1Z0-808 Exam Topic Reviewer

TopicId: 1006
Topic: Wrapper Classes and Autoboxing/Unboxing

August 5, 2025

Topic 1006: Wrapper Classes and Autoboxing/Unboxing

Thinking Like the Compiler: Primitives in a World of Objects

We've established that primitives are fast and simple. However, much of Java's power, like the Collections Framework (e.g., `ArrayList`), is built to work with objects, not primitives. How do we bridge this gap? With Wrapper Classes. Java 5 introduced automatic conversion features—autoboxing and unboxing—which are convenient but hide behaviors the exam will absolutely test you on.

What are Wrapper Classes?

For each of the eight primitive types, there is a corresponding class in the `java.lang` package. These classes "wrap" a primitive value inside an object.

Primitive Type	Wrapper
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

Notice the full spelling for `Integer` and `Character`. Since they are in `java.lang`, you never need to import them.

Autoboxing and Unboxing

This is the automatic conversion process that makes using wrappers seamless.

- **Autoboxing:** Automatic conversion from a primitive to its wrapper object.
- **Unboxing:** Automatic conversion from a wrapper object to its primitive value.

```
// Autoboxing: the primitive int 10 is converted to an Integer object
Integer wrapperInt = 10;

// Unboxing: the Integer object is converted to a primitive int
int primitiveInt = wrapperInt;

// A practical example with a list:
java.util.List<Integer> numbers = new java.util.ArrayList<>();
numbers.add(1); // Autoboxing: int 1 -> Integer object
numbers.add(Integer.valueOf(2)); // Same as above, but explicit

int first = numbers.get(0); // Unboxing: Integer object -> int primitive
```

The Ultimate Exam Trap: NullPointerException

What happens when you try to unbox a wrapper that is `null`? This is a classic exam question because it compiles perfectly but fails at runtime.

```
Integer myValue = null;
int primitiveValue = myValue; // Throws NullPointerException at runtime!

// It's equivalent to writing:
// int primitiveValue = myValue.intValue();
// Calling a method on a null reference causes the NPE.
```

If you see a `null` wrapper being assigned to a primitive, suspect a `NullPointerException`.

Object Caching: The Wrapper Pool

To improve performance, Java reuses common wrapper objects. You MUST know this for the exam. The behavior of the equality operator (`==`) changes based on the value being wrapped.

- **The Rule:** By default, `Integer` objects created through autoboxing or `Integer.valueOf()` for values from **-128 to 127** are cached. This means the JVM will return the *same exact object* for these values.
- **The Consequence:** For values in this range, `==` will return `true`. For values outside this range, the JVM creates new objects, so `==` will return `false`.

```
// In the cache range [-128, 127]
Integer a = 100;
Integer b = 100;
System.out.println(a == b); // true (same object from cache)

// Outside the cache range
Integer x = 128;
Integer y = 128;
System.out.println(x == y); // false (different objects created)

// Using 'new' always creates a new object, ignoring the cache
Integer p = new Integer(100);
Integer q = new Integer(100);
System.out.println(p == q); // false (new objects)

// The correct way to compare wrapper values
System.out.println(x.equals(y)); // true
```

Golden Rule for the Exam: Always use `.equals()` to compare the values of wrapper objects. Use `==` only for comparing primitives or checking if two references point to the same object.

Key Wrapper Methods: `parseXxx()` vs. `valueOf()`

You need to know the difference between these two ways of converting a `String` to a number.

- `public static primitive parseXxx(String s)`: Returns a **primitive**. Example: `int i = Integer.parseInt("123");`
- `public static Wrapper valueOf(String s)`: Returns a **wrapper object**. Example: `Integer i = Integer.valueOf("123");`

Both can throw a `NumberFormatException` if the string is not a valid number.

Key Takeaways for the 1Z0-808 Exam

- **The NPE Trap:** Unboxing a `null` wrapper is a runtime `NullPointerException`, not a compile error.
- **Integer Pool:** Memorize the range **-128 to 127**. Understand why `==` works for values inside it and fails for values outside it.
- **Always use `.equals()`:** To compare wrapper objects by their content, never use `==` unless you specifically want to check for reference identity.
- **`parseInt` vs `valueOf`**: Know that one returns a primitive and the other returns a wrapper object.