Reference Data Types in Java

Java provides two types of [data types](https://www.javatpoint.com/java-data-types)

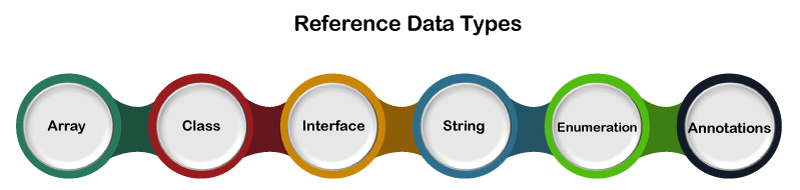
**primitive** and **reference** data type. The primitive data types are predefined in Java that serves as a fundamental building block while the **reference** data type refers to where data is stored.

In this section, we will discuss **what is a reference data type in Java**, and **how they differ from the primitive data type**.

In Java, **non-primitive** data types are known as **reference types**. In other words, a variable of class type is called **reference data type**. It contains the address (or reference) of dynamically created objects. For example, if Demo is a class and we have created its object **d**, then the variable d is known as a reference type.

It refers to objects. It is not pre-defined. It is created by the programmer if required. The reference types hold the references of objects. All **reference types** are a subclass of type **java.lang.Object**. It provides access to the objects stored in the memory.

The examples of reference data types are **class, interface, String, Arrays,** etc.



Java Reference Types

There are the following five types of reference types in Java:

|  |  |
| --- | --- |
| **Reference Type** | **Description** |
| [Class](https://www.javatpoint.com/object-and-class-in-java) | It is a set of instructions. It describes the content of the object. |
| [Array](https://www.javatpoint.com/array-in-java) | It provides the fixed-size data structure that stores the elements of the same type. |
| [Annotations](https://www.javatpoint.com/java-annotation) | It provides a way to associate metadata with program elements. |
| [Interface](https://www.javatpoint.com/interface-in-java) | It is implemented by Java classes. |
| [Enumeration](https://www.javatpoint.com/enum-in-java) | It is a special kind of class that is type-safe. Each element inside the enum is an instance of that enum. |

Reference vs Primitive Data Types

|  |  |
| --- | --- |
| **Reference Type** | **Primitive Type** |
| It is not pre-defined except the String. | It is pre-defined in Java. |
| All reference type begins with Uppercase letter. | All primitive type begins with a lowercase letter. |
| Non-primitive types have all the same size. | The size of a primitive type depends on the data type. |
| It is used to invoke or call methods. | We cannot invoke the method with a primitive type. |
| It can be null. | It cannot be null. It always has value. |
| Examples of reference data types are class, Arrays, String, Interface, etc. | Examples of primitive data types are int, float, double, Boolean, long, etc. |
| JVM allocates 8 bytes for each reference variable, by default. | Its size depends on the data type. |
| **Example**: Demo d1; | **Example**: int num=78; |

Memory Allocation and Garbage Collection

In Java, the [new keyword](https://www.javatpoint.com/new-keyword-in-java)

is used to create an **instance** of the class. In other words, it instantiates a class by allocating [memory](https://www.javatpoint.com/memory-management-in-java)

for a new object and returning a reference to that memory. Objects occupy memory in the [Java heap](https://www.javatpoint.com/java-heap)

space. We can also use the new keyword to create the **array object**.

1. ClassName objectName = **new** ClassName();

If there are no references to an object, the memory used by that object can be reclaimed during the [garbage collection](https://www.javatpoint.com/Garbage-Collection) process.

Conversion Between Primitive Type and Reference Type

The conversion of primitive type to reference type is called **[autoboxing](https://www.javatpoint.com/autoboxing-and-unboxing)**

and the conversion of reference type to primitive type is called **[unboxing](https://www.javatpoint.com/autoboxing-and-unboxing)**

.

Comparing Reference Type

We can also compare the reference types in Java. Java provides two ways to compare reference types:

By using the equal (==) operator

It compares the memory locations of the objects. If the memory address (reference) of both objects is the same, the objects are equal. Note that it does not compare the contents of the object. For example:

1. Demo d1 = **new** Demo("Atlanta"); //creating a reference of Demo class
2. Demo d2 = d1; //assigning the reference d1 into d2
3. **if**(d1=d2) //comparing reference

By using the String.equals() Method

The method belongs to the [String](https://www.javatpoint.com/java-string-equals)

class. It overrides the [equals()](https://www.javatpoint.com/java-string-equals)

method of the **[Object](https://www.javatpoint.com/object-class)**[class](https://www.javatpoint.com/object-class)

. It also uses the equal operator (==) for comparing the reference type. For example, consider the following code snippet:

1. String str1="JAVA";
2. String str2="JAVA";
3. String str3="java";
4. System.out.println(str1.equals(str2));     //returns true because content and case is same
5. System.out.println(str1.equals(str3));   //returns false because case is not same

Copying Reference Type

There are two possibilities when we copy reference types, either a copy of the reference to an object is made or an actual copy (creating a new copy) of the object is made.

In the following example, we have assigned a reference to the object. If we made any changes in the object, it will also reflect the reference and vice-versa.

1. Sum s1 = **new** Sum();
2. Sum s2= s1;