When to Use Autoboxing in Java

Autoboxing is the automatic conversion of primitive types into their corresponding wrapper classes (e.g., int to Integer, double to Double). While convenient, it should be used carefully to avoid unnecessary object creation and performance issues.

[Yes] When Autoboxing is Useful (Recommended Usage)

1. When Using Collections (e.g., List, Set, Map)
   - Collections only store objects, so primitives must be boxed.
   Example:
   ```
   List<Integer> numbers = new ArrayList<>();
   numbers.add(10);  // Autoboxing: int to Integer
   ```

2. When Using Generics
   - Generic types work with objects, not primitives.
   Example:
   ```
   public class Box<T> { private T value; }
   Box<Integer> intBox = new Box<>();
   intBox.setValue(100); // Autoboxing: int to Integer
   ```

3. When Using Wrapper Methods (valueOf, parseInt, etc.)
   - Wrapper class methods return objects.
   Example:
   ```
   Integer num = Integer.valueOf(50); // No need to manually cast
   ```

4. When Working with Streams and Functional Interfaces
   - Java 8+ features like Stream require wrapper classes.
   Example:
   ```
   List<Integer> nums = List.of(1, 2, 3);
   int sum = nums.stream().mapToInt(Integer::intValue).sum();
   ```

[No] When to Avoid Autoboxing (Performance Concerns)

1. In Performance-Critical Loops
   - Avoid autoboxing inside loops due to object creation.
   Bad:
   ```
   Integer sum = 0;
   for (int i = 0; i < 1000000; i++) { sum += i; } // Autoboxing issue
   ```
   Good:
   ```
   int sum = 0;
   for (int i = 0; i < 1000000; i++) { sum += i; } // No autoboxing
   ```

2. When Comparing Values (== vs. .equals())

- Integer objects may not always be equal due to caching.

Example:

```
Integer a = 500, b = 500;
System.out.println(a == b);  // false (different objects)
System.out.println(a.equals(b));  // true (same value)
```

3. When Using Large HashMaps or Lists

- Autoboxing increases memory usage.

Example:

```
Map<Integer, Integer> map = new HashMap<>();
for (int i = 0; i < 1_000_000; i++) { map.put(i, i); } // Unnecessary boxing
```

[Yes] Summary: When to Use and Avoid Autoboxing

Scenario	Use Autoboxing?	Reason
Storing values in List<Integer>	[Yes] Yes	Collections require objects
Using generic methods (Box<T>)	[Yes] Yes	Generics work with objects
Functional programming (Stream)	[Yes] Yes	Stream API requires objects
Performance-critical loops	[No] No	Avoid unnecessary object creation
Comparing numbers (Integer == Integer)	[No] No	Use .equals() instead
Large HashMaps (Map<Integer, Integer>)	[No] No	Can cause high memory usage

[Note] Golden Rule:

- Use primitives (int, double, etc.) whenever possible.
- Use wrapper classes (Integer, Double, etc.) only when necessary.