Java Wrapper Classes Cheat Sheet

What are Wrapper Classes?

Wrapper classes convert primitive data types into objects. They "wrap" primitive values in an object so they can be used where objects are required.

Primitive to Wrapper Mapping

Primitive Type	Wrapper Class	Size	Example
byte	Byte	8-bit	Byte b = 10;
short	Short	16-bit	Short s = 100;
int	Integer	32-bit	Integer i = 1000;
long	Long	64-bit	Long l = 10000L;
float	Float	32-bit	Float f = 3.14f;
double	Double	64-bit	Double d = 3.14159;
char	Character	16-bit	Character c = 'A';
boolean	Boolean	1-bit	Boolean bool = true;

Why Use Wrapper Classes?

java

```java
// 1. Collections require objects, not primitives
ArrayList<int> list1 = new ArrayList<>();  // ERROR!
ArrayList<Integer> list2 = new ArrayList<>();  // Correct

// 2. Nullability
int primitive = null;  // ERROR! Primitives cannot be null
Integer wrapper = null;  // Correct - Objects can be null

// 3. Utility methods
int num = Integer.parseInt("123");  // String to int
String str = Integer.toString(123);  // int to String

// 4. Generics require objects
HashMap<int, String> map1 = new HashMap<>();  // ERROR!
HashMap<Integer, String> map2 = new HashMap<>();  // Correct

// 5. Method parameters that require Object
void processObject(Object obj) { }
processObject(10);  // Auto-boxing: int -> Integer
```

Creating Wrapper Objects

1. Using Constructors (Deprecated since Java 9)

java

```java
Integer num1 = new Integer(100);  // Deprecated
Double d1 = new Double(3.14);     // Deprecated
Character c1 = new Character('A'); // Deprecated
Boolean b1 = new Boolean(true);   // Deprecated
```

2. Using valueOf() Method (Recommended)

java

```java
Integer num2 = Integer.valueOf(100);
Double d2 = Double.valueOf(3.14);
Character c2 = Character.valueOf('A');
Boolean b2 = Boolean.valueOf(true);

// From String
Integer num3 = Integer.valueOf("123");
Double d3 = Double.valueOf("3.14");
Boolean b3 = Boolean.valueOf("true");
```

3. Auto-boxing (Automatic Conversion)

```java
// Primitive to Wrapper (Auto-boxing)
Integer num = 100;  // Automatically converts int to Integer
Double d = 3.14;    // Automatically converts double to Double
Character c = 'A';  // Automatically converts char to Character
Boolean b = true;   // Automatically converts boolean to Boolean
```

Auto-boxing and Unboxing

Auto-boxing (Primitive → Wrapper)

```java
int primitive = 50;
Integer wrapper = primitive;  // Auto-boxing

// In collections
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(10);  // Auto-boxes int to Integer
numbers.add(20);
numbers.add(30);
```

Unboxing (Wrapper → Primitive)

```java
Integer wrapper = 100;
int primitive = wrapper;  // Unboxing

// In operations
Integer a = 10;
Integer b = 20;
int sum = a + b;  // Both unboxed to int, then added
```

Auto-boxing/Unboxing in Operations

```java
```

```java
Integer x = 5;  // Auto-boxing
Integer y = 10;  // Auto-boxing

// Unboxing happens during arithmetic
Integer result = x + y;  // x and y unboxed, result auto-boxed

// Comparison
if (x < y) {  // Both unboxed for comparison
    System.out.println("x is less than y");
}
```

Common Methods

Integer Class

java

```java
// Parsing
int num = Integer.parseInt("123");  // String to int
Integer obj = Integer.valueOf("123");  // String to Integer

// Conversion
String str = Integer.toString(123);  // int to String
String binary = Integer.toBinaryString(10);  // "1010"
String hex = Integer.toHexString(255);  // "ff"
String octal = Integer.toOctalString(8);  // "10"

// Comparison
Integer a = 10;
Integer b = 20;
int compare = a.compareTo(b);  // -1 (a < b)
int compare2 = Integer.compare(10, 20);  // -1

// Get primitive value
int value = a.intValue();

// Constants
int maxValue = Integer.MAX_VALUE;  // 2147483647
int minValue = Integer.MIN_VALUE;  // -2147483648
int size = Integer.SIZE;  // 32 bits
int bytes = Integer.BYTES;  // 4 bytes

// Math operations
int max = Integer.max(10, 20);  // 20
int min = Integer.min(10, 20);  // 10
int sum = Integer.sum(10, 20);  // 30

// Check if parseable
try {
    int n = Integer.parseInt("abc");  // NumberFormatException
} catch (NumberFormatException e) {
    System.out.println("Not a valid integer");
}
```

