Map in Java

1️⃣ What is a `Map` in Java?

A **`Map`** in Java is a part of the **Java Collection Framework** that stores **key-value pairs**.
Unlike `List` and `Set`, a `Map` does not allow duplicate keys, but values can be duplicate.

Syntax:

java
Copy code
```
Map<KeyType, ValueType> mapName = new HashMap<>();
```

Example:

java
Copy code
```
import java.util.*;

public class MapExample {
    public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<>();
        map.put(1, "Apple");
        map.put(2, "Banana");
        map.put(3, "Cherry");

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

Output:

plaintext
Copy code
```
{1=Apple, 2=Banana, 3=Cherry}
```

2️⃣ Commonly Used Implementations of `Map` in Java

Implementation	Ordering	Performance	Null Key	Use Case
`HashMap`	No order	Fast (`O(1)`)	✅ Yes (1 `null` key)	Best for general-purpose use
`TreeMap`	Sorted (Ascending)	Slower (`O(log n)`)	❌ No	Best when sorting by keys is required
`LinkedHashMap`	Insertion Order	Medium (`O(1)`)	✅ Yes	Best when maintaining order is important

---

3️⃣ Difference Between `HashMap` and `TreeMap`

Feature	HashMap	TreeMap
Ordering	Unordered	Sorted (Ascending order)
Performance	Faster (`O(1)`)	Slower (`O(log n)`)
Allows `null` Key?	✅ Yes (One `null` key)	❌ No
Implementation	Uses **Hash Table**	Uses **Red-Black Tree**

✅ **Use `HashMap`** when fast lookups are needed.
✅ **Use `TreeMap`** when sorting keys is required.

4️⃣ How to Check if a Key Exists in a `Map`?

You can use **`containsKey()`** method:

java
Copy code
```
if (map.containsKey(2)) {
    System.out.println("Key 2 exists!");
}
```

🔹 Generics in Java

5️⃣ What are Generics in Java?

Generics allow you to create **classes, interfaces, and methods** with a type parameter.
They help in **type safety, code reusability, and eliminating type casting**.

Example Without Generics (Using `Object`)

java
Copy code

```java
import java.util.ArrayList;

public class WithoutGenerics {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add("Java");
        list.add(100);  // No type safety

        String str = (String) list.get(0);  // Type casting required
        System.out.println(str);
    }
}
```

Example With Generics

java
Copy code

```java
import java.util.ArrayList;

public class WithGenerics {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Java");
        // list.add(100);  // Compile-time error

        String str = list.get(0);  // No type casting required
        System.out.println(str);
    }
}
```

6️⃣ Benefits of Using Generics in Java

✅ **Type Safety** – Ensures that only a specific type of object is added.
✅ **Code Reusability** – A single class can work with different data types.
✅ **Eliminates Type Casting** – No need for explicit type conversion.

7️⃣ What is a Generic Class in Java?

A **Generic Class** allows defining a **class with a type parameter**.

Example:

java
Copy code

```
class Box<T> {
    private T value;

    public void setValue(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}

public class GenericClassExample {
    public static void main(String[] args) {
        Box<String> strBox = new Box<>();
        strBox.setValue("Hello");
        System.out.println(strBox.getValue());

        Box<Integer> intBox = new Box<>();
        intBox.setValue(100);
        System.out.println(intBox.getValue());
    }
```

```
}
```

Output:

plaintext
Copy code
```
Hello
100
```

---

8 What is a Type Parameter in Java Generics?

A **Type Parameter** is a placeholder for a data type (e.g., `T`, `E`, `K`, `V`) that gets replaced by an actual type at runtime.

Example:

java
Copy code
```
class Container<T> {  // T is the type parameter
    private T data;

    public void setData(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }
}
```

Here, **`T`** is the **Type Parameter** that will be replaced by `String`, `Integer`, etc.

---

9 What is a Generic Method in Java?

A **Generic Method** is a method that has **its own type parameter**, independent of the class.

Example:

java
Copy code

```java
class Utility {
    public static <T> void printArray(T[] arr) {
        for (T item : arr) {
            System.out.print(item + " ");
        }
        System.out.println();
    }
}

public class GenericMethodExample {
    public static void main(String[] args) {
        Integer[] intArr = {1, 2, 3};
        String[] strArr = {"A", "B", "C"};

        Utility.printArray(intArr);
        Utility.printArray(strArr);
    }
}
```

Output:

plaintext
Copy code

```
1 2 3
A B C
```

🔟 Difference Between `ArrayList` and `ArrayList<T>`

Feature	`ArrayList` (Raw Type)	`ArrayList<T>` (Generic)
Type Safety	No	Yes
Requires Type Casting?	Yes	No

Performance	Slightly slower	Faster (no casting overhead)
Best Practice	Not recommended	Recommended