TYPE WRAPPERS IN JAVA

# Topics

1. What are wrapper classes
2. Need for wrapper classes
3. Autoboxing and unboxing
4. Immutability of primitive wrapper classes

# What are wrapper classes

A wrapper class in general is a class whose object includes other independent programming entities, such as objects and data types. A primitive wrapper class is a wrapper class whose object contains primitive data types.

# Need for wrapper classes

A wrapper class allows you to use various programming entities as objects, particularly data types as objects. This is useful because

* Some data structures such as ArrayList, Stack or Vector only accept objects
* All classes in java.util only handle objects
* Methods relevant for the data type can be accessed by the wrapper class object

# Autoboxing and unboxing

## Autoboxing

Autoboxing is the automatic conversion of a primitive data type variable or constant to an object of its corresponding class. For example

int i;

Integer n = i;

Here, the object of the class Integer is instantiated automatically without the new keyword, whereas deliberate instantiation would look like

int i;

Integer n = new Integer(i);

## Unboxing

Unboxing is the automatic conversion of a primitive wrapper object to a primitive data type variable or constant. It is the reverse process of autoboxing. For example

Integer n = 2; //Autoboxing

int i = n; //Unboxing

Here, the variable i is automatically assigned the corresponding field of n, without needing to specify that field.

# Immutability of primitive wrapper classes

This means that objects of primitive wrapper classes do not change when assigned different values in a different method after having been passed as an argument.

This is demonstrated in the program as follows

**class** PrimitiveWrapperImmutability

{

**static** **void** modify(**Integer** n)

{

n++;

}

**public** **static** **void** main(**String** args[])

{

**Integer** n = 3;

System.out.println("n before modification:\t" + n);

modify(n);

System.out.println("n after modification:\t" + n);

}

}

The output is

n before modification: 3

n after modification: 3

In the program, what happens is when the object n is passed to the method modify, n is unboxed into an integer n, incremented, then boxed into another object of the Integer class. This way, the object in the main method is left untouched.