**Arrays**

Louis Botha, louis.botha@tuni.fi



**Java array** is an object which contains elements of a similar data type. Additionally, an array is a linear data structure representing that can be accessed by index.

# Collections

Louis Botha, louis.botha@tuni.fi

Java Collections Class | Common Methods of Java Collections Class

Java Collections are a group of classes and interfaces that provide a framework for storing and manipulating groups of objects in Java. The framework provides a set of reusable data structures such as lists, sets, maps, and queues to manage collections of objects in Java.

Some of the commonly used Java collections classes and interfaces include:

1. **List Interface**: This interface extends the Collection interface and is used to store an ordered collection of elements. The most commonly used classes implementing this interface are ArrayList and LinkedList.
2. **Set Interface**: This interface extends the Collection interface and is used to store a collection of unique elements. The most commonly used classes implementing this interface are HashSet and TreeSet.
3. **Map Interface**: This interface is used to store a collection of key-value pairs, where each key is associated with a unique value. The most commonly used classes implementing this interface are HashMap and TreeMap.
4. **Queue Interface**: This interface is used to store a collection of elements in which the element entered first is the first to be removed (FIFO). The most commonly used classes implementing this interface are LinkedList and PriorityQueue.
5. **Stack Class**: This class represents a last-in, first-out (LIFO) stack of objects.

Java collections provide a number of useful methods to manipulate, iterate and sort the elements in the collections. The collections framework is also highly extensible, allowing developers to create custom collections that suit their specific needs.

Java collections are an important part of the Java language and are widely used in the development of Java applications. Understanding the collections framework and its various classes and interfaces is crucial for any Java developer.

We wil

## The List interface

The List interface in Java is an ordered collection of elements that can contain duplicate elements. It extends the Collection interface and provides methods to access elements by their index or position in the list.

Some of the important methods provided by the List interface are:

1. add(E element): Adds the specified element to the end of the list.
2. add(int index, E element): Inserts the specified element at the specified position in the list.
3. remove(int index): Removes the element at the specified position in the list.
4. get(int index): Returns the element at the specified position in the list.
5. set(int index, E element): Replaces the element at the specified position in the list with the specified element.
6. size(): Returns the number of elements in the list.
7. indexOf(Object o): Returns the index of the first occurrence of the specified element in the list.

The most commonly used classes implementing the List interface are ArrayList and LinkedList. The ArrayList class is backed by an array and provides fast random access to elements, while the LinkedList class provides efficient insertion and deletion of elements, especially for large lists.



List interface is often used in situations where elements need to be maintained in a specific order and duplicates are allowed. It provides many useful methods for manipulating the elements in the list, such as sorting, searching, and modifying.

In summary, the List interface in Java provides a powerful tool for managing ordered collections of elements, and is an essential component of the Java collections framework.

import java.util.ArrayList;

import java.util.List;

public class ArrayListExample {

   public static void main(String[] args) {

      // Create an ArrayList to store a list of integers

      List<Integer> myList = new ArrayList<Integer>();

      // Add some integers to the list

      myList.add(10);

      myList.add(20);

      myList.add(30);

      myList.add(40);

      // Print the list to the console

      System.out.println("My list of integers: " + myList);

      // Add an integer to the beginning of the list

      myList.add(0, 5);

      // Remove an integer from the list

      myList.remove(2);

      // Print the list to the console again

      System.out.println("My updated list of integers: " + myList);

      // Get the value at a specific index

      int value = myList.get(1);

      System.out.println("The value at index 1 is: " + value);

      // Check if the list contains a specific value

      boolean containsValue = myList.contains(30);

      System.out.println("Does the list contain the value 30? " + containsValue);

      // Get the size of the list

      int size = myList.size();

      System.out.println("The size of the list is: " + size);

*// Print the list*

      System.out.print("The list: ");

      myList.forEach((num) -> System.out.print(num + " "));   }

}

// Output

My list of integers: [10, 20, 30, 40]

My updated list of integers: [5, 10, 30, 40]