Double Class

java

```java
// Parsing
double num = Double.parseDouble("3.14");
Double obj = Double.valueOf("3.14");

// Conversion
String str = Double.toString(3.14);

// Comparison
Double d1 = 3.14;
Double d2 = 2.71;
int compare = d1.compareTo(d2);  // 1 (d1 > d2)
int compare2 = Double.compare(3.14, 2.71);  // 1

// Get primitive value
double value = d1.doubleValue();

// Constants
double maxValue = Double.MAX_VALUE;
double minValue = Double.MIN_VALUE;
double posInf = Double.POSITIVE_INFINITY;
double negInf = Double.NEGATIVE_INFINITY;
double notANumber = Double.NaN;

// Special checks
boolean isNaN = Double.isNaN(Double.NaN);  // true
boolean isInfinite = Double.isInfinite(Double.POSITIVE_INFINITY);  // true
boolean isFinite = Double.isFinite(3.14);  // true

// Math operations
double max = Double.max(3.14, 2.71);  // 3.14
double min = Double.min(3.14, 2.71);  // 2.71
double sum = Double.sum(3.14, 2.71);  // 5.85
```

Character Class

```java
```

```java
char ch = 'A';

// Type checking
boolean isLetter = Character.isLetter(ch);  // true
boolean isDigit = Character.isDigit('5');  // true
boolean isWhitespace = Character.isWhitespace(' ');  // true
boolean isUpperCase = Character.isUpperCase('A');  // true
boolean isLowerCase = Character.isLowerCase('a');  // true
boolean isLetterOrDigit = Character.isLetterOrDigit('5');  // true

// Case conversion
char upper = Character.toUpperCase('a');  // 'A'
char lower = Character.toLowerCase('A');  // 'a'

// Get numeric value
int digit = Character.getNumericValue('5');  // 5
int hexValue = Character.getNumericValue('A');  // 10

// Comparison
Character c1 = 'A';
Character c2 = 'B';
int compare = c1.compareTo(c2);  // -1
int compare2 = Character.compare('A', 'B');  // -1

// Constants
char minValue = Character.MIN_VALUE;  // '\u0000'
char maxValue = Character.MAX_VALUE;  // '\uffff'

// String conversion
String str = Character.toString('A');  // "A"
```

Boolean Class

java

```java
// Creating
Boolean b1 = Boolean.valueOf(true);
Boolean b2 = Boolean.valueOf("true");  // Case insensitive

// Parsing
boolean bool = Boolean.parseBoolean("true");  // true
boolean bool2 = Boolean.parseBoolean("yes");  // false (only "true" is true)

// Comparison
Boolean b3 = true;
Boolean b4 = false;
int compare = b3.compareTo(b4);  // 1
int compare2 = Boolean.compare(true, false);  // 1

// Get primitive value
boolean value = b3.booleanValue();

// Constants
Boolean trueObj = Boolean.TRUE;
Boolean falseObj = Boolean.FALSE;

// Logical operations
boolean and = Boolean.logicalAnd(true, false);  // false
boolean or = Boolean.logicalOr(true, false);  // true
boolean xor = Boolean.logicalXor(true, false);  // true

// String conversion
String str = Boolean.toString(true);  // "true"
```

Long Class

java

```java
// Parsing
long num = Long.parseLong("1234567890");
Long obj = Long.valueOf("1234567890");

// Conversion
String str = Long.toString(1234567890L);
String binary = Long.toBinaryString(10L);
String hex = Long.toHexString(255L);

// Constants
long maxValue = Long.MAX_VALUE;  // 9223372036854775807
long minValue = Long.MIN_VALUE;  // -9223372036854775808
int size = Long.SIZE;  // 64 bits
int bytes = Long.BYTES;  // 8 bytes

// Math operations
long max = Long.max(100L, 200L);
long min = Long.min(100L, 200L);
long sum = Long.sum(100L, 200L);
```

