSet
  - Map: HashMap, TreeMap
- Generics essential.
  - Generic class and interface
  - Generic methods

Collection framework



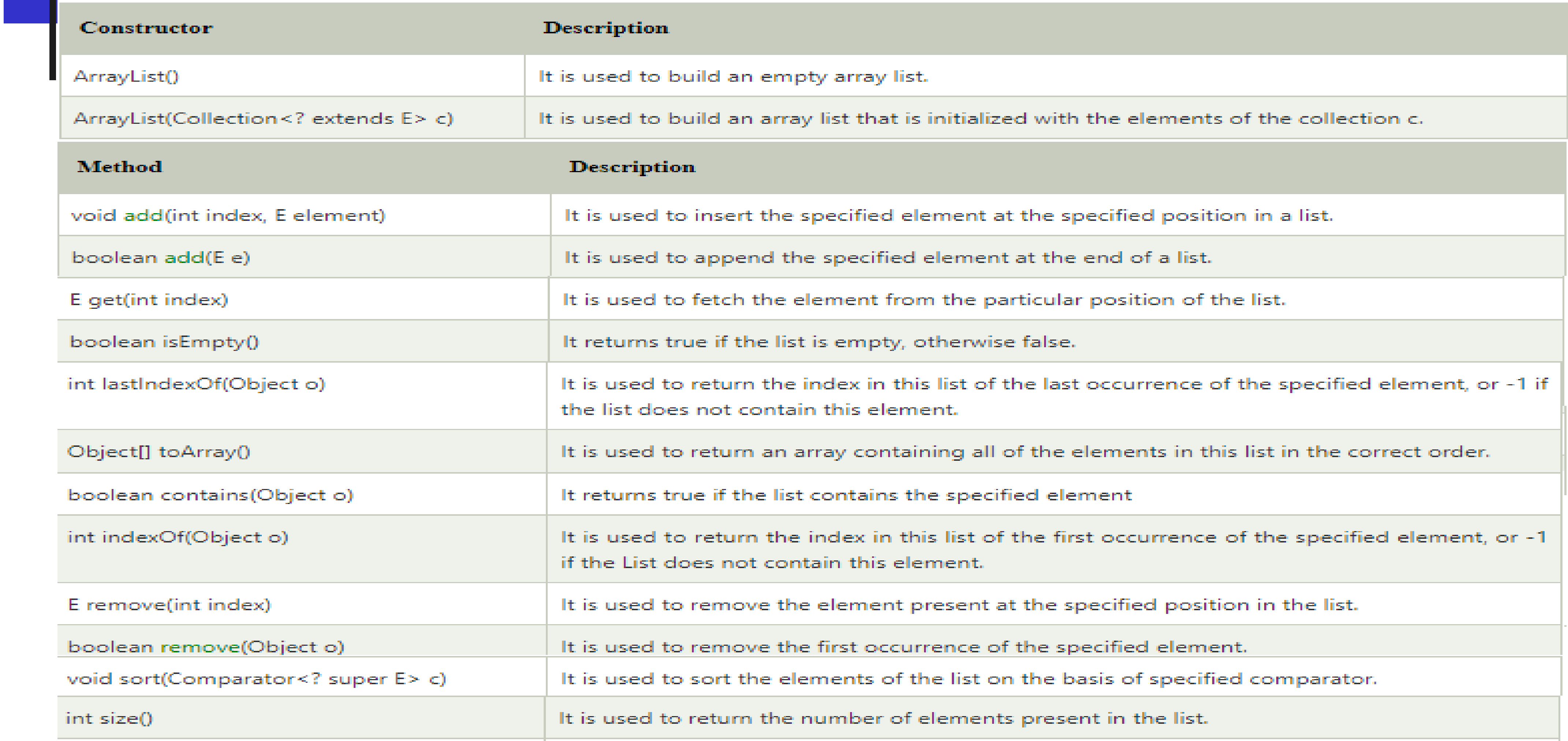
List – dynamic array

- **List** provides the facility to maintain the *ordered collection*.
  - contains the index-based methods
  - can have the duplicate elements
  - can store the null elements in the list. classes of List interface are ArrayList, LinkedList, Stack and Vector
  - implementation classes of List interface are ArrayList, LinkedList, Stack and Vector

ArrayList

- *Dynamic array* for storing the elements.
  - It is like an array, but there is *no size limit*.
  - can contain duplicate elements.
  - maintains insertion order.
  - non synchronized.
  - allows random access because array works at the index basis.
  - manipulation is little bit slower than the LinkedList because a lot of shifting needs to occur if element is removed from list.
  - better for storing and accessing data

ArrayList



Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
boolean contains(Object o)	It returns true if the list contains the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.
int size()	It is used to return the number of elements present in the list.

