# Understanding Primitive Types and Wrapper Objects

The **key difference** between wrapper class and primitive type in Java is that **wrapper class is used to convert a primitive type to an object and object back to primitive type while a primitive type is a predefined data type provided by the Java programming language.**

Java defines eight primitive data types: byte, short, int, long, float, double, boolean and char. All other variables in java are object reference types.

Primitive types in Java are called **literals**. A literal is the source code representation of a fixed value in memory. Each primitive type varies in its size and the way in which it is stored.

Text

Description automatically generated

Primitive types in Java are **statically typed**, which means all variables must first be declared before they can be used.

**Primitive Data Type Default Values**

When an object is created that has primitive type fields, if their values are not defined they will receive default values.

Table

Description automatically generated

# Wrapper Classes

A wrapper class is an object that encapsulates a primitive type. Each primitive type has a corresponding wrapper:

Each wrapper class has **Object** as a superclass. Byte, Short, Integer, Long Float and Double have **Number** as their direct superclass. This means that each wrapper class can implement the methods of the [Object class](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) such as hashCode(), equals(Object obj), clone(), and toString().

Diagram

Description automatically generated

## **Wrapper Objects are Immutable**

All primitive wrapper objects in Java are final, which means they are immutable. When a wrapper object get its value modified, the compiler must create a new object and then reassign that object to the original.

A picture containing graphical user interface

Description automatically generated

**Autoboxing**

Introduced in Java 5.0, [Autoboxing](https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html) is the automatic conversion of primitive types to their corresponding object wrapper classes.

List<**Integer**> **numbers** = new ArrayList<Integer>();

for(**int** i = 0; i < 10; i++) {  
 numbers.add(i);   
}

Since primitive types cannot be used in Collections or Generics, each time i is added to numbers a new Integer object is created.

**Unboxing**

Likewise, unboxing is the automatic conversion of object wrapper types into their corresponding primitive types.

Integer a = new Integer(5);  
int b = a;

The compiler recognizes that unboxing is needed and does it automatically for you. Be careful, this automatic conversion can cause performance issues and unexpected behavior.

# Dangers of Autoboxing and Unboxing

Performance

Autoboxing and Unboxing can cause performance to suffer by creating intermediate objects which creates more work for the Garbage Collector.

**When to Use Primitive Types**

* When doing a large amount of calculations, primitive types are always faster — they have much less overhead.
* When you don’t want the variable to be able to be null.
* When you don’t want the default value to be null.
* If the method must return a value

**When to Use Wrapper Class**

* When you are using Collections or Generics — it is required
* If you want the MIN\_SIZE or MAX\_SIZE of a type.
* When you want the variable to be able to be null.
* When you want to default value to be null.
* If sometimes the method can return a null value.

# Verdict

Generally, choose primitive types over wrapper classes unless using a wrapper class is necessary. Primitive Types will never be slower than Wrapper Objects, however Wrapper Objects have the advantage of being able to be null.

Reference:

<https://medium.com/@bpnorlander/java-understanding-primitive-types-and-wrapper-objects-a6798fb2afe9>

<https://www.differencebetween.com/difference-between-wrapper-class-and-vs-primitive-type-in-java/>