1. What are Generics in Java?

Generics in Java are a feature that allows you to define classes, interfaces, and methods with type parameters. This means that the type of data upon which the class, interface, or method operates can be specified as a parameter when the class or method is instantiated or invoked. Generics provide a way to create classes and methods that are type-safe and reusable.

2. What are the benefits of using Generics in Java?

Using Generics in Java provides several benefits:

- Type Safety: Generics enforce compile-time type checking, reducing the risk of ClassCastException at runtime by catching type errors during compilation.
- Code Reusability: Generics allow you to write a single class, method, or interface that can work with different types, reducing code duplication.
- Elimination of Casts: When retrieving elements from a collection that uses Generics, there is no need to cast the elements to their expected types, as the type is already known.
- Improved Code Readability: Generics make the code more expressive and easier to understand, as the type information is explicitly specified.

3. What is a Generic Class in Java?

A Generic Class in Java is a class that can operate on objects of various types while providing compile-time type safety. When defining a generic class, you specify one or more type parameters in angle brackets (`<>`) after the class name. For example:

Eg:-

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

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

    public T get() {
        return value;
    }
}
```

Eg:-

Here, `T` is a type parameter, and `Box` can store any type of object.

4. What is a Type Parameter in Java Generics?

A Type Parameter in Java Generics is a placeholder for a specific type that is provided when a generic class, interface, or method is instantiated or invoked. The type parameter is specified within angle brackets (`<>`) and can be any valid Java identifier. For example, in the generic class `Box<T>`, `T` is the type parameter.

5. What is a Generic Method in Java?
A Generic Method in Java is a method that can operate on different types specified by a type parameter. The type parameter is declared in the method signature before the return type. Generic methods allow for methods that are flexible and can be reused with different data types. For example:

Eg:-

```
public <T> void printArray(T[] array) {
    for (T element : array) {
        System.out.println(element);
    }
}
```

Eg:-
Here, `<T>` is the type parameter, and the method can print arrays of any type.

6. What is the difference between `ArrayList` and `ArrayList<T>`?
- `ArrayList`: This refers to the raw type of the `ArrayList` class, which existed before Java introduced Generics. Using the raw type does not provide type safety, meaning you can add any type of object to the list, and type checks are not performed at compile-time.

- `ArrayList<T>`: This refers to a generic `ArrayList` where `T` is a placeholder for a specific type. When you create an `ArrayList<T>`, you specify the type that the list will hold (e.g., `ArrayList<String>`). This provides type safety, ensuring that only objects of the specified type can be added to the list, and it eliminates the need for casting when retrieving elements.

In summary, `ArrayList<T>` is the generic, type-safe version of `ArrayList`, whereas `ArrayList` is the non-generic, raw type version.