## **Break and Continue in loops in Java**

**🔹 break Statement in Java**

The break statement **exits the loop immediately**, skipping any remaining iterations.

**🔹 continue Statement in Java**

The continue statement **skips the current iteration** and jumps to the next one.

🔁 Using break and continue in Different Loops

**Real World Scenario:** A user has 3 chances to enter the correct ATM PIN. Stop as soon as the correct one is entered.  
  
  
import java.util.Scanner;

public class BreakInWhileLoop {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String correctPin = "1234";

int attempts = 0;

while (attempts < 3) {

System.out.print("Enter PIN: ");

String input = sc.nextLine();

attempts++;

if (input.equals(correctPin)) {

System.out.println("Access granted.");

break; // Stop loop on correct PIN

} else {

System.out.println("Incorrect PIN.");

}

}

if (attempts == 3) {

System.out.println("Card blocked due to 3 incorrect attempts.");

}

}

}

**Real World Scenario:** You're scanning products in a warehouse but **skip checking items that are out of stock**.

public class ContinueInDoWhileLoop {

public static void main(String[] args) {

int[] stock = {10, 0, 5, 0, 8}; // quantities of products

int i = 0;

do {

if (stock[i] == 0) {

i++;

continue; // Skip out-of-stock item

}

System.out.println("Checking product with quantity: " + stock[i]);

i++;

} while (i < stock.length);

}

}

**Real World Scenario:** A delivery system that operates only Monday to Friday.

public class ContinueInForLoop {

public static void main(String[] args) {

String[] days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

for (String day : days) {

if (day.equals("Saturday") || day.equals("Sunday")) {

continue; // Skip delivery on weekends

}

System.out.println("Delivering on: " + day);

}

}

}

**✅ Guess the Output – Java switch + Loop Statements**

**1] Basic Switch Statement**

int day = 3;

switch (day) {

case 1: System.out.println("Monday");

case 2: System.out.println("Tuesday");

case 3: System.out.println("Wednesday");

case 4: System.out.println("Thursday");

}

**2] Switch with Breaks**

int option = 2;

switch (option) {

case 1: System.out.println("Start"); break;

case 2: System.out.println("Stop"); break;

default: System.out.println("Invalid");

}

**3] Switch with String**

String size = "Medium";

switch (size) {

case "Small": System.out.println("S"); break;

case "Medium": System.out.println("M"); break;

case "Large": System.out.println("L"); break;

}

**4] Switch Default First**

int code = 5;

switch (code) {

default: System.out.println("Unknown");

case 1: System.out.println("One"); break;

}

**5] For Loop with Continue**

for (int i = 1; i <= 5; i++) {

if (i == 3)

continue;

System.out.print(i + " ");

}

**6] While with No Execution**

int i = 5;

while (i < 3) {

System.out.println(i);

i++;

}

**7] Do-While Executes Once**

int x = 10;

do {

System.out.println("Running");

} while (x < 5);

**8] Nested Loop (Multiplication Table)**

for (int i = 1; i <= 2; i++) {

for (int j = 1; j <= 3; j++) {

System.out.print(i \* j + " ");

}

System.out.println();

}  
  
**✅ What is the Enhanced for loop?**

Also known as the **for-each** loop, the enhanced for loop is used to iterate over **arrays** or **collections** in a simplified way.

**🔄 Syntax:**

for (datatype variable : arrayOrCollection) {

// use variable

}

**📌 Rules of Enhanced for Loop:**

1. **Works only with arrays or classes that implement Iterable** (e.g., ArrayList).
2. **Read-only access** to array elements — you can read but not modify the original array element (unless it's a reference to a mutable object).
3. You **can’t get the index** of the current element directly.
4. **No control over iteration** (e.g., skipping elements, going in reverse, or early exit beyond break).