Float Class

java

```java
// Parsing
float num = Float.parseFloat("3.14");
Float obj = Float.valueOf("3.14");

// Conversion
String str = Float.toString(3.14f);

// Constants
float maxValue = Float.MAX_VALUE;
float minValue = Float.MIN_VALUE;
float posInf = Float.POSITIVE_INFINITY;
float negInf = Float.NEGATIVE_INFINITY;
float notANumber = Float.NaN;

// Special checks
boolean isNaN = Float.isNaN(Float.NaN);
boolean isInfinite = Float.isInfinite(Float.POSITIVE_INFINITY);
boolean isFinite = Float.isFinite(3.14f);
```

Caching and Object Pool

Integer Caching (-128 to 127)

java

```java
// Cached values (same object reference)
Integer a = 100;
Integer b = 100;
System.out.println(a == b);  // true (same object)

// Outside cache range (different objects)
Integer c = 1000;
Integer d = 1000;
System.out.println(c == d);  // false (different objects)

// Always use equals() for value comparison
System.out.println(c.equals(d));  // true (same value)
```

Why Caching Matters

java

```java
// DON'T use == for wrapper comparison
Integer x = 200;
Integer y = 200;
if (x == y) {  // false - different objects
    System.out.println("Equal");
}

// DO use equals() for wrapper comparison
if (x.equals(y)) {  // true - same value
    System.out.println("Equal");
}

// For primitives, == is fine
int p = 200;
int q = 200;
if (p == q) {  // true
    System.out.println("Equal");
}
```

Practical Examples

Example 1: Collections with Wrappers

java

```java
import java.util.*;

public class WrapperCollections {
    public static void main(String[] args) {
        // ArrayList with Integer
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(10);  // Auto-boxing
        numbers.add(20);
        numbers.add(30);

        // Accessing (Unboxing)
        int first = numbers.get(0);  // Unboxing

        // Iteration
        for (Integer num : numbers) {
            System.out.println(num);  // Auto-unboxing
        }

        // HashMap with wrappers
        HashMap<Integer, String> map = new HashMap<>();
        map.put(1, "One");    // Auto-boxing
        map.put(2, "Two");
        map.put(3, "Three");

        String value = map.get(2);  // Auto-unboxing for key
    }
}
```

Example 2: Type Conversion

java

```java
public class TypeConversion {
    public static void main(String[] args) {
        // String to primitive
        String strNum = "123";
        int intNum = Integer.parseInt(strNum);
        double doubleNum = Double.parseDouble("3.14");
        boolean bool = Boolean.parseBoolean("true");

        // Primitive to String
        String str1 = Integer.toString(123);
        String str2 = Double.toString(3.14);
        String str3 = Boolean.toString(true);
        String str4 = String.valueOf(123);  // Alternative

        // Wrapper to primitive
        Integer wrapperInt = 100;
        int primitiveInt = wrapperInt.intValue();

        // Primitive to wrapper
        int primitive = 50;
        Integer wrapper = Integer.valueOf(primitive);

        // Between number types
        Integer intObj = 10;
        Double doubleObj = intObj.doubleValue();
        Long longObj = intObj.longValue();
    }
}
```

Example 3: Number Validation

java

```java
public class NumberValidation {
    public static Integer parseInteger(String str) {
        try {
            return Integer.parseInt(str);
        } catch (NumberFormatException e) {
            System.out.println("Invalid integer: " + str);
            return null;
        }
    }

    public static Double parseDouble(String str) {
        try {
            return Double.parseDouble(str);
        } catch (NumberFormatException e) {
            System.out.println("Invalid double: " + str);
            return null;
        }
    }

    public static void main(String[] args) {
        Integer num1 = parseInteger("123");  // Valid
        Integer num2 = parseInteger("abc");  // Invalid (null)

        if (num1 != null) {
            System.out.println("Valid number: " + num1);
        }
    }
}
```

