**Let’s fight “Primitives” vs “Boxed Primitives” in Java**

<https://codecooks.com/primitivs-vs-boxedprimitives-java/>

As many of us might know, Java has Primitive data types like int, byte, boolean, short, char, long, float & double. Each one of them has its own corresponding Reference type or Boxed Primitives like Character for char, Long for long and similarly for others as well.

**1. Specifics**

* While Primitive data types can have only Functional values, on the contrary, their counterparts Boxed Primitives can have functional values or null
* Primitives are memory & time efficient than Boxed
* Each Boxed Primitive has an identity so essentially, if we have, for example “5.0 == 5.0” to be true in Primitives, then in Boxed Primitives “new Float(5.0) != new Float(5.0)”, so we should avoid “==” & instead prefer “.equals”
* Boxed Primitives have similar behavior as of an object
* Boxed Primitives are primarily used when we have to store primitive data types (like int, long etc) in Collections like an ArrayList of <Int> or a Map of <Char, Long> or any other

**2. Declarations**

// Ways to declare Boxed Primitives

Integer v1 = new Integer(20);

Integer v2 = new Integer("10");

Float f1 = new Float(3.45);

Double d1 = new Double(23.34);

Character c1 = new Character('t');

Short s1 = new Short("20");

// Primitive data types

int val4 = 40;

boolean flag = false;

float test = 20.02f;

double db = 30.03;

char c = 'f';

short s = 10;

**3. Which one to use in Collections-**

In Collections we use Boxed primitives as defining a collection with primitive data types would give compile time exception

// Primitives being used in Collections

List<Integer> abc = new ArrayList<>();

Map<Character, Long> ab = new HashMap<>();

// Compile time error as we cannot have a Collection of Primitive type

List<int> def = new ArrayList<>();

Map<char, long> mpex = new HashMap<>();

**4. Boxing v’s Unboxing-**

While Boxing is converting Primitive data type to its reference type like int to Integer or char to Character, UnBoxing is the reverse process of converting reference type to its primitive type. Java compiler internally does this job for us hence it is also called AutoBoxing

// Boxing

int testBox = 30;

Integer val3 = 30;

Integer val5 = testBox;

// Unboxing

int val6 = new Integer(10);

int val7 = new Integer(10).intValue();

**5. Cases when Compiler does Autoboxing?**

**5.1. When Passing Parameters to Method-**

// Autoboxing - Passing parameters to Method

// In below example first 'a' is boxed into Character & then it is Unboxed to char c2

char c2 = testing('a');

char c6 = testing1('b');

Character testing(Character c4) {

return c4;

}

**5.2. While Returning from Method-**

// While returning from Method from boxed to boxed primitive data type compiler automatically does Autoboxing

Character testing1(char c4) {

return c4;

}

**5.3. Arithmetic Operations-**

// AutoBoxing in Arithmetic Operations

Integer i7 = new Integer(10);

int i9 = i7 + 10;

**6. Memory Impact–**

* Primitives store actual values in stack, hence they are fast, while Boxed primitives store references to values in a heap hence they are slow
* There is no Garbage Collection for Primitives so they don’t hold up on memory

**7. Boxed Primitives like an Object**

All Boxed Primitives are wrapper classes for which we declare objects like – “Character c = new  Character(‘a’)”, here c is an object of Character class. Each Primitive wrapper class has its respective methods which we can use in our programs

// Boxed Primitives are like objects, declaring them exposes various methods

Integer value = new Integer(80);

//some examples

double a = value.doubleValue(); // convert into double value

Integer newmaxval = value.MAX\_VALUE; // Assigns max integer val

System.out.println(value.compare(4, 5)); // helps compare two integers

**8. “Number” Boxed Primitive-**

It is an abstract base class for all Numeric primitive types Integer, Float, Double, Byte & Short. In the below example, we have defined a List of “Number” Boxed Primitive and in further steps, adding different types of values, of its child class type

// Number Boxed Primitive

List<Number> numericPrim = new ArrayList<>();

numericPrim.add(45665); // or numericPrim.add(new Integer(324));

numericPrim.add(435.345); // or numericPrim.add(new Double(324));

numericPrim.add(new Float(45.34));

numericPrim.add(new Byte("435"));

numericPrim.add(new Short("34"));

for (Number n : numericPrim) {

System.out.println(n.getClass());

System.out.println(n);

}

**9. Java Default Values–**

Fields which are not defined are given default values by the Java compiler, though local or final variables need to be defined values by developers if not we will get a compiler exception, below are list of default values for each Primitive type

|  |  |
| --- | --- |
| **Variable** | **Default Value** |
| **int** | **0** |
| **byte** | **0** |
| **double** | **0d** |
| **Object** | **null** |
| **boolean** | **false** |
| **short** | **0** |
| **char** | ‘**\u0000**‘ |
| **long** | **0L** |

**10. Quick Bites-**

**Q1. Are Primitive data types thread safe?**

No, long and double are not thread safe. When multiple threads would access a long or double value with sync, it can cause problems

**Q2. Are Primitives “pass by value”?**

Yes, in Java primitives are “pass by value” and also since there are no pointers or references to access their memory address so we cannot also alter their references.

**Q3. Are primitives “immutable”?**

Yes, Java primitives are immutable. Their identity is defined by its value so if we have defined “int j = 1” and we can change j=2, that would mean a complete change in identity of j

**Q4. How are primitives stored?**

Depends on where we declare them, in case we declare them in methods and they are being out of scope after the method finishes then it is stored on the stack. But if variables or references are being used outside the function scope then it is usually stored on heap

**Q5. Can primitives be null?**

No, primitives cannot be null in Java

**Q6. Can arraylist store Primitives?**

As discussed earlier, it would not be possible as collections only store Boxed Primitives or Objects

For an additional guide on some of the best practices with Primitives and Boxed primitives, Please [**refer**](https://codecooks.com/best-practices-primitives-boxed-primitives/)