The value at index 1 is: 10

Does the list contain the value 30? true

The size of the list is: 4

The list: 5 10 30 40

Here is an example using an object created from a custom class.

import java.util.ArrayList;

import java.util.List;

class Person {

    private String name;

    private int age;

    public Person(String name, int age) {

        this.name = name;

        this.age = age;

    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

}

public class ArrayListObjectExample {

    public static void main(String[] args) {

*// Create an ArrayList to store a list of Person objects*

        List<Person> people = new ArrayList<Person>();

*// Add some Person objects to the list*

        people.add(new Person("Alice", 30));

        people.add(new Person("Bob", 25));

        people.add(new Person("Charlie", 35));

        people.add(new Person("David", 40));

*// Print the list to the console*

        System.out.println("My list of people");

        people.forEach((person)

            -> System.out.println(person.getName() + " - " + person.getAge()));

*// Add a new Person object to the list*

        people.add(new Person("Emily", 20));

*// Remove a Person object from the list*

        people.remove(2);

*// Get the name of a specific Person object in the list*

        String name = people.get(1).getName();

        System.out.println("The name of the person at index 1 is: " + name);

*// Check if the list contains a specific Person object*

        boolean containsPerson = people.contains(new Person("Bob", 25));

        System.out.println("Does the list contain a person with name 'Bob' and age 25? " + containsPerson);

*// Get the size of the list*

        int size = people.size();

        System.out.println("The size of the list is: " + size);

*// Print the list to the console again*

        System.out.println("My updated list of people: ");

        people.forEach((person)

            -> System.out.println(person.getName() + " - " + person.getAge()));

    }

}

// Output

My list of people

Alice - 30

Bob - 25

Charlie - 35

David - 40

The name of the person at index 1 is: Bob

Does the list contain a person with name 'Bob' and age 25? false

The size of the list is: 4

My updated list of people:

Alice - 30

Bob - 25

David - 40

Emily - 20

## The Set Interface

The Set interface represents a collection that contains **no duplicate** elements. In other words, each element in a Set must be unique. Set is a subtype of the Collection interface and inherits all of its methods.

Here are some important characteristics of the Set interface:

* Sets cannot contain duplicate elements. If you attempt to add a duplicate element to a set, the second addition will be ignored and the set will remain unchanged.
* The order of elements in a set is not guaranteed. Implementations of the Set interface may choose to order elements in a specific way, but the API does not specify any particular ordering.
* The Set interface does not provide methods for accessing elements by index, since the order of elements is not guaranteed. However, you can use an iterator to iterate over the elements of a set.

Some of the most commonly used methods of the Set interface include:

* add(E e): Adds the specified element to the set if it is not already present.
* remove(Object o): Removes the specified element from the set if it is present.
* contains(Object o): Returns true if the set contains the specified element, and false otherwise.
* size(): Returns the number of elements in the set.
* isEmpty(): Returns true if the set contains no elements, and false otherwise.



Here's an example of how you might use a Set in Java:

import java.util.HashSet;

import java.util.Set;

public class SetExample {

    public static void main(String[] args) {

        // Create an HashSet to store a list of names(Strings)

        Set<String> names = new HashSet<>();

        // Add to the elements to the set

        names.add("Alice");

        names.add("Bob");

        names.add("Charlie");

        // This addition will be ignored, since "Alice" is already in the set

        names.add("Alice");

        // prints "Size of set: 3"

        System.out.println("Size of set: " + names.size());

        // prints "Is Bob in the set? true"

        System.out.println("Is Bob in the set? " + names.contains("Bob"));

        // Removes the name(String) from the set

        names.remove("Charlie");

        // prints "Updated size of set: 2"

        System.out.println("Updated size of set: " + names.size());

*// Print the set to the console*

        System.out.println("My updated set of people: ");

        for (String name : names) {

            System.out.println(name);

        }

    }

}

//Output

Size of set: 3

Is Bob in the set? true

Updated size of set: 2

My updated set of people:

Bob

Alice

More advanced example using a custom object

import java.util.HashSet;

import java.util.Set;

class Person {

    private String name;

    private int age;

    public Person(String name, int age) {

        this.name = name;

        this.age = age;

