# **Wrapper Classes in Java**

* Java is a strongly-typed language, which means that every variable must have a specific data type
* However, sometimes you need to treat primitive data types like objects, especially when working with collections, generics, or when you want to call methods on them. This is where wrapper classes come into play.
* The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.
* autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.
* Each primitive data type has a corresponding wrapper class, which encapsulates the primitive value and provides additional functionality.

Example-

* Integer (int)
* Double (double)
* Character (char)
* Boolean (boolean)
* Byte (byte)
* Short (short)
* Long (long)
* Float (float)

# **Why Are Wrapper Classes Needed?**

1. Null Values: Primitive data types cannot be null. In many scenarios, you need to represent the absence of a value. Wrapper classes allow you to use null to represent a lack of value.
2. Generics: Generics work with objects, not primitives. If you want to use collections, lists, or other generic data structures with primitive data types, you can use wrapper classes.
3. Method Overloading: In some cases, method overloading requires parameters to have different data types. Using wrapper classes, you can create overloaded methods that accept the corresponding wrapper class as a parameter.
4. Data Structures: Many Java libraries and data structures (e.g., ArrayList, HashMap) work with objects. Wrapper classes allow you to use primitives in these data structures.

## Wrapper Classes vs. Data Types

* Memory Overhead: Wrapper classes consume more memory than primitive data types because they store additional information about the object. For performance-critical applications, this can be a concern.
* Immutability: Wrapper objects are immutable, meaning their values cannot be changed once set. In contrast, primitive data types can be modified.
* Auto-Boxing and Auto-Unboxing: Java provides auto-boxing (converting primitives to their corresponding wrapper objects) and auto-unboxing (converting wrapper objects to primitives) for convenience. However, these operations can have a performance cost.

example1-

Integer age = 30; // Auto-boxing

int newAge = age + 5; // Auto-unboxing

Example2-

Integer value = null;

if (value == null) {

System.out.println("Value is null.");

}

## Common Functions in Wrapper Classes

* Each wrapper class provides methods to work with their respective primitive data type. For example, Integer has methods like intValue(), doubleValue(), and compareTo(). Here are some common functions in wrapper classes:
* valueOf(): Creates a wrapper object from a given primitive value.
* parseInt(): Parses a string and returns an int (available in Integer, Double, etc.).
* toString(): Converts the wrapper object to a string representation.
* compareTo(): Compares two wrapper objects.
* equals(): Checks if two wrapper objects are equal.
* hashCode(): Returns a hash code for the object.

Example3

// Java program to demonstrate Autoboxing

import java.util.ArrayList;

class Autoboxing {

public static void main(String[] args)

{

char ch = 'a';

// Autoboxing- primitive to Character object

// conversion

Character a = ch;

ArrayList<Integer> arrayList

= new ArrayList<Integer>();

// Autoboxing because ArrayList stores only objects

arrayList.add(25);

// printing the values from object

System.out.println(arrayList.get(0));

}

}

Example4

// Java program to demonstrate Unboxing

import java.util.ArrayList;

class Unboxing {

public static void main(String[] args)

{

Character ch = 'a';

// unboxing - Character object to primitive

// conversion

char a = ch;

ArrayList<Integer> arrayList

= new ArrayList<Integer>();

arrayList.add(24);

// unboxing because get method returns an Integer

// object

int num = arrayList.get(0);

// printing the values from primitive data types

System.out.println(num);

}

}

Example5

// Java program to demonstrate Wrapping and UnWrapping

// in Classes

import java.io.\*;

class GFG {

public static void main(String[] args)

{

// byte data type

byte a = 1;

// wrapping around Byte object

Byte byteobj = new Byte(a);

// Use with Java 9

// Byte byteobj = Byte.valueOf(a);

// int data type

int b = 10;

// wrapping around Integer object

Integer intobj = new Integer(b);

// Use with Java 9

// Integer intobj = Integer.valueOf(b);

// float data type

float c = 18.6f;

// wrapping around Float object

Float floatobj = new Float(c);

// Use with Java 9

// Float floatobj = Float.valueOf(c);

// double data type

double d = 250.5;

// Wrapping around Double object

Double doubleobj = new Double(d);

// Use with Java 9

// Double doubleobj = Double.valueOf(d);

// char data type

char e = 'a';

// wrapping around Character object

Character charobj = e;

// printing the values from objects

System.out.println(

"Values of Wrapper objects (printing as objects)");

System.out.println("\nByte object byteobj: "

+ byteobj);

System.out.println("\nInteger object intobj: "

+ intobj);

System.out.println("\nFloat object floatobj: "

+ floatobj);

System.out.println("\nDouble object doubleobj: "

+ doubleobj);

System.out.println("\nCharacter object charobj: "

+ charobj);

// objects to data types (retrieving data types from

// objects) unwrapping objects to primitive data

// types

byte bv = byteobj;

int iv = intobj;

float fv = floatobj;

double dv = doubleobj;

char cv = charobj;

// printing the values from data types

System.out.println(

"\nUnwrapped values (printing as data types)");

System.out.println("\nbyte value, bv: " + bv);

System.out.println("\nint value, iv: " + iv);

System.out.println("\nfloat value, fv: " + fv);

System.out.println("\ndouble value, dv: " + dv);

System.out.println("\nchar value, cv: " + cv);

}

}

Example6

// Java Program to implement

// Custom wrapper class

import java.io.\*;

// wrapper class

class Maximum {

private int maxi = 0;

private int size = 0;

public void insert(int x)

{

this.size++;

if (x <= this.maxi)

return;

this.maxi = x;

}

public int top() { return this.maxi; }

public int elementNumber() { return this.size; }

};

//

class GFG {

public static void main(String[] args)

{

Maximum x = new Maximum();

x.insert(12);

x.insert(3);

x.insert(23);

System.out.println("Maximum element: " + x.top());

System.out.println("Number of elements inserted: "

+ x.elementNumber());

}

}