LinkedList

- LinkedList uses a doubly linked list to store the elements.
  - manipulation is fast because no shifting needs
  - can contain duplicate elements.
  - maintains insertion order.
  - non synchronized.
  - can be used as a list, stack or queue
  - better for manipulating data.

HashSet

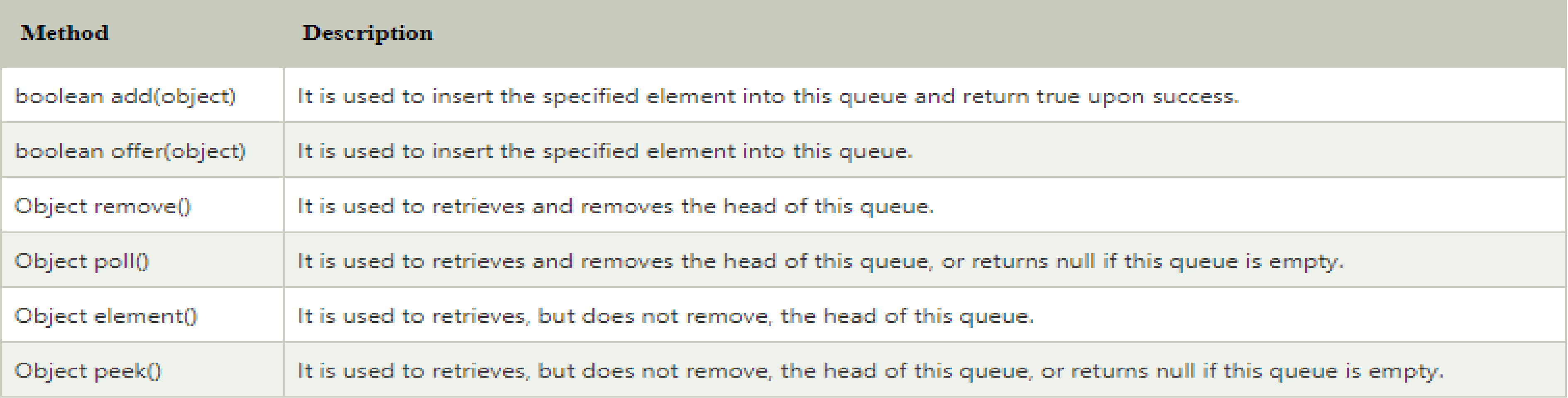
- HashSet is used to create a collection that uses a hash table for storage.
  - stores the elements by using a mechanism called **hashing.**
  - contains unique elements only.
  - allows null value.
  - non synchronized.
  - doesn't maintain the insertion order.
  - inserted on the basis of element hashcode.
  - best approach for search operations.

TreeSet

- TreeSet implements the Set interface that uses a tree for storage.
  - contains unique elements only.
  - access and retrieval times are quiet fast.
  - doesn't allow null element.
  - non synchronized.
  - maintains ascending order

Queue – Deque - Stack

- **Queue** orders the element in FIFO(First In First Out).



Method	Description
boolean add(object)	It is used to insert the specified element into this queue and return true upon success.
boolean offer(object)	It is used to insert the specified element into this queue.
Object remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of this queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