    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

*// Implement the equals method for comparing the person object*

    @Override

    public boolean equals(Object o) {

        if (this == o) {

            return true;

        }

        if (o == null || getClass() != o.getClass()) {

            return false;

        }

        Person person = (Person) o;

        return this.name.equals(person.getName());

    }

*// Implement the hashCode method for generating a hash of the object based on the name property*

    @Override

    public int hashCode() {

        return this.name.hashCode();

    }

}

public class SetObjectExample {

    public static void main(String[] args) {

        Set<Person> personSet = new HashSet<>();

*// Add some people to the set*

        Person alice = new Person("Alice", 30);

        personSet.add(alice);

        Person bob = new Person("Bob", 25);

        personSet.add(bob);

        Person charlie = new Person("Charlie", 40);

        personSet.add(charlie);

*// Add a duplicate person*

        Person alice2 = new Person("Alice", 30);

        personSet.add(alice2);

*// The size of the set should be 3, since Alice is a duplicate*

        System.out.println("Size of set: " + personSet.size());

*// Check if Bob is in the set*

        if (personSet.contains(bob)) {

            System.out.println("Bob is in the set");

        }

*// Remove Charlie from the set*

        personSet.remove(charlie);

*// Print out the remaining people in the set*

        System.out.println("People in set:");

        for (Person person : personSet) {

            System.out.println(person.getName() + ", age " + person.getAge());

        }

*// Clear the set*

        personSet.clear();

        System.out.println("Size of set after clearing: " + personSet.size());

    }

}

//Output

Size of set: 3

Bob is in the set

People in set:

Bob, age 25

Alice, age 30

Size of set after clearing: 0

The hashCode() method is used to generate a hash code for an object, which is an integer that is used to identify the object in hash-based collections like HashSet and HashMap.

When you add an object to a HashSet, the HashSet implementation calls the hashCode() method of the object to determine which bucket to put the object in. If two objects have the same hash code, they are considered equal and only one of them will be added to the set.

In the case of the hashCode() method, it is important to override it in order to ensure that objects that are considered equal have the same hash code. If you don't override hashCode(), then the default implementation from the Object class will be used, which returns a different hash code for every object.

By overriding hashCode() to use the name field of the Person class, we ensure that two Person objects with the same name will have the same hash code, and will be considered equal by the HashSet.

## The Map interface

The Map interface in Java is a collection that stores key-value pairs, where each key is associated with a unique value. The Map interface does not extend the Collection interface, but provides methods to access, add, remove and manipulate key-value pairs.

Some of the important methods provided by the Map interface are:

1. put(K key, V value): Associates the specified value with the specified key in the map.
2. get(Object key): Returns the value associated with the specified key in the map.
3. remove(Object key): Removes the key-value pair associated with the specified key from the map.
4. containsKey(Object key): Returns true if the map contains the specified key.
5. containsValue(Object value): Returns true if the map contains the specified value.
6. keySet(): Returns a Set view of the keys in the map.
7. values(): Returns a Collection view of the values in the map.
8. entrySet(): Returns a Set view of the key-value pairs in the map.

The most commonly used classes implementing the Map interface are HashMap and TreeMap. HashMap is based on a hash table and provides constant-time performance for most operations, while TreeMap is based on a tree structure and maintains the elements in sorted order.



HashMaps are commonly used when we need to store and retrieve data in key-value pairs, and we need to do it with constant time complexity. It provides a fast and efficient way to access and manipulate key-value pairs.

Here are some examples of good use cases for HashMaps:

1. Caching - In applications that need to frequently access data from a database or an external API, it's often faster to store the data in a HashMap and access it from memory rather than making repeated requests to the external source. HashMaps can be used as a cache to store the data and quickly retrieve it when needed.
2. Indexing - HashMaps can be used to build indexes for large datasets, making it faster to search and retrieve data based on certain criteria.
3. Counting - HashMaps can be used to count the occurrences of elements in a collection. For example, you could use a HashMap to count the frequency of words in a document.
4. Configuration - HashMaps can be used to store configuration settings for an application, where the keys represent the configuration names and the values represent the corresponding values.
5. Memoization - In dynamic programming, HashMaps can be used to store intermediate results of computations to avoid redundant calculations, leading to faster execution times.

