1. What is input and output stream in Java?

Ans: [In Java, streams are the sequence of data that are read from the source and written to the destination1](https://www.programiz.com/java-programming/io-streams). [An input stream is used to read data from the source, such as a file, an array, or a keyboard](about:blank)[2](https://www.javatpoint.com/java-io). [An output stream is used to write data to the destination, such as a file, an array, or a screen2](https://www.javatpoint.com/java-io).

For example, in the following code, we use System.out to print a string to the standard output device (the screen):

class HelloWorld {

public static void main(String [] args) {

System.out.println ("Hello, World!");

}

}

1. What are the methods of OutputStream?

Ans: The OutputStream class provides different methods that are implemented by its subclasses. Here are some of the methods:

* write(int b) - writes the specified byte to the output stream
* write(byte[] array) - writes the bytes from the specified array to the output stream
* flush() - forces to write all data present in output stream to the destination
* close() - closes the output stream

1. What is serialization in Java?

Ans: As I mentioned before, serialization is a mechanism of converting the state of an object into a byte stream. It is used to persist the object or transfer it over a network. You can use the **ObjectOutputStream** class to perform serialization in Java.

1. What is the serializable interface in java?

Ans: [The **Serializable** interface is a marker interface (has no data member and method) that is used to mark Java classes so that the objects of these classes may get the capability of being serialized or deserialized1](https://www.javatpoint.com/serialization-in-java)[2](https://www.geeksforgeeks.org/serializable-interface-in-java/). [Serialization is a mechanism of converting the state of an object into a byte stream, and deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory3](https://www.geeksforgeeks.org/serialization-in-java/).

The **Serializable** interface must be implemented by the class whose object needs to be persisted. [The String class and all the wrapper classes implement the java.io.Serializable interface by default1](https://www.javatpoint.com/serialization-in-java).

1. What is deserializable in Java?

Ans: Deserialization is the reverse process of serialization, where the byte stream is used to recreate the actual Java object in memory. It is used to restore the state of the object or receive it from a network. You can use the **ObjectInputStream** class to perform deserialization in Java

1. How is serialization achieved in Java?

Ans: [Serialization in Java can be achieved by implementing the **java.io.Serializable** interface, which is a marker interface that indicates the class is serializable](about:blank)[1](https://www.scaler.com/topics/java/serialization-in-java/)[2](https://www.geeksforgeeks.org/serialization-in-java/). [To serialize an object, we can use the **writeObject()** method of the **ObjectOutputStream** class, which writes the state of the object into a byte stream1](https://www.scaler.com/topics/java/serialization-in-java/)[2](https://www.geeksforgeeks.org/serialization-in-java/)[3](https://www.baeldung.com/java-serialization-approaches). [To deserialize an object, we can use the **readObject()** method of the **ObjectInputStream** class, which reads the byte stream and recreates the object in memory](about:blank).

1. How is deserialization achieved in Java?

Ans: Deserialization is the reverse process of serialization, where the byte stream is used to recreate the actual Java object in memory. It is used to restore the state of the object or receive it from a network. You can use the **ObjectInputStream** class to perform deserialization in Java.

1. How can you avoid certain member variables of class from getting serialized?

Ans: You can avoid certain member variables of a class from getting serialized by marking them as either static or transient1. Static variables are not part of the state of an object, and transient variables are ignored by the serialization mechanism1. You can also implement the writeObject() and readObject() methods in the class and throw NotSerializableException from these methods to prevent the class or its subclass from serialization.

1. What classes are available in the Java IO File classes API?

Ans: [The Java IO File classes API contains the following classes which are relevant to working with files in Java1](https://jenkov.com/tutorials/java-io/files.html):

* File: This class represents a file or directory path in the file system. It can be used to obtain information about the file or directory, such as its name, size, permissions, etc. [It can also be used to create, delete, rename, or list files or directories2](https://docs.oracle.com/javase/8/docs/api/java/io/File.html)[3](https://docs.oracle.com/javase/7/docs/api/java/io/File.html).
* RandomAccessFile: This class allows reading and writing to a file at any position. [It can be used to access files in a random-access manner, such as skipping bytes, seeking to a specific position, or reading and writing data of different types1](https://jenkov.com/tutorials/java-io/files.html)[2](https://docs.oracle.com/javase/8/docs/api/java/io/File.html).
* FileInputStream: This class is a subclass of InputStream that reads bytes from a file. [It can be used to read binary data from a file, such as images, audio, video, etc1](https://jenkov.com/tutorials/java-io/files.html)[2](https://docs.oracle.com/javase/8/docs/api/java/io/File.html).
* FileReader: This class is a subclass of InputStreamReader that reads characters from a file. [It can be used to read text data from a file, such as plain text, XML, JSON, etc1](https://jenkov.com/tutorials/java-io/files.html)[2](https://docs.oracle.com/javase/8/docs/api/java/io/File.html).
* FileOutputStream: This class is a subclass of OutputStream that writes bytes to a file. [It can be used to write binary data to a file, such as images, audio, video, etc1](https://jenkov.com/tutorials/java-io/files.html)[2](https://docs.oracle.com/javase/8/docs/api/java/io/File.html).
* FileWriter: This class is a subclass of OutputStreamWriter that writes characters to a file. [It can be used to write text data to a file, such as plain text, XML, JSON, etc](https://jenkov.com/tutorials/java-io/files.html).

