**Wrapper Classes in Java**

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

**Need of Wrapper Classes**

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) and [Vector](https://www.geeksforgeeks.org/vector-vs-arraylist-java/), store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

**Primitive Data types and their Corresponding Wrapper class**



// Java program to demonstrate Wrapping and UnWrapping

// in Java Classes

class WrappingUnwrapping

{

public static void main(String args[])

{

// byte data type

byte a = 1;

// wrapping around Byte object

Byte byteobj = new Byte(a);

// int data type

int b = 10;

//wrapping around Integer object

Integer intobj = new Integer(b);

// float data type

float c = 18.6f;

// wrapping around Float object

Float floatobj = new Float(c);

// double data type

double d = 250.5;

// Wrapping around Double object

Double doubleobj = new Double(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("Byte object byteobj: " + byteobj);

System.out.println("Integer object intobj: " + intobj);

System.out.println("Float object floatobj: " + floatobj);

System.out.println("Double object doubleobj: " + doubleobj);

System.out.println("Character 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("Unwrapped values (printing as data types)");

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

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

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

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

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

}

}

**Output:**

Values of Wrapper objects (printing as objects)

Byte object byteobj: 1

Integer object intobj: 10

Float object floatobj: 18.6

Double object doubleobj: 250.5

Character object charobj: a

Unwrapped values (printing as data types)

byte value, bv: 1

int value, iv: 10

float value, fv: 18.6

double value, dv: 250.5

char value, cv: a

# Java Comparator interface

**Java Comparator interface** is used to order the objects of a user-defined class.

This interface is found in java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, rollno, name, age or anything else.

### Methods of Java Comparator Interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| public int compare(Object obj1, Object obj2) | It compares the first object with the second object. |
| public boolean equals(Object obj) | It is used to compare the current object with the specified object. |
| public boolean equals(Object obj) | It is used to compare the current object with the specified object. |

**import** java.util.\*;

**class** NameComparator **implements** Comparator{

**public** **int** compare(Object o1,Object o2){

Student s1=(Student)o1;

Student s2=(Student)o2;

**return** s1.name.compareTo(s2.name);

}

}

**import** java.util.\*;

**class** AgeComparator **implements** Comparator{

**public** **int** compare(Object o1,Object o2){

Student s1=(Student)o1;

Student s2=(Student)o2;

**if**(s1.age==s2.age)

**return** 0;

**else** **if**(s1.age>s2.age)

**return** 1;

**else**

**return** -1;

}

}

**import** java.util.\*;

**import** java.io.\*;

**class** Simple{

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

   ArrayList al=**new** ArrayList();

al.add(**new** Student(101,"Vijay",23));

al.add(**new** Student(106,"Ajay",27));

al.add(**new** Student(105,"Jai",21));

# Java Comparable interface

Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.

### compareTo(Object obj) method

**public int compareTo(Object obj):** It is used to compare the current object with the specified object. It returns

* positive integer, if the current object is greater than the specified object.
* negative integer, if the current object is less than the specified object.
* zero, if the current object is equal to the specified object.

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

### Collections class

**Collections** class provides static methods for sorting the elements of collections. If collection elements are of Set or Map, we can use TreeSet or TreeMap. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

System.out.println("Sorting by Name");

Collections.sort(al,**new** NameComparator());

Iterator itr=al.iterator();

**while**(itr.hasNext()){

Student st=(Student)itr.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

System.out.println("Sorting by age");

Collections.sort(al,**new** AgeComparator());

Iterator itr2=al.iterator();

**while**(itr2.hasNext()){

Student st=(Student)itr2.next();

System.out.println(st.rollno+" "+st.name+" "+st.age);

}

}

}