In summary, the Map interface in Java provides a powerful tool for managing collections of key-value pairs, and is an essential component of the Java collections framework.

import java.util.HashMap;

public class HashMapExample {

    public static void main(String[] args) {

        // create a new HashMap instance with String keys and Integer values

        HashMap<String, Integer> map = new HashMap<>();

        // add some key-value pairs to the map

        map.put("Alice", 25);

        map.put("Bob", 32);

        map.put("Charlie", 28);

        // retrieve the value for a given key

        int aliceAge = map.get("Alice");

        System.out.println("Alice's age is " + aliceAge);

        // check if a key is present in the map

        boolean hasBob = map.containsKey("Bob");

        System.out.println("The map " + (hasBob ? "has" : "does not have") + " a value for Bob");

        // iterate over the key-value pairs in the map

        for (String key : map.keySet()) {

            int value = map.get(key);

            System.out.println(key + " is " + value + " years old");

        }

    }

}

In this example, we create a new HashMap instance with String keys and Integer values. We add three key-value pairs to the map, with keys "Alice", "Bob", and "Charlie", and values 25, 32, and 28, respectively.

We then retrieve the value for the key "Alice" using the get() method and store it in a variable called aliceAge. We print out a message to the console showing Alice's age.

Next, we use the containsKey() method to check if the key "Bob" is present in the map. We print out a message to the console indicating whether or not the map has a value for Bob.

Finally, we use a for loop to iterate over the key-value pairs in the map. For each key in the map, we retrieve the corresponding value using the get() method and print out a message to the console showing the key and value. This allows us to print out a list of all the people in the map and their ages.

Overall, this example demonstrates some of the basic functionality of the HashMap class in Java, including adding and retrieving key-value pairs, checking for the presence of keys, and iterating over the key-value pairs in the map.

import java.util.HashMap;

import java.util.Map;

public class AdvancedHashMapExample {

    public static void main(String[] args) {

        // create a new HashMap instance with String keys and Double values

        HashMap<String, Double> salaries = new HashMap<>();

        // add some key-value pairs to the map

        salaries.put("Alice", 50000.0);

        salaries.put("Bob", 75000.0);

        salaries.put("Charlie", 60000.0);

        // print out the salaries of all employees

        System.out.println("Initial salaries:");

        for (String name : salaries.keySet()) {

            double salary = salaries.get(name);

            System.out.println(name + ": " + salary);

        }

        // increase the salaries of all employees by 10%

        for (String name : salaries.keySet()) {

            double salary = salaries.get(name);

            salaries.put(name, salary \* 1.1);

        }

        // print out the new salaries of all employees

        System.out.println("New salaries:");

        for (Map.Entry<String, Double> entry : salaries.entrySet()) {

            String name = entry.getKey();

            double salary = entry.getValue();

            System.out.println(name + ": " + salary);

        }

        // remove the lowest-paid employee from the map

        String lowestPaid = null;

        double lowestSalary = Double.MAX\_VALUE;

        for (Map.Entry<String, Double> entry : salaries.entrySet()) {

            String name = entry.getKey();

            double salary = entry.getValue();

            if (salary < lowestSalary) {

                lowestPaid = name;

                lowestSalary = salary;

            }

        }

        salaries.remove(lowestPaid);

        // print out the updated salaries of all employees

        System.out.println("Updated salaries:");

        for (Map.Entry<String, Double> entry : salaries.entrySet()) {

            String name = entry.getKey();

            double salary = entry.getValue();

            System.out.println(name + ": " + salary);

        }

    }

}

In this example, we create a HashMap with String keys and Double values, representing the salaries of different employees. We add three key-value pairs to the map, representing the salaries of Alice, Bob, and Charlie.

We then use a for loop to iterate over the keys in the map and print out the initial salaries of all employees.