1. What is difference between externalizable and serialization interface.

Ans: [The main difference between Externalizable and Serializable interfaces in Java is that Externalizable allows the programmer to customize the serialization and deserialization process, while Serializable relies on the default mechanism provided by the JVM1](https://www.tutorialspoint.com/difference-between-serialization-and-externalization-in-java)[2](https://www.java67.com/2012/10/difference-between-serializable-vs-externalizable-interface.html)[3](https://www.baeldung.com/java-externalizable).

Some other differences are:

* Externalizable is not a marker interface, while Serializable is. This means that Externalizable defines two methods: writeExternal() and readExternal(), which must be implemented by the class that implements it. [Serializable does not define any methods2](https://www.java67.com/2012/10/difference-between-serializable-vs-externalizable-interface.html)[4](https://www.geeksforgeeks.org/difference-between-serializable-and-externalizable-in-java-serialization/)[5](https://stackoverflow.com/questions/817853/what-is-the-difference-between-serializable-and-externalizable-in-java).
* Externalizable requires a public no-arg constructor in the class that implements it, while Serializable does not. This is because Externalizable creates a new instance of the class using reflection and then calls the readExternal() method to populate its state. [Serializable uses the information about the class fields to restore the state of the object](about:blank)[1](https://www.tutorialspoint.com/difference-between-serialization-and-externalization-in-java)[2](https://www.java67.com/2012/10/difference-between-serializable-vs-externalizable-interface.html)[5](https://stackoverflow.com/questions/817853/what-is-the-difference-between-serializable-and-externalizable-in-java).
* Externalizable gives more control and flexibility to the programmer, but also more responsibility and complexity. [Serializable is simpler and easier to use, but also less efficient and secure](about:blank)
  1. What are Generics in Java?

Ans: Generics in Java are a feature that allows the programmer to write code that can work with different types of objects, without having to specify the exact type at every point. [Generics enable code reusability, type safety, and performance improvement1](https://www.tutorialspoint.com/java/java_generics.htm)[2](https://www.javatpoint.com/generics-in-java)[3](https://www.educba.com/what-is-generics-in-java/).

Some examples of generics in Java are:

* Generic classes: These are classes that can have one or more type parameters, which are placeholders for the actual types that will be used when creating an instance of the class. [For example, ArrayList<E> is a generic class that can store a list of any type of objects, where E is the type parameter1](https://www.tutorialspoint.com/java/java_generics.htm)[4](https://www.geeksforgeeks.org/generics-in-java/)[5](https://www.programiz.com/java-programming/generics).
* Generic methods: These are methods that can have one or more type parameters, which are placeholders for the actual types that will be used when calling the method. [For example, public static <T> void printArray(T[] array) is a generic method that can print an array of any type of objects, where T is the type parameter1](https://www.tutorialspoint.com/java/java_generics.htm)[4](https://www.geeksforgeeks.org/generics-in-java/)[5](https://www.programiz.com/java-programming/generics).
* Bounded types: These are type parameters that have some restrictions on what types they can accept. [For example, <T extends Number> is a bounded type parameter that can only accept subclasses of Number, such as Integer, Double, etc](https://www.tutorialspoint.com/java/java_generics.htm).
  1. What are the benefits of using Generics in Java?

Ans: Some of the benefits of using generics in Java are:

* Type safety: Generics allow you to specify the type of data that a class, method, or interface can work with, which prevents errors and bugs at compile time. [For example, if you use ArrayList<String> to store a list of strings, you cannot add an integer or any other type of object to it1](https://www.javatpoint.com/benefits-of-generics-in-java)[2](https://javagoal.com/advantages-of-generics-in-java/).
* Code reusability: Generics enable you to write code that can be used with different types of data, without having to repeat the same logic for each type. [For example, if you write a generic method that can sort an array of any type, you do not need to write separate methods for sorting arrays of integers, strings, etc](about:blank)[1](https://www.javatpoint.com/benefits-of-generics-in-java)[2](https://javagoal.com/advantages-of-generics-in-java/).
* Performance: Generics can improve the performance of your code by reducing the need for type casting and boxing/unboxing operations. Type casting is the process of converting one type of object to another, which can be costly and unsafe. [Boxing/unboxing is the process of wrapping primitive types (such as int, double, etc) in wrapper classes (such as Integer, Double, etc) and vice versa, which can also affect the performance and memory usage1](https://www.javatpoint.com/benefits-of-generics-in-java)[3](https://www.javatpoint.com/advantages-of-generics-in-java).
* Easier debugging: Generics can make debugging easier by providing more information about the data types and reducing the chances of runtime errors. [With generics, you can easily identify the source of errors and fix them quickly1](https://www.javatpoint.com/benefits-of-generics-in-java).
* Interoperability: Generics can make it easier to write code that can work with other Java libraries and frameworks that use generics. [For example, if you use the Java Collections Framework, which is based on generics, you can easily store and manipulate different types of objects in a consistent and type-safe way1](https://www.javatpoint.com/benefits-of-generics-in-java).
* Cleaner code: Generics can help you write cleaner and more readable code by reducing the amount of boilerplate code and type casting. [With generics, you can express your intentions more clearly and avoid unnecessary clutter in your code](about:blank).
  1. What is a Generic class in Java?