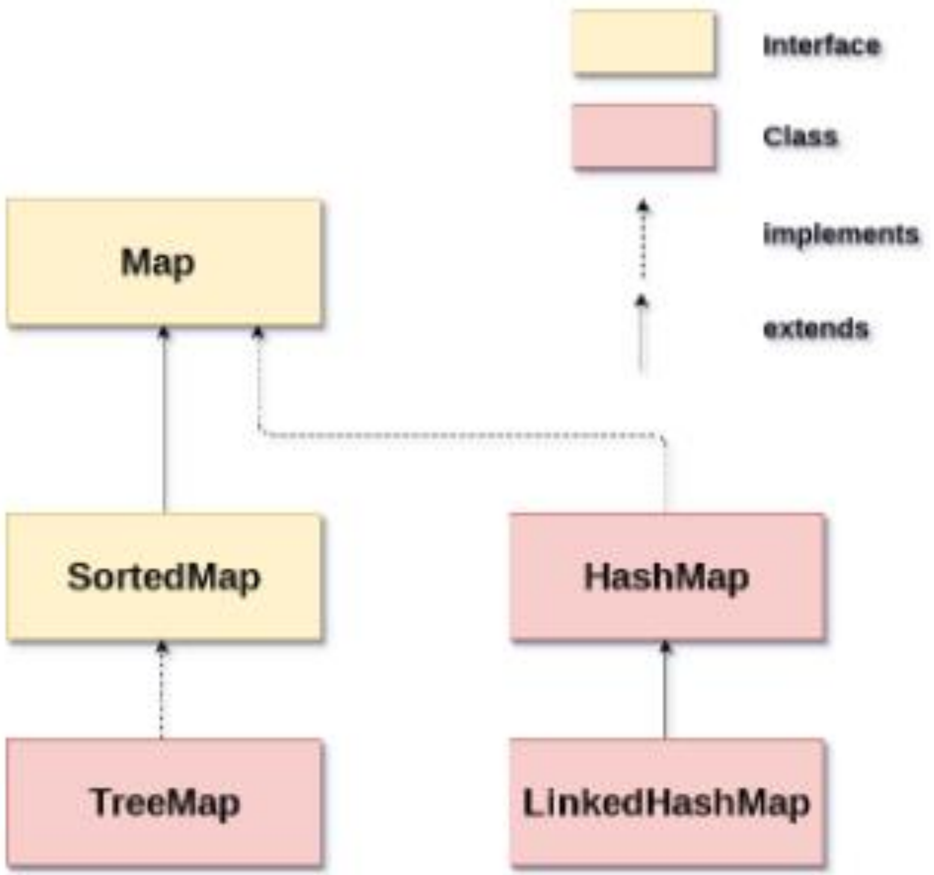
- **Deque** is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for **"double ended queue".**
- **Stack** store the collection of objects in LIFO **Last-In-First-Out**