Next, we use another for loop to increase the salaries of all employees by 10%. We retrieve the value for each key using the get() method, multiply it by 1.1, and then store the new value using the put() method.

We then use a for loop with Map.Entry objects to iterate over the key-value pairs in the map and print out the new salaries of all employees.

Finally, we use another for loop to find the lowest-paid employee in the map and remove them from the map using the remove() method. We then use another for loop to print out the updated salaries of all employees.

This example demonstrates some additional functionality of the HashMap class, including iterating over key-value pairs using Map.Entry objects, performing calculations on the values stored in the map, and removing key-value pairs from the map.

**Collections Workshop**

Louis Botha, louis.botha@tuni.fi

light bulbDo each section in it’s own folder.

**ArrayList**

Create a .java file where you can complete the steps of the workshop.

class Person {

    private String name;

    private int age;

    public Person(String name, int age) {

        this.name = name;

        this.age = age;

    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

}

class Workshop1 {

    public static void main(String[] args) {

    }

}

laptop computer**01**

Create an array of fruit with 3 elements like, Mango, Apple and Pear called fruit. Print the content of the array with System.out.println().

laptop computer**02**

Instantiate an ArrayList of String fruit names with fruitList.

Add the import for ArrayLists, import java.util.ArrayList;.

Using the add() method of an ArrayList, add the three fruit and print the fruitList to the console.

laptop computer**03**

Try to add another fruit to the fruits array. Did it work?

* No, because the size of the array is still 3

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3

Remove the above and add another fruit to the fruitList. Did it work?

* Yes, ArrayLists are dynamic in size and expand and shrink.

light bulbDid you notice that the size of the ArrayList is not specified?

laptop computer**04**

Now remove a fruit from the fruitList using the remove() method. Print the fruitList to the console.

laptop computer**05**

Now check if a fruit is already in the fruitList using the contains() method. Print the result of the check to the console.

laptop computer**06**

Now check the size of the fruitList using the size() method. Print the result of the check to the console.

laptop computer**07**

Now find the position of the “Apple” element in the fruitList using the indexOf() method. Print the result of the check to the console.

laptop computer**08**

Now use forEach to iterate over the fruitList and print each element to the console.

laptop computer**09**

The ArrayList iterator() method returns an iterator to access each element of the ArrayList in a proper sequence.

Add the import for the Iterarator object import java.util.Iterator;

Now iterate over the fruitList and print each element to the console using the ArrayList Iterarator.

laptop computer**10**

Change the ArrayList instantiation to:

List<String> fruitList = new ArrayList<String>();

Add the import for List, import java.util.List;

Compile and run the application.

Did it work? Yes. ArrayList is a List and implements the List interface.

This is a good practice. When we create the ArrayList like this, the list can be potentially typecast into any other type of Java list. This is called **decoupling**. You are **decoupling** your code from a specific implementation of the interface. The advantages it provides, is when writing large amounts of code, you can switch between types of lists to suit your preferences (speed, memory etc), as all of your code, can treat your list as just type List. You can also pass a List as parameters and returns List from functions. The practice is called [programming to an interface.](https://stackoverflow.com/questions/383947/what-does-it-mean-to-program-to-an-interface)

laptop computer**11**

Now empty the fruitList using the clear() method. Print the fruitList to the console.

laptop computer**12**

Initialise an ArrayList called peopleList that has Person objects as elements.

Add 3 Person objects to the peopleList.

Haley, 22

Alex, 20

Luke, 18

Print the content of the peopleList with System.out.println().

What is the output that you see?

laptop computer**13**

Override and implement the toString() method in the Person class so that the peopleList can be printed with System.out.println().

laptop computer**14**

Create a Person object with the same name and age that you used in step 12.

Now check if the same Person is already in the peopleList using the contains() method. Print the result of the check to the console.

Is the result correct?

Override and implement the equals() method in the Person class that will check if an Object passed as a parameter is the same as this object.

