CLASSES : Wrapper

Overview

For every **primitive data type**, there exists a corresponding **Wrapper Class**. These classes are used to represent primitive values as **objects**.

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

Constructors of Wrapper Classes

Nearly all wrapper classes include **two types of constructors**:

1. With primitive type as an argument.
2. With string type as an argument.

Examples

```java
Integer i1 = new Integer(10);         // primitive
Integer i2 = new Integer("10");      // string

Integer i3 = new Integer();           // ✖
Integer i4 = new Integer('a');        // ✖
Integer i5 = new Integer(10.5);       // ✖
Integer i6 = new Integer("ten");     // ✖ => RE: NumberFormatException

Double d1 = new Double(10.5);
Double d2 = new Double("10.5");
Double d3 = new Double('a');          // ✖
Double d4 = new Double(10.5f);
```

Boolean and Character Wrapper

- **Character class** accepts only primitive `char` type as argument.
- **Boolean class** accepts:
    - Primitive `true` or `false` (case-sensitive)
    - String: If passed, only “true” (case-insensitive) is evaluated as true; everything else becomes false.

Examples

```java
Character c1 = new Character('a');          // ✔
Character c2 = new Character("a");         // ✖

Boolean b1 = new Boolean(true);             // ✔
Boolean b2 = new Boolean(false);            // ✔
Boolean b3 = new Boolean("true");          // ✔
Boolean b4 = new Boolean("abc");           // false
```

AutoBoxing and AutoUnboxing

Definitions

- **Autoboxing**: Automatic conversion of primitive into its corresponding wrapper class object by the compiler.
- **Auto-unboxing**: Automatic conversion of wrapper class object into its corresponding primitive type.

Compiler Internals

```java
Integer I = 10;         // Compiler: Integer I = Integer.valueOf(10);
int i = I;              // Compiler: int i = I.intValue();
```

Important Concepts

- Autoboxing/unboxing was added in **Java 1.5**.
- `valueOf()` is used for autoboxing.
- `xxxValue()` is used for auto-unboxing.

Null Handling in Autounboxing

```java
class Demo {
    static Integer I;
    public static void main(String[] args) {
        int x = I;  // ❗ NullPointerException
    }
}
```

Wrapper Classes are Immutable

```java
Integer x = 10;
Integer y = x;
x++;
System.out.println(x);    // 11
System.out.println(y);    // 10
System.out.println(x == y); // false
```

Buffering in Wrapper Classes

JVM Buffer Optimization

- JVM maintains a **buffer of wrapper objects** for common values:
    - Range: `-128 to 127`
- Reuses objects from buffer when autoboxing.
- Outside the range, new objects are created.

Examples

```java
Integer a = 100;
Integer b = 100;
System.out.println(a == b);      // true (from buffer)

Integer a1 = 1000;
Integer b1 = 1000;
System.out.println(a1 == b1);   // false (new objects)
```

Methods of Wrapper Classes

1. `valueOf()`

Used to **create wrapper class objects** from primitives or strings.

Syntax

```java
public static Wrapper valueOf(String s);
```

Examples

```java
Integer i = Integer.valueOf("10");
Double d = Double.valueOf("10.5");
Boolean b = Boolean.valueOf("abc");
System.out.println(i + " " + d + " " + b);  // 10 10.5 false
```

2. `xxxValue()`

Used to convert **wrapper objects into primitives**.

Examples:

```java
Integer I = new Integer(140);
System.out.println(I.byteValue());   // -116
System.out.println(I.shortValue());  // 140
System.out.println(I.intValue());    // 140
System.out.println(I.longValue());   // 140
System.out.println(I.floatValue());  // 140.0
System.out.println(I.doubleValue()); // 140.0
```

```java
Character c = new Character('a');
System.out.println(c.charValue());   // a

Boolean b = new Boolean(true);
System.out.println(b.booleanValue()); // true
```

3. `parseXxx()`

Used to convert **String to primitive** (not object).

- Available in every wrapper class except `Character`.

Syntax

```java
public static Primitive parseXxx(String s);
```

Example

```java
int i = Integer.parseInt("10");
double d = Double.parseDouble("20.5");
boolean b = Boolean.parseBoolean("abc");
System.out.println(i + " " + d + " " + b);  // 10 20.5 false
```

Method Overloading and Type Preference

Example 1: Priority Order in Overloading

```java
class Demo {
    public static void m1(Integer I) {
        System.out.println("Autoboxing");
    }
    public static void m1(double d) {
        System.out.println("Upcasting/Widening");
    }
    public static void main(String[] args) {
        int x = 10;
        m1(x);
    }
}
// Output: Upcasting/Widening
```

- *Upcasting (widening)* is preferred over **Autoboxing**.

Example 2: Varargs in Overloading

```java
class Demo {
    public static void m1(long l) {
        System.out.println("Hi");
    }
    public static void m1(int... x) {
        System.out.println("Bye");
    }
    public static void main(String[] args) {
        int x = 10;
        m1(x);  // Output: Hi
    }
}
// Typecasting has higher priority than varargs.
```

Summary

- Wrapper classes provide **object representations** for primitives.

- Java offers **autoboxing** and **auto-unboxing** to simplify conversion.
- The **valueOf()**, `xxxValue()`, and `parseXxx()` methods are critical for conversion operations.
- JVM uses **buffering for Integer objects** between -128 to 127 to optimize memory.
- Wrapper classes are **immutable**.
- **Overloading** resolves based on **widening > boxing > varargs**.