Map

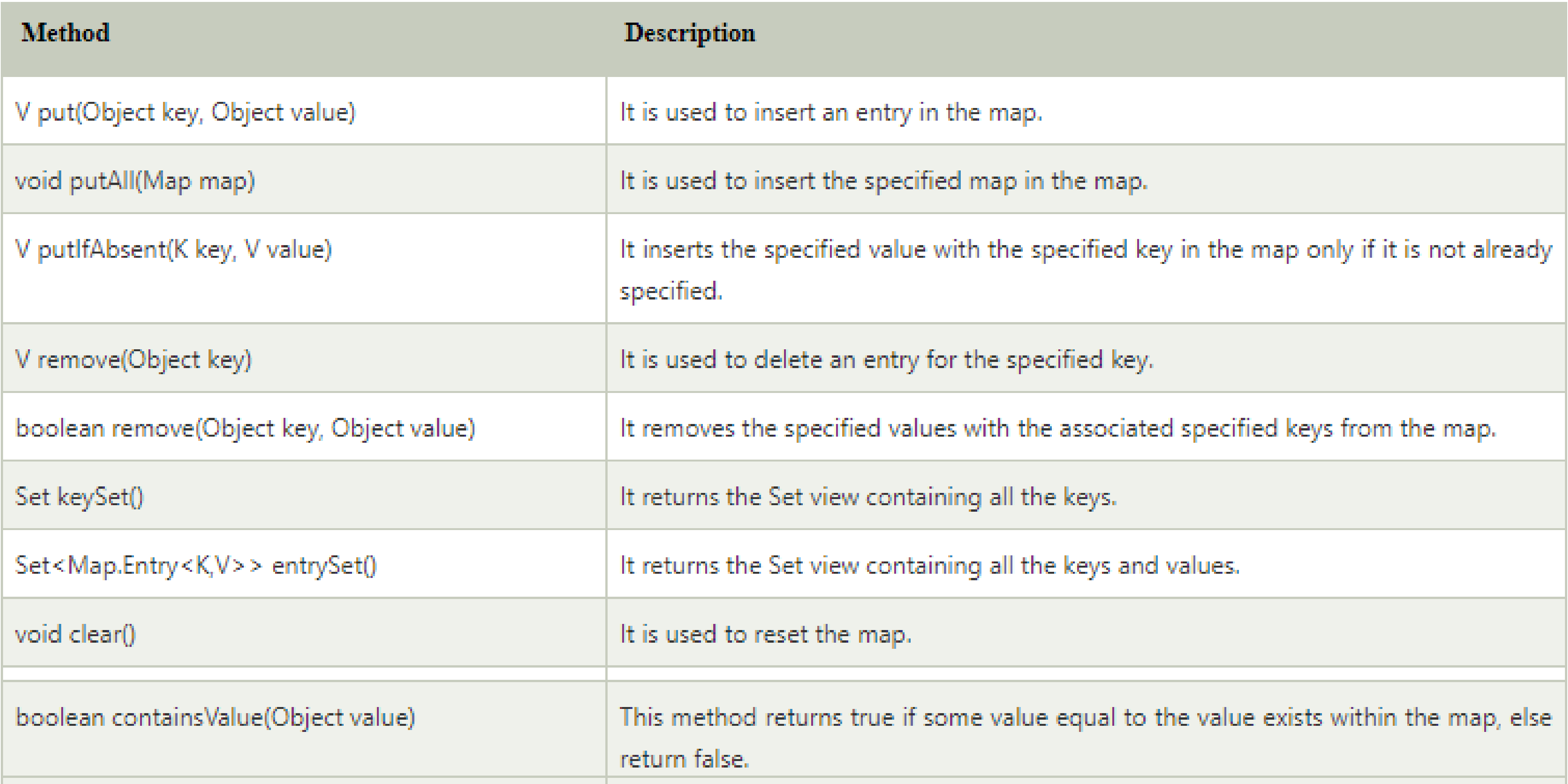
- A map contains values on key-value pair.
  - Each key-value pair is known as an entry.
  - A Map contains unique keys, but can have duplicate values
- A Map is useful for search, update or delete elements on the basis of a key.
- HashMap and LinkedHashMap allow null keys and values, TreeMap doesn't allow any null key or value.
- Map can't be traversed, convert it into Set using *keySet()* or *entrySet()* method.

Map

- HashMap is the implementation of Map, doesn't maintain any order.
- LinkedHashMap inherits HashMap and maintains insertion order.
- TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