Ans: A generic class in Java is a class that can have one or more type parameters, which are placeholders for the actual types that will be used when creating an instance of the class. [A generic class can work with any type of data, without having to specify the exact type at every point](about:blank)[1](https://www.geeksforgeeks.org/generic-class-hierarchies-in-java/)[2](https://www.geeksforgeeks.org/generic-class-in-java/).

For example, ArrayList<E> is a generic class that can store a list of any type of objects, where E is the type parameter. [When you create an instance of ArrayList<E>, you can specify the actual type of objects that you want to store in the list, such as ArrayList<String>, ArrayList<Integer>, etc2](https://www.geeksforgeeks.org/generic-class-in-java/)[3](https://docs.oracle.com/javase/tutorial/java/generics/types.html)[4](https://www.javatpoint.com/generics-in-java).

A generic class is defined with the following format:

class name<T1, T2,..., Tn> {

/\*... \*/

}

The type parameter section, delimited by angle brackets (<>), follows the class name. [It specifies the type parameters (also called type variables) T1, T2,…, and Tn3](https://docs.oracle.com/javase/tutorial/java/generics/types.html).

* 1. What is a type parameter in Java Generics?

Ans: A type parameter in Java Generics is a placeholder for the actual type that will be used when creating an instance of a generic class or invoking a generic method. [A type parameter defines the formal type, usually named with one uppercase letter (e.gT, E, K, etc)](about:blank)[1](https://www.geeksforgeeks.org/generics-in-java/)[2](https://www.javatpoint.com/java-generics-questions).

For example, in the generic class ArrayList<E>, E is the type parameter that represents the type of objects that the list can store. [When you create an instance of ArrayList<E>, you can specify the actual type of E, such as ArrayList<String>, ArrayList<Integer>, etc](about:blank)[2](https://www.javatpoint.com/java-generics-questions).

A type parameter can also have bounds, which are restrictions on the types that can be used as arguments for the type parameter. Bounds are specified using the extends keyword in Java. For example, in the generic interface Comparable<T>, T is a type parameter that is bounded by the type Object. [This means that any type that implements Comparable<T> must be able to compare itself with any object](about:blank)[3](https://www.baeldung.com/java-generics-type-parameter-vs-wildcard).

* 1. What is a generic method in Java?

Ans: A generic method in Java is a method that can have one or more type parameters, which are placeholders for the actual types that will be used when invoking the method. [A generic method can work with any type of data, without having to specify the exact type at every point](about:blank)[1](https://www.tutorialspoint.com/java/java_generics.htm)[2](https://www.geeksforgeeks.org/generics-in-java/).

[For example, you can write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements](about:blank)[1](https://www.tutorialspoint.com/java/java_generics.htm).

A generic method is defined with the following format:

public <T1, T2,..., Tn> returnType methodName(T1 param1, T2 param2,..., Tn paramn) {

/\*... \*/

}

The type parameter section, delimited by angle brackets (<>), precedes the method’s return type. [It specifies the type parameters (also called type variables) T1, T2,…, and Tn](about:blank)[2](https://www.geeksforgeeks.org/generics-in-java/).

* 1. What is the difference between ArrayList and ArrayList<T??

Ans: ArrayList and ArrayList<T> are different ways of declaring a generic class in Java. ArrayList [is the raw type of the generic class, which means it does not specify any type parameterArrayList<T> is the parameterized type of the generic class, which means it specifies a type parameter T that represents the type of objects that the list can store](https://stackoverflow.com/questions/2309694/arraylist-vs-list-in-c-sharp" \t "_blank)[1](https://stackoverflow.com/questions/2309694/arraylist-vs-list-in-c-sharp)[2](https://stackoverflow.com/questions/18513308/what-is-the-difference-between-arraylist-arraylist-arraylistobject).

Using raw types, such as ArrayList, is discouraged because they are not type-safe and can lead to runtime errors. [Using parameterized types, such as ArrayList<T>, is recommended because they are type-safe and can prevent errors at compile time2](https://stackoverflow.com/questions/18513308/what-is-the-difference-between-arraylist-arraylist-arraylistobject).

For example, if you declare a list as ArrayList, you can add any type of object to it, such as String, Integer, or Object. However, when you retrieve an element from the list, you have to cast it to the appropriate type, which can cause a ClassCastException if the types do not match. [On the other hand, if you declare a list as ArrayList<String>, you can only add String objects to it, and when you retrieve an element from the list, you do not need to cast it because it is already a String](about:blank)[2](https://stackoverflow.com/questions/18513308/what-is-the-difference-between-arraylist-arraylist-arraylistobject)