**Map and Generics-Assignment**

Question 1: -What is a Map in Java?

Ans: -A [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html) is an object that maps keys to values.Each key and value pair is known as an entry. A map cannot contain duplicate keys: Each key can map to at most one value.

Question 2: -What are the commonly used- implementations of Map in Java?

Ans: - There are three major implementation in java -

1.A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null values like the HashMap and LinkedHashMap, but some do not like the TreeMap.

2.The order of a map depends on the specific implementations. For example, TreeMap and LinkedHashMap have predictable orders, while HashMap does not.

3.There are two interfaces for implementing Map in java. They are Map and SortedMap, and three classes: HashMap, TreeMap, and LinkedHashMap.

Question 3: -What is the difference between HashMap and TreeMap?

Ans: -

| **Basis** | **HashMap** | **TreeMap** |
| --- | --- | --- |
| **Definition** | Java **HashMap** is a hashtable based implementation of Map interface. | Java **TreeMap** is a Tree structure-based implementation of Map interface. |
| **Interface Implements** | HashMap implements **Map, Cloneable**, and **Serializable** interface. | TreeMap implements **NavigableMap, Cloneable**, and **Serializable** interface. |
| **Null Keys/ Values** | HashMap allows a **single** null key and **multiple** null values. | TreeMap does not allow **null** keys but can have **multiple** null values. |
| **Homogeneous/ Heterogeneous** | HashMap allows heterogeneous elements because it does not perform sorting on keys. | TreeMap allows homogeneous values as a key because of sorting. |
| **Performance** | HashMap is **faster** than TreeMap because it provides constant-time performance that is O(1) for the basic operations like get() and put(). | TreeMap is **slow** in comparison to HashMap because it provides the performance of O(log(n)) for most operations like add(), remove() and contains(). |
| **Data Structure** | The HashMap class uses the **hash table**. | TreeMap internally uses a **Red-Black** tree, which is a self-balancing Binary Search Tree. |
| **Comparison Method** | It uses the equals**()** method of the **Object** class to compare keys. The equals() method of Map class overrides it. | It uses the **compareTo()** method to compare keys. |
| **Functionality** | HashMap class contains only basic functions like **get(), put(), KeySet()**, etc. . | TreeMap class is rich in functionality, because it contains functions like: **tailMap(), firstKey(), lastKey(), pollFirstEntry(), pollLastEntry()**. |
| **Order of elements** | HashMap does not maintain any order. | The elements are sorted in **natural order** (ascending). |

Question 4: -How do you check if a key exists in a Map in Java?

Ans: - In java we can check Key exists in map using containsKey() method.

Example: -

package hello;

import java.util.HashMap;

import java.util.Map;

public class Main

{

public static void main(String[] args)

{

Map<Character, Integer> hashMap = new HashMap<>();

hashMap.put('A', 1);

hashMap.put('B', 2);

hashMap.put('C', 3);

char key = 'C';

boolean isKeyExists = hashMap.containsKey(key);

System.out.println(isKeyExists);

}

}

**Output: -true**

Question 5: -What are Generics in Java?

Ans: -Java generics are mainly used to impose type safety in programs. Type safety is when the compiler validates the datatype of constants, variables, and methods whether it is rightly assigned or not.

Question 6: -What are the benefits of using Generics in Java?

Ans: -There are mainly 3 advantages of generics.

**1) Type-safety:** We can hold only a single type of objects in generics. It does not allow to store other objects.

**2) Type casting is not required:** There is no need to typecast the object.

**3) Compile-Time Checking:** It is checked at compile time so problems will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

Question 7: -What is the Generics class in Java?

Ans: -A class that can refer to any type is known as a generic class. Here, we are using the T type parameter to create the generic class of specific type.

Example: -

**package** hello;

**class** customGen<T> {

T obj;

**void** add(T obj) {

**this**.obj = obj;

}

T get() {

**return** obj;

}

}

**public** **class** Test\_Generics {

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

customGen<Integer> m = **new** customGen<Integer>();

m.add(22);

// m.add("vikash chauhan");//Compile time error

System.***out***.println(m.get());

}

}

Question 8: -What is the type parameter in Java Generics ?

Ans: -The type parameters naming conventions are important to learn about generics thoroughly. The common type parameters are as follows:

1. T - Type
2. E - Element
3. K - Key
4. N - Number
5. V - Value

Question 9: -What is the Generics method in Java?

Ans: -we can create a generic method that can accept any type of arguments. Here, the scope of arguments is limited to the method where it is declared. It allows static as well as non-static methods.

Example:-

**package** hello;

**public** **class** generics\_method {

**public** **static** <E> **void** print(E[] elements) {

**for** (E element : elements) {

System.***out***.println(element);

}

System.***out***.println();

}

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

Integer[] array = { 10, 20, 30, 40 };

Character[] ch = { 'H', 'E', 'L', 'L', 'O' };

System.***out***.println("printing integer");

*print*(array);

System.***out***.println("printing Characters");

*print*(ch);

}

}

Question 10: -What is the difference between ArrayList and ArrayList<T>?

Ans: -

| **ArrayList** | **ArrayList<T>** |
| --- | --- |
| This is **non-generics** version of ArrayList | This is **generics version** of ArrayList with type-parameter T |
| In this non-generics version, ArrayList **allows you to add any type of Objects** like String, Integer, references-types, etc. | But Generics version of ArrayList **allows to add specific type of objects only**    Like, if type-parameter T is replaced by String then only String-type of Objects are allowed to add to ArrayList |
| Basically, it **doesn’t** assures **type-safety** as any type of Objects can be added to ArrayList | It assures **type-safety,** as it allows to store same type of Objects only |
| During iteration of ArrayList, compulsorily **explicit type-casting** needs to be done even if ArrayList stores same type of Objects | In Generics version of ArrayList, **no explicit type-casting** is required    **Reason:** Generics ArrayList stores same type of Objects only, therefore **type-casting isn’t required** at the time of iteration or getting Objects |