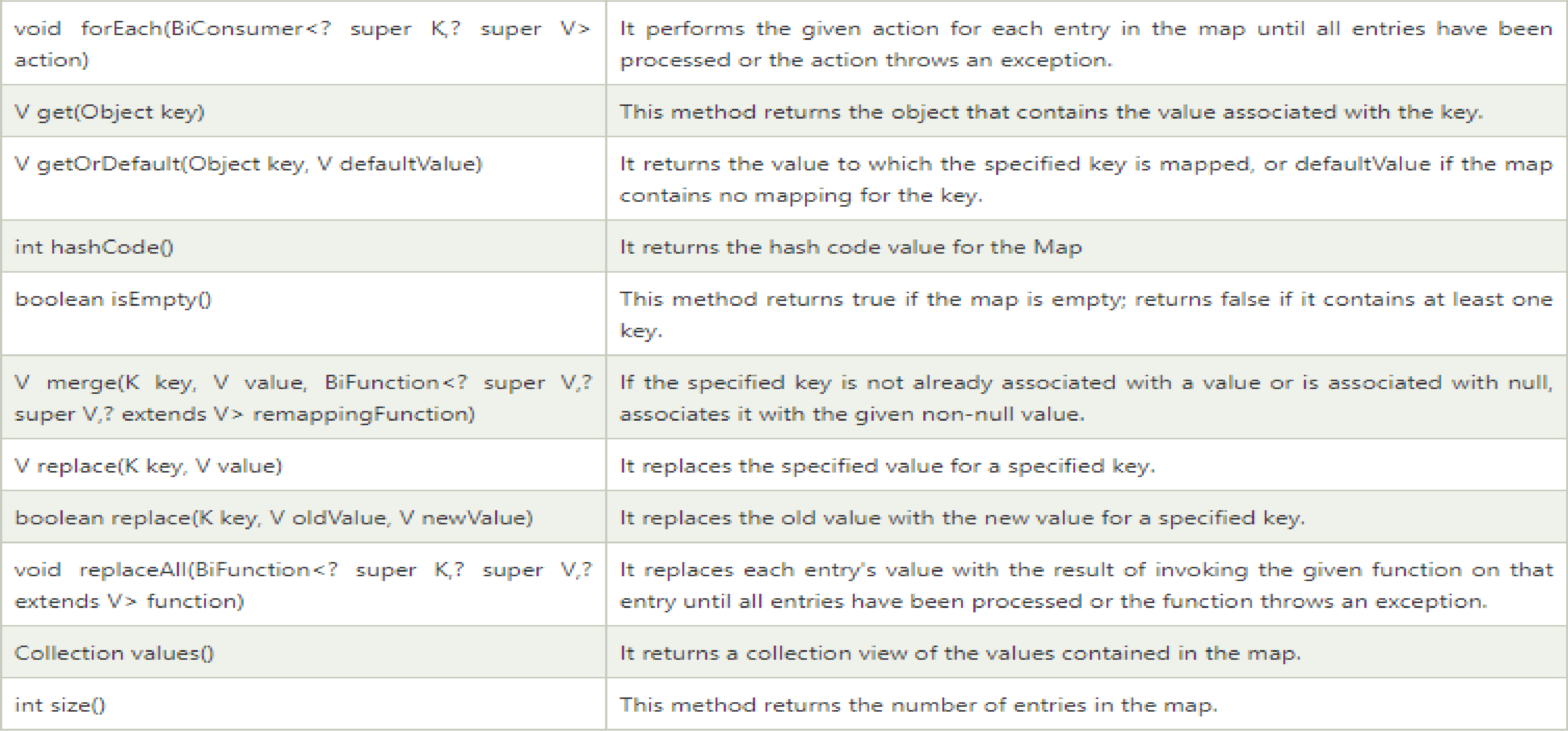


Map



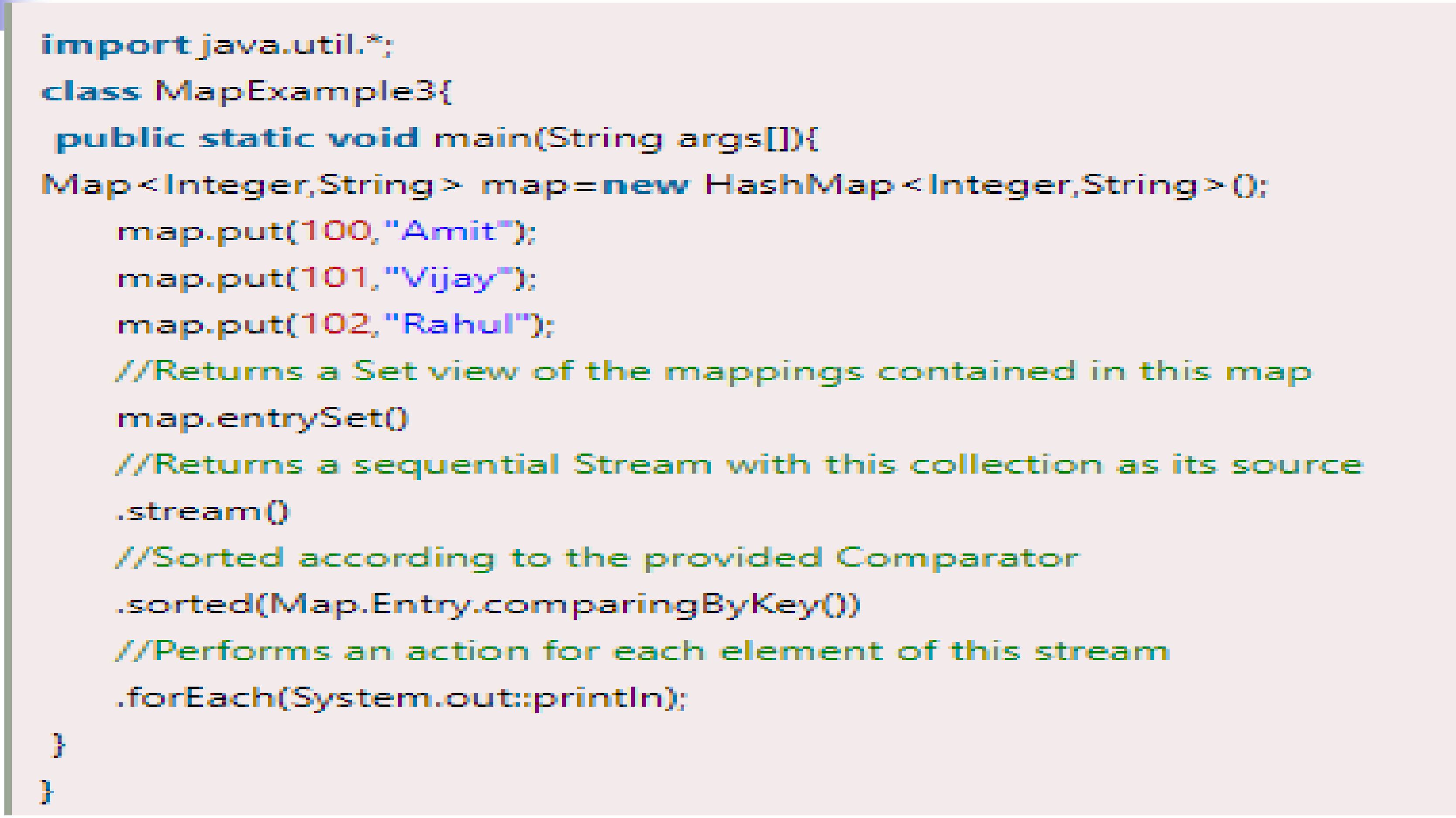
Method	Description
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
Set keySet()	It returns the Set view containing all the keys.
Set<Map.Entry<K,V>> entrySet()	It returns the Set view containing all the keys and values.
void clear()	It is used to reset the map.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.

Map



void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V get(Object key)	This method returns the object that contains the value associated with the key.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
int hashCode()	It returns the hash code value for the Map
boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection values()	It returns a collection view of the values contained in the map.
int size()	This method returns the number of entries in the map.

Map



```
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);
 }
}
```

Generics

- Generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods.
- Generics has many benefits over non-generic code:
  - Stronger type checks at compile time
  - Elimination of casts
  - Enabling programmers to implement generic algorithms
- A *generic class* is defined with the following format:
  - class name<T1, T2, ..., Tn> { /* ... */ }
  - public class Box<T> { *// T stands for "Type"* can be any **non-primitive** type
    private T t;
    public void set(T t) { this.t = t; }
    public T get() { return t; }
    }