**🧪 Simple Array Example:**

public class EnhancedForLoopExample {

public static void main(String[] args) {

int[] numbers = {10, 20, 30, 40, 50};

for (int num : numbers) {

System.out.println("Number: " + num);

}

}

}

**Output:**

Number: 10

Number: 20

Number: 30

Number: 40

Number: 50

**🌍 Real-World Example 1: Student Grades**

public class StudentGrades {

public static void main(String[] args) {

double[] grades = {85.5, 90.0, 76.5, 88.0};

for (double grade : grades) {

System.out.println("Grade: " + grade);

}

}

}

**🌍 Real-World Example 2: List of Cities**

public class CityList {

public static void main(String[] args) {

String[] cities = {"New York", "London", "Tokyo", "Paris"};

for (String city : cities) {

System.out.println("City: " + city);

}

}

}

**⚠️ Modifying Array Values?**

If you try to modify primitive values inside a for-each loop, it **won’t affect the original array**:

int[] arr = {1, 2, 3};

for (int num : arr) {

num = num \* 2; // This doesn't change the array

}

System.out.println(Arrays.toString(arr)); // Output: [1, 2, 3]

**✅ Use Enhanced For Loop When:**

* You just need to **read** each element.
* You don't care about the **index**.
* You want **clean and concise** code.

**🧠 Java Loops – MCQ Quiz**

**Q1. What is the output of the following code?**

int i = 1;

while (i < 3) {

System.out.print(i + " ");

i++;

}

A) 1 2 3  
B) 1 2  
C) 0 1 2  
D) Infinite loop

**Answer:** B) 1 2

**Q2. Which loop guarantees at least one execution of the loop body?**  
A) for loop  
B) while loop  
C) do-while loop  
D) None of the above

**Answer:** C) do-while loop

**Q3. What is the output of this code?**

for (int i = 0; i < 5; i += 2) {

System.out.print(i + " ");

}

A) 0 1 2 3 4  
B) 0 2 4  
C) 0 1 2 3 4 5  
D) 2 4

**Answer:** B) 0 2 4

**Q4. Which keyword is used to exit a loop prematurely in Java?**  
A) exit  
B) return  
C) break  
D) stop

**Answer:** C) break

**Q5. What is the output?**

int i = 0;

do {

System.out.print(i + " ");

i++;

} while (i < 3);

A) 1 2 3  
B) 0 1 2  
C) 0 1 2 3  
D) No output

**Answer:** B) 0 1 2

**Q6. Which of the following is a valid for loop?**  
A) for (int i = 0; i < 10; i++)  
B) for (i = 0; i < 10; i++)  
C) for int i = 0; i < 10; i++  
D) for (int i < 10; i++)

**Answer:** A) for (int i = 0; i < 10; i++)

**Q7. What does the continue statement do in a loop?**  
A) Stops the loop completely  
B) Skips the current iteration and moves to the next  
C) Ends the program  
D) Resets the loop counter

**Answer:** B) Skips the current iteration and moves to the next

**Q8. How many times will the loop run?**

for (int i = 10; i > 0; i -= 2) {

System.out.print(i + " ");

}

A) 10  
B) 5  
C) 2  
D) Infinite

**Answer:** B) 5

**Q9. Which loop is best suited when the number of iterations is known?**  
A) while  
B) do-while  
C) for  
D) None of the above

**Answer:** C) for

**Q10. What is the output?**

int count = 0;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 2; j++) {

count++;

}

}

System.out.print(count);

A) 6  
B) 5  
C) 3  
D) 2

**Answer:** A) 6

# **Wrapper Classes, Boxing & Unboxing in Java**

**1. Wrapper Classes in Java**

Wrapper classes are object representations of primitive data types. They allow primitives to be used in object-oriented contexts (e.g., collections, generics, and APIs that require objects).

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

**Why Use Wrapper Classes?**

* **Collections (e.g., ArrayList, HashMap)** require objects, not primitives.
* **Generics (e.g., List<Integer>)** work only with objects.
* **Nullable values** (primitives cannot be null, but wrappers can).
* **Utility methods** (e.g., Integer.parseInt(), Character.isLetter()).

**2. Boxing & Unboxing**

**Boxing (Primitive → Wrapper)**

Converting a primitive to its corresponding wrapper class.

**Types of Boxing:**

1. **Manual Boxing** (Explicit conversion)

int num = 10;

Integer boxedNum = Integer.valueOf(num); // Manual boxing

1. **AutoBoxing** (Java automatically does it)

int num = 10;

Integer boxedNum = num; // Auto-boxing (Java 5+)

**Unboxing (Wrapper → Primitive)**

Converting a wrapper object back to its primitive type.

