TOP 20 INTERVIEW QUESTIONS ON JAVA GENERICS

1. **What are generics in Java?**
    - Explain that generics provide a way to create classes, interfaces, and methods that operate on types specified by the programmer, enhancing type safety and code reusability.

2. **What is the advantage of using generics in Java?**
    - Generics offer type safety at compile-time, eliminate casting, and provide the flexibility to work with different data types without creating multiple versions of the same class.

3. **Can you explain the syntax of generic classes and methods?**
    - Show how to declare generic classes (`class MyClass<T>`) and generic methods (`<T> T method(T param)`), where `<T>` is the type parameter.

4. **What is type erasure in generics?**
    - Type erasure is the process by which generic type information is removed at runtime, ensuring backward compatibility with older versions of Java.

5. **What is the difference between bounded and unbounded type parameters?**
    - **Unbounded** type parameter (`<T>`) allows any type.
    - **Bounded** type parameter (`<T extends Number>`) restricts the type to `Number` or its subclasses.

6. **What is the difference between `List<Object>` and `List<?>` in Java?**
    - `List<Object>` can accept any object, but `List<?>` (unbounded wildcard) can accept any generic list type. However, with `List<?>`, you cannot add elements (except `null`).

7. **What are upper-bounded wildcards in Java generics?**

    - `<? extends T>` allows a method to accept arguments of a type that is either `T` or a subclass of `T`.
8. **What are lower-bounded wildcards in Java generics?**
    - `<? super T>` allows a method to accept arguments of a type that is either `T` or a superclass of `T`.

9. **Can you overload methods when one uses generics and the other doesn't?**
    - Yes, method overloading can be done with generics as long as the signatures differ (in terms of parameter types).

10. **What is the purpose of the `T`, `E`, `K`, `V`, and `?` in generics?**

- These are type parameters: `T` (Type), `E` (Element), `K` (Key), `V` (Value), and `?` (Wildcard). They are placeholders for actual types.

11. **Why can't we use primitives in generics?**

- Generics work only with reference types, as type parameters need to be objects, not primitive types (e.g., `int`). Use wrapper classes (`Integer`, `Double`, etc.) instead.

12. **What are generic bounds and why are they useful?**

- Generic bounds (`<T extends Number>`) allow the type parameter to be restricted to specific types, ensuring that only certain types are passed to generic methods or classes.

13. **Can generic methods be static in Java?**

- Yes, generic methods can be static, but the type parameter must be declared before the return type (e.g., `public static <T> void method(T param)`).

14. **How can we create generic interfaces in Java?**

- Similar to generic classes, interfaces can be generic by using type parameters (`interface MyInterface<T>`). Implementing classes specify the actual type (`class MyClass implements MyInterface<String>`).

15. **What are raw types in Java generics?**
- A raw type is a generic type without specifying its type parameter. For example, using `List` instead of `List<String>`. This can lead to runtime errors and is not type-safe.

16. **What are the limitations of Java generics?**
- Type erasure leads to limitations like the inability to create instances of generic types, arrays of parameterized types, or static fields using generic types.

17. **Why can't you create an array of generic types?**
- Due to type erasure, creating an array of generic types would lead to runtime exceptions. You cannot verify the type at runtime, so Java doesn't allow it.

18. **What is a generic constructor in Java?**
- A constructor in a generic class or a specific generic constructor can use type parameters (`public <T> MyClass(T param)`).

19. **Can a generic class implement a non-generic interface?**
- Yes, a generic class can implement a non-generic interface. The generic type is used only for the class, not the interface.

20. **What is the diamond operator (`<>`) in Java?**
- The diamond operator was introduced in Java 7 to simplify the instantiation of generic types by allowing the compiler to infer the type parameters, e.g., `List<String> list = new ArrayList<>();`.