Generics

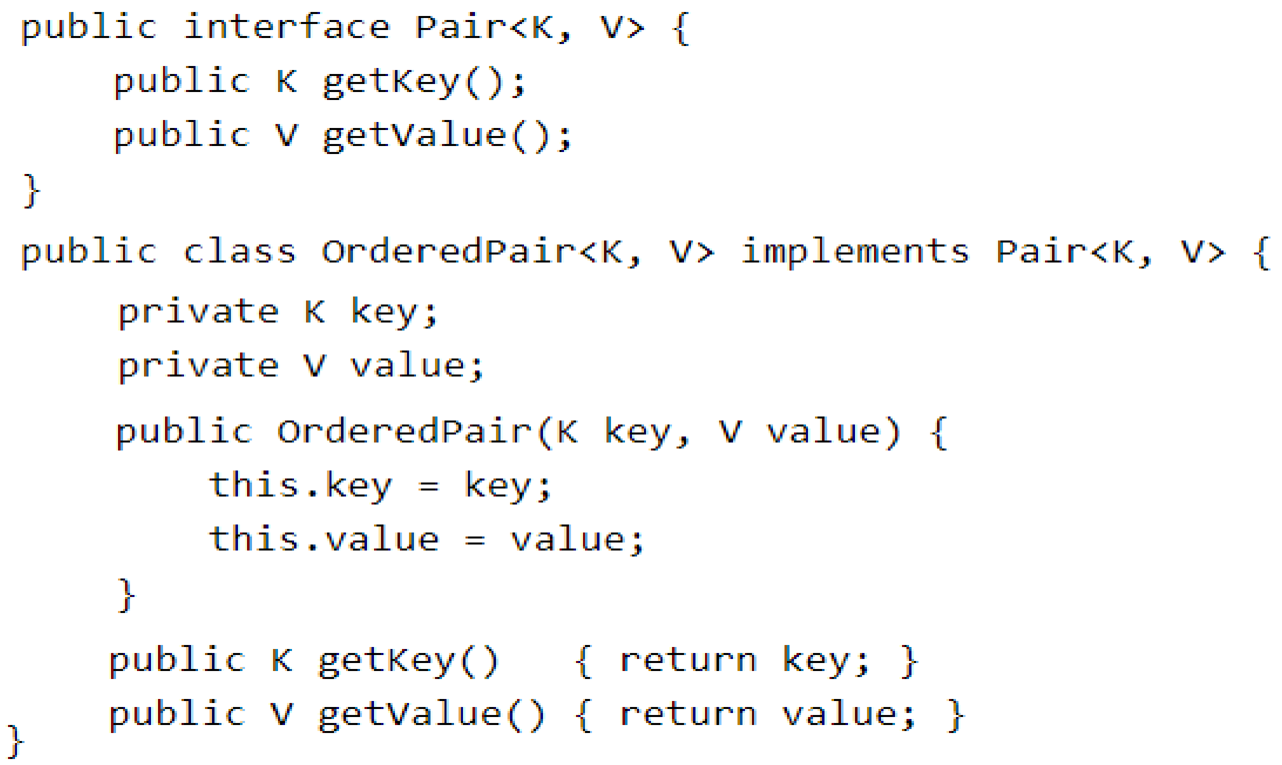
- **Type Parameter Naming Conventions**
  - E - Element (used extensively by the Java Collections Framework)
  - K - Key
  - N - Number
  - T - Type
  - V - Value
  - S,U,V etc. - 2nd, 3rd, 4th types
- **Invoking and Instantiating a Generic Type**
  - generic type invocation replaces T with concrete value: type argument

    ```
    Box<Integer> integerBox = new Box<Integer>();
    ```
  - pair of angle brackets, <>, is informally called *the diamond*

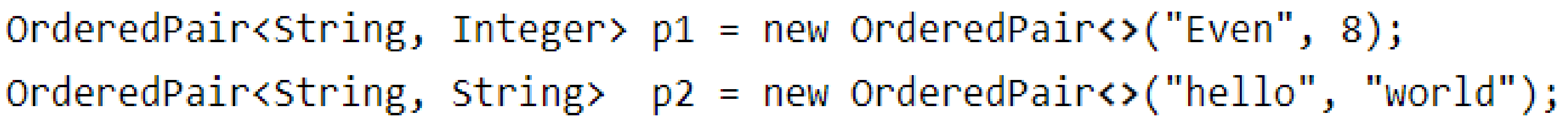
    ```
    Box<Integer> integerBox = new Box<>();
    ```
- Generic class can have multiple type parameters