**Types of Unboxing:**

1. **Manual Unboxing**

Integer boxedNum = 20;

int num = boxedNum.intValue(); // Manual unboxing

1. **Auto-Unboxing**

Integer boxedNum = 20;

int num = boxedNum; // Auto-unboxing

**3. Performance Considerations**

* **Boxing/Unboxing has a small overhead** (not a big issue in most cases).
* **Avoid in performance-critical loops** (use primitive arrays instead of List<Integer>).
* **Beware of NullPointerException** when unboxing null values.

Integer nullableValue = getNullableValue();

int value = nullableValue; // NPE if nullableValue is null

**📅 When Were Enums Introduced?**

* **Java 5** (March 2004, JDK 1.5)

Prior to Java 5, constants were typically represented by public static final int fields (the “constant interface” anti‑pattern). The enum construct gave us a first‑class, type‑safe way to define fixed sets of values.

**❓ Why Use Enums?**

* **Type safety**: You can’t pass an invalid value—only one of the defined constants.
* **Namespace management**: A clear grouping instead of loose static fields.
* **Built‑in utility**: Every enum gets values(), valueOf(String), ordinal(), plus the Enum API.
* **Extensibility**: You can give each constant fields, methods, and even implement interfaces, so they can carry behavior as well as data.

**✅ When to Use Enums**

* Any time you have a **fixed set of known values**, for example:
  + Days of the week
  + Order or payment statuses
  + Card suits or ranks
  + Configuration modes (DEBUG/INFO/WARN/ERROR)
* When you want to leverage a **switch** over those values, ensuring exhaustiveness checking.
* When each constant needs its own behavior or metadata (override a method per constant, store display strings, etc.).

**📋 Key Rules & Syntax**

1. **Declaration**

public enum Direction {

NORTH, EAST, SOUTH, WEST

}

* + Constants **must** come first, comma‑separated.
  + If you add fields or methods, terminate the constants list with a semicolon.

1. **Implicit Superclass**
   * Every enum implicitly extends java.lang.Enum<E>.
   * You **cannot** extend another class.
2. **Constructors**
   * You can define a constructor, but it’s always **private** (or package‑private).
   * Example:

public enum Status {

NEW(0), IN\_PROGRESS(1), DONE(2);

private final int code;

Status(int code) { this.code = code; }

public int code() { return code; }

}

1. **Methods**
   * You get values(), valueOf(String), ordinal(), and name() for free.
   * You can add your own instance or static methods.
2. **Interfaces**
   * Enums may **implement** interfaces:

public enum Operation implements BinaryOperator<Double> {

PLUS { public Double apply(Double a, Double b) { return a + b; } },

MINUS { public Double apply(Double a, Double b) { return a - b; } };

}

1. **Switch Compatibility**
   * Works seamlessly in both legacy switch statements and new switch expressions.

**🌍 Real‑World Examples**

**1. Order Status**

public enum OrderStatus {

NEW, PROCESSING, SHIPPED, DELIVERED, CANCELED;

}

**2. HTTP Status Codes with Metadata**

public enum HttpStatus {

OK(200, "Success"),

NOT\_FOUND(404, "Not Found"),

INTERNAL\_ERROR(500, "Server Error");

private final int code;

private final String reason;

HttpStatus(int code, String reason) {

this.code = code;

this.reason = reason;

}

public int code() { return code; }

public String reason() { return reason; }

}

**🧠 MCQs: Test Your Enum Knowledge**

1. **Which Java version added enum support?**  
   A) Java 1.4 B) Java 5 C) Java 6 D) Java 7

**Answer:** B) Java 5

1. **Which of these is illegal in an enum?**  
   A) Defining a private constructor  
   B) Extending another class  
   C) Implementing an interface  
   D) Overriding toString()

**Answer:** B) Extending another class

1. **What does Day.values() return?**  
   A) A List<Day> B) A Set<Day> C) A Day[] array D) An Iterator<Day>