Example 4: Math Operations

java

```java
public class WrapperMath {
    public static void main(String[] args) {
        Integer a = 10;
        Integer b = 20;

        // Arithmetic (auto-unboxing)
        Integer sum = a + b;  // 30
        Integer diff = b - a;  // 10
        Integer prod = a * b;  // 200
        Integer quot = b / a;  // 2

        // Comparison
        boolean isGreater = a > b;  // false
        boolean isEqual = a.equals(b);  // false

        // Using wrapper methods
        Integer max = Integer.max(a, b);  // 20
        Integer min = Integer.min(a, b);  // 10

        // Null safety
        Integer c = null;
        // int result = c + a;  // NullPointerException!

        // Safe approach
        if (c != null) {
            int result = c + a;
        }
    }
}
```

Example 5: Stream Operations

java

```java
import java.util.*;
import java.util.stream.*;

public class WrapperStreams {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        // Filter even numbers
        List<Integer> evens = numbers.stream()
            .filter(n -> n % 2 == 0)
            .collect(Collectors.toList());

        // Sum
        int sum = numbers.stream()
            .mapToInt(Integer::intValue)  // Convert to IntStream
            .sum();

        // Average
        OptionalDouble avg = numbers.stream()
            .mapToInt(Integer::intValue)
            .average();

        // Max and Min
        Optional<Integer> max = numbers.stream()
            .max(Integer::compareTo);

        Optional<Integer> min = numbers.stream()
            .min(Integer::compareTo);

        System.out.println("Sum: " + sum);
        System.out.println("Average: " + avg.orElse(0.0));
        System.out.println("Max: " + max.orElse(0));
        System.out.println("Min: " + min.orElse(0));
    }
}
```

Common Pitfalls

1. NullPointerException with Auto-unboxing

```java
```

```java
// DANGER: NullPointerException
Integer num = null;
int value = num;  // NullPointerException!

// SAFE: Check for null
Integer num2 = null;
if (num2 != null) {
    int value2 = num2;
}

// Or use Optional
Integer num3 = null;
int value3 = Optional.ofNullable(num3).orElse(0);
```

2. Using == Instead of equals()

java

```java
// WRONG: Comparing objects with ==
Integer a = 1000;
Integer b = 1000;
if (a == b) {  // false
    System.out.println("Equal");
}

// CORRECT: Use equals()
if (a.equals(b)) {  // true
    System.out.println("Equal");
}
```

3. Performance Issues

java

```java
// INEFFICIENT: Unnecessary boxing/unboxing
Integer sum = 0;
for (int i = 0; i < 1000000; i++) {
    sum += i;  // Boxing and unboxing in each iteration
}

// EFFICIENT: Use primitive
int sum2 = 0;
for (int i = 0; i < 1000000; i++) {
    sum2 += i;  // No boxing/unboxing
}
```

Quick Reference Table

Operation	Example	Result
Auto-boxing	`Integer i = 10;`	`Integer` object
Unboxing	`int x = new Integer(10);`	`10`
Parse String	`Integer.parseInt("10")`	`10` (primitive)
Value Of	`Integer.valueOf("10")`	`Integer` object
To String	`Integer.toString(10)`	`"10"`
Compare	`i1.compareTo(i2)`	`-1`, `0`, or `1`
Equals	`i1.equals(i2)`	`true`/`false`
Get Primitive	`i.intValue()`	primitive value

Best Practices

1. **Use primitives when possible** - Better performance
2. **Use equals() for comparison** - Not ==
3. **Check for null** - Before unboxing
4. **Prefer valueOf() over constructors** - More efficient (caching)
5. **Use appropriate wrapper** - Match your primitive type
6. **Be careful with collections** - Auto-boxing can hide performance issues
7. **Use primitive streams** - IntStream, DoubleStream, LongStream for better performance
8. **Understand caching** - Know when objects are reused (-128 to 127 for Integer)

Summary

Wrapper classes are essential for:

- Working with Collections
- Using Generics
- Handling null values
- Providing utility methods
- Converting between types

Remember: Wrapper classes are **immutable** - once created, their values cannot be changed!