Generics

- Generic class can have multiple type parameters



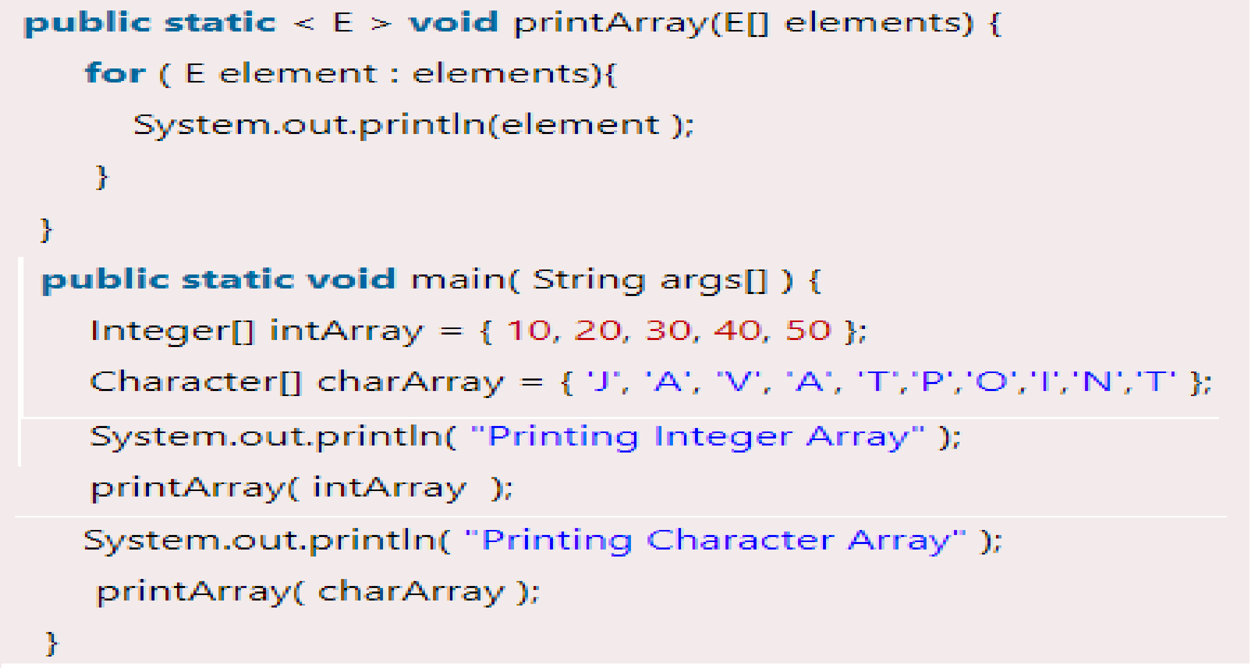
```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}
public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;
    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey()   { return key; }
    public V getValue() { return value; }
}
```



```
OrderedPair<String, Integer> p1 = new OrderedPair<>("Even", 8);
OrderedPair<String, String>  p2 = new OrderedPair<>("hello", "world");
```

Generic methods

- *Generic methods* are methods that introduce their own type parameters
  - The syntax for a generic method includes a list of type parameters, inside angle brackets, which appears before the method's return type.
- *bounded type parameters*
  - to restrict the types that can be used as type arguments
  - type parameter's name,
    followed by the extends keyword,
    followed by its upper bound;
    or followed by the super keyword,
    followed by its lower bound
  - the question mark (?), called the wildcard represents an unknown type
  - **public static void** drawShapes(List<? **extends** Shape> lists)
  - **public static void** addNumbers(List<? **super** Integer> list) {



```
public static < E > void printArray(E[] elements) {
    for ( E element : elements){
       System.out.println(element );
    }
 }
 public static void main( String args[] ) {
    Integer[] intArray = { 10, 20, 30, 40, 50 };
    Character[] charArray = { 'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };
    System.out.println( "Printing Integer Array" );
    printArray( intArray  );
   System.out.println( "Printing Character Array" );
    printArray( charArray );
 }
```

Restrictions on Generics

- Cannot Instantiate Generic Types with Primitive Types
- Cannot Create Instances of Type Parameters
- Cannot Declare Static Fields Whose Types are Type Parameters
- Cannot Use Casts or instanceof With Parameterized Types
- Cannot Create Arrays of Parameterized Types
- Cannot Create, Catch, or Throw Objects of Parameterized Types
- Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type

Homework

- Develop Library Management application:
  - Book searching
  - User registration
  - Book borrow - return
  - Add new book
  - Renew booking
  - View logs
  - List over-due book

Constructive questions

- Compare ArrayList and LinkedList
- For effective searching Book by BookID, how should the the book catalog be organized?
- Analyze the difference of search algorithm on ArrayList and on TreeSet
- Find real life data examples that has similar structure as Stack, Queue
- Analyze the difference of search algorithm on HashMap and on TreeMap

Constructive questions

- Find real life examples that illustrate the benefits of using generics
- Why can't cast or use instanceof with Parameterized Types?
- Write a function that prints the information of an arbitrary list of objects as arguments to the function.
- Write a search function that takes any object as an argument and any search condition.