Do the check again. Did the result change?

laptop computer**15**

Check out some of the other [ArrayList methods](https://www.javatpoint.com/java-arraylist) and find a method that was not tested above. Create a code snippet that will demonstrate the use of the method. Think of it as a tutorial. Remember to add comments.

Now take a screenshot of your tutorial code and paste it on this Mural board

* [EA](https://app.mural.co/t/tg5g00ev163001websovelluskeh4798/m/tg5g00ev163001websovelluskeh4798/1676907732401/7376ab16c0ae9b9fbafad737eb5e432217a648ea?sender=u9bcbbed311777a14d9149309)
* [EB](https://app.mural.co/t/tg5g00ev163001websovelluskeh4798/m/tg5g00ev163001websovelluskeh4798/1676907961634/e28324a613840bf42fd827c1de27b5352a5d26df?sender=u9bcbbed311777a14d9149309)

**HashMaps**

Create a .java file where you can complete the steps of the workshop.

class Person {

    private String name;

    private int age;

    public Person(String name, int age) {

        this.name = name;

        this.age = age;

    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

}

class Workshop2 {

    public static void main(String[] args) {

    }

}

laptop computer**01**

Create three integers in the file and call the integers a, b, and c and the integers have the values 10, 3, and 88 respectively.

int a = 10;

int b = 3;

int c = 88;

This is still manageable but what if there is a hundred of these variables that needs to be used and accessible.

Add the import for a HashMap import java.util.HashMap;

Create a HashMap called varMap that will store variables like above. Use the name of the variable as key and the value as value.

Add the 3 variables above to the varMap.

Print the content of the map with System.out.println().

laptop computer**02**

Get and print the value of a and c to the console.

laptop computer**03**

Create a new HashMap called usersMap that will store usernames and passwords. Add the following 3 users to the usersMap.

"johnwick12" : "h!tm4n"

"TonyStark1122" : "!r0nm4n"

"SelinaKyle9999" : "c4tw0m4n"

Print the content of the map with System.out.println().

laptop computer**04**

Now remove user johnwick12 from the usersMap with the remove() method. Print the content of the map to verify the change.

laptop computer**05**

Now check if the usersMap contains a value called h!tm4n using the containsValue() method.

Print the result of the check to the console.

Try some of the other values from the map and some not in the map.

laptop computer**06**

Now check if the usersMap contains a value called SelinaKyle9999 using the containsValue() method.

Print the result of the check to the console.

What was the result? False, it is checking the values and not the keys.

Now change the containsValue() method to the containsKey() method and run the application.

Try some of the other keys from the map and some not in the map.

laptop computer**07**

Now check the size of the usersMap using the size() method. Print the result of the check to the console.

laptop computer**08**

User TonyStark1122 has updated his password to b3tt3rp@ssw0rd. Make the change to usersMap using the replace() method. Print the usersMap to console and make sure it changed.

Notice that the old value was returned by the method.

laptop computer**09**

Use the keySet() method to print all the keys in usersMap to the console.

Use the values() method to print all the values in usersMap to the console.

laptop computer**10**

Now use forEach to iterate over the usersMap and print the element to the console.

laptop computer**11**

Change the HashMap instantiation to:

Map<String,String> usersMap = new HashMap<String,String>();

Add the import for List, import java.util.Map;

Compile and run the application.

Did it work? Why?

laptop computer**12**

Create a new HashMap called positionMap

Map<Integer,Person> positionMap = new HashMap<Integer,Person>();

Create 3 Person objects and it to the positionMap.

Person haley = new Person("Haley", 22));

Person alex = new Person("Alex", 20));

Person luke = new Person("Luke", 18));

Add the Person objects to the positionMap.

Print the content of the map with System.out.println().

What is the output that you see?

laptop computer**13**

Override and implement the toString() method in the Person class so that the positionMap can be printed with System.out.println().

laptop computer**14**

Check if the same Person is already in the positionMap using the contains() method. Print the result of the check to the console.

Is the result correct?

laptop computer**15**

Check out some of the other [HashMap methods](https://www.javatpoint.com/java-hashmap) and find a method that was not tested above. Create a code snippet that will demonstrate the use of the method. Think of it as a tutorial. Remember to add comments.

