### **1. What is a Map in Java?**

A **Map** in Java is a part of the Collection Framework that stores key-value pairs.

* Keys must be unique, but values can be duplicated.
* Examples: HashMap, TreeMap, LinkedHashMap.
* Common operations include adding (put), retrieving (get), and checking for keys or values.

### **2. What are the commonly used implementations of Map in Java?**

1. **HashMap**:
   * Unordered, allows one null key and multiple null values.
   * Best for fast lookups and insertions.
2. **TreeMap**:
   * Maintains elements in sorted order based on the natural ordering of keys or a custom comparator.
   * Slower than HashMap.
3. **LinkedHashMap**:
   * Maintains insertion order.
   * Slightly slower than HashMap.
4. **Hashtable**:
   * Thread-safe but slower, doesn’t allow null keys or values.
   * Legacy implementation, rarely used now.

### **3. What is the difference between HashMap and TreeMap?**

| **Feature** | **HashMap** | **TreeMap** |
| --- | --- | --- |
| **Order** | Does not maintain order. | Maintains keys in sorted order. |
| **Performance** | Faster for insert and lookup operations. | Slower due to sorting overhead. |
| **Null Keys** | Allows one null key. | Does not allow null keys. |
| **Implementation** | Backed by a hash table. | Backed by a red-black tree. |

### **4. How do you check if a key exists in a Map in Java?**

Use the containsKey() method to check if a key exists.

java

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

map.put("One", 1);

System.out.println(map.containsKey("One")); // true

### **5. What are Generics in Java?**

Generics allow you to write type-safe code by specifying the type of objects a class, interface, or method can handle.

* Introduced in Java 5.
* Provides compile-time type checking and avoids ClassCastException.

**Example:**

java

ArrayList<String> list = new ArrayList<>();

list.add("Hello");

// list.add(123); // Compile-time error

### **6. What are the benefits of using Generics in Java?**

1. **Type Safety:** Prevents runtime ClassCastException.
2. **Code Reusability:** A generic class or method works with any type.
3. **Readability:** Reduces casting, making the code cleaner.
4. **Compile-Time Checking:** Errors are caught at compile time.

### **7. What is a Generic Class in Java?**

A **Generic Class** allows you to define a class with a parameterized type.

* **Example:**

java

class Box<T> {

private T value;

public void set(T value) {

this.value = value;

}

public T get() {

return value;

}

}

Box<Integer> box = new Box<>();

box.set(10);

System.out.println(box.get()); // 10

### **8. What is a Type Parameter in Java Generics?**

A **Type Parameter** is a placeholder for a specific type in a generic class, interface, or method.

* Commonly used type parameters are T (Type), E (Element), K (Key), V (Value).

**Example:**

Java

class Container<T> {

T item;

}

### **9. What is a Generic Method in Java?**

A **Generic Method** is a method that can operate on any type of data.

* Defined with a type parameter <T> before the return type.

**Example:**

java

public static <T> void printArray(T[] array) {

for (T element : array) {

System.out.println(element);

}

}

### **10. What is the difference between ArrayList and ArrayList<T>?**

| **Feature** | **ArrayList** | **ArrayList<T>** |
| --- | --- | --- |
| **Type** | Stores any type of object (non-generic). | Stores objects of a specific type (generic). |
| **Type Safety** | Not type-safe; requires casting. | Type-safe; no casting required. |
| **Compile-Time Checking** | No compile-time checking for type errors. | Provides compile-time checking. |
| **Usage** | Legacy-style, less preferred. | Preferred modern approach. |

**Example:**

**ArrayList:**java  
ArrayList list = new ArrayList();

list.add("Hello");

list.add(10); // Allowed, but unsafe.

**ArrayList<T>:**java  
ArrayList<String> list = new ArrayList<>();

list.add("Hello");

* // list.add(10); // Compile-time error.