Now take a screenshot of your tutorial code and paste it on this Mural board

* [EA](https://app.mural.co/t/tg5g00ev163001websovelluskeh4798/m/tg5g00ev163001websovelluskeh4798/1676907732401/7376ab16c0ae9b9fbafad737eb5e432217a648ea?sender=u9bcbbed311777a14d9149309)
* [EB](https://app.mural.co/t/tg5g00ev163001websovelluskeh4798/m/tg5g00ev163001websovelluskeh4798/1676907961634/e28324a613840bf42fd827c1de27b5352a5d26df?sender=u9bcbbed311777a14d9149309)

**HashSet**

Create a .java file where you can complete the steps of the workshop.

class Person {

    private String name;

    private int age;

    public Person(String name, int age) {

        this.name = name;

        this.age = age;

    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

}

class Workshop3 {

    public static void main(String[] args) {

    }

}

laptop computer**01**

Add the import for a HashSet at the top of the file,  import java.util.HashSet;

Create a HashSet called animalSet add 3 values like Lion, Tiger and Cheetah to it. Print the content of the set with System.out.println().

laptop computer**02**

Try to add another Lion to the animalSet and print the result of the add  method with System.out.println().

Did it succeed in adding? What did the method return?

Try to add Rhino to the animalSet and print the result of the add  method with System.out.println().

Did it succeed in adding? What did the method return? Why?

laptop computer**03**

Now remove user Lion from the animalSet with the remove() method. Print the content of the map to verify the change.

laptop computer**04**

Now check the size of the animalSet using the size() method. Print the result of the check to the console.

laptop computer**05**

Now check if animalSet already have a Rhino element using the contains() method. Print the result of the check to the console.

laptop computer**06**

Now use forEach to iterate over the animalSet and print each element to the console.

laptop computer**07**

Use the iterator() to iterate and print all the elements in the animalSet.

Add the import for the Iterator object import java.util.Iterator;

laptop computer**08**

Now empty the animalSet using the clear() method. Print the animalSetto the console.

laptop computer**09**

Now check the if the animalSet is empty with the isEmpty() method. Print the result of the check to the console.

laptop computer**10**

Change the HashSet instantiation to:

Set<String> animalSet = new HashSet<String>();

Add the import for Set, import java.util.Set;

Compile and run the application. Did it work?

laptop computer**11**

Create a new HashSet called peopleSet which has Person objects.

Set<Person> peopleSet = new HashSet<Person>();

Create 3 Person objects and it to the peopleSet.

Person haley = new Person("Haley", 22));

Person alex = new Person("Alex", 20));

Person luke = new Person("Luke", 18));

Add the 3 Person objects to the peopleSet.

Print the content of the map with System.out.println().

What is the output that you see?

laptop computer**12**

Override and implement the toString() method in the Person class so that the peopleSet can be printed with System.out.println().

laptop computer**13**

Check if the same Person is already in the peopleSet using the contains() method. Print the result of the check to the console.

Is the result correct?

Override and implement the boolean equals(Object o) method in the Person class.

Do the check again, is the result now correct?

Override and implement the public int hashCode() method in the Person class. Use the code below.

Do the check again, is the result now correct?

laptop computer**14**

Check out some of the other [HashSet methods](https://www.javatpoint.com/java-hashset) and find a method that was not tested above. Create a code snippet that will demonstrate the use of the method. Think of it as a tutorial. Remember to add comments.

Now take a screenshot of your tutorial code and paste it on this Mural board

* [EA](https://app.mural.co/t/tg5g00ev163001websovelluskeh4798/m/tg5g00ev163001websovelluskeh4798/1676907732401/7376ab16c0ae9b9fbafad737eb5e432217a648ea?sender=u9bcbbed311777a14d9149309)
* [EB](https://app.mural.co/t/tg5g00ev163001websovelluskeh4798/m/tg5g00ev163001websovelluskeh4798/1676907961634/e28324a613840bf42fd827c1de27b5352a5d26df?sender=u9bcbbed311777a14d9149309)