**Answer:** C) A Day[] array

1. **Given**

public enum Color { RED, GREEN, BLUE }

**What is Color.valueOf("red")?**  
A) Color.RED B) null C) IllegalArgumentException D) Compiler error  
  
**Answer:** C) IllegalArgumentException

1. **How can you associate data with each enum constant?**  
   A) Define fields and a private constructor  
   B) Use a Map outside the enum  
   C) Enum constants cannot hold data  
   D) By subclassing each constant

**Answer:** A) Define fields and a private constructor

1. **Which method gives the declaration order of constants?**  
   A) name() B) ordinal() C) toString() D) compareTo()

**Answer:** B) ordinal()

## **Assignments for this week:**

**1. Parking Lot Slot Finder (break)**

**Scenario:** Ask the user to enter the status of 10 parking slots (1 for occupied, 0 for available). Reserve the **first available slot** and stop checking further.

📥 **Sample Input:**

Enter slot status (0 = available, 1 = occupied):

1 1 0 1 0 1 1 1 1 1

📤 **Expected Output:**

Reserving slot #3

**2. Scan Product Prices – Skip Invalid (Use continue)**

Ask the user to input prices of 5 products. Some prices may be negative due to input error. **Skip negative prices** and display only valid ones.

📥 **Sample Input:**

Enter 5 product prices:

12.99 -1.0 0.0 15.5 -3.75

📤 **Expected Output:**

Valid price: 12.99

Valid price: 0.0

Valid price: 15.5

**3. Sensor Monitoring – Stop on Critical Alert (Use break)**

Ask the user to enter temperature readings (up to 10). Stop immediately if any reading exceeds 100.

📥 **Sample Input:**

Enter 6 temperature readings:

45 60 78 102 87 90

📤 **Expected Output:**

Reading: 45

Reading: 60

Reading: 78

Critical temperature detected: 102°C – Stopping monitoring.

**4. Email Validation – Stop on Invalid Email (Use break)**

Ask the user to input 5 email addresses. If any email is invalid (missing "@"), stop processing.

📥 **Sample Input:**

Enter 5 emails:

alice@example.com

bob@domain.com

charlie.domain.com

dave@example.com

eve@test.com

📤 **Expected Output:**

Valid: alice@example.com

Valid: bob@domain.com

Invalid email found: charlie.domain.com – Stopping batch.

**5. Process Student Scores – Skip Failures (Use continue)**

Ask the user to enter scores of 5 students. Skip scores less than 40 (fail).

📥 **Sample Input:**

Enter 5 scores:

75 39 60 20 85

📤 **Expected Output:**

Passed: 75

Failed – skipped

Passed: 60

Failed – skipped

Passed: 85

**6. Sensor Reading Logger**

Take 10 temperature readings and store them as Double objects in a list. Find how many readings are above 37.5°C (fever cases).

📥 Input:

Enter 10 temperature readings:

36.2 37.8 38.1 36.5 39.0 37.4 36.9 38.2 37.0 36.8

📤 Output:

Total fever cases: 4

🧠 **Concepts Used:**

* Wrapper class Double
* Autoboxing from double to Double
* Unboxing during comparisons

**7. Voting Eligibility Checker**

Ask the user to input age values of people at an event. Store them as Integer. Count how many are eligible to vote (18+).

📥 Input:

Enter ages of 6 people:

17 21 16 45 18 19

📤 Output:

Eligible voters: 4

🧠 **Concepts Used:**

* Boxing int to Integer
* Unboxing for comparison (>= 18)

**8. ATM Cash Withdrawal Simulation**

Simulate an ATM machine. The user enters a PIN and can perform operations: check balance, deposit, withdraw, or exit. The loop continues until the user chooses to exit.

📥 Input:

Enter PIN: 1234

Choose option:

1. Check Balance

2. Deposit

3. Withdraw

4. Exit

Option: 2

Enter deposit amount: 500

Option: 1

Balance: 500

Option: 4

📤 Output:

Deposit successful.

Balance: 500

Thank you for using the ATM.

**9. Bus Fare Discount Calculator**

Take age and number of trips from user input. For each trip, apply the following fare rules:

* Age < 5 or > 60: 50% discount
* Else: full fare

📥 Input:  
Enter age: 45

Enter number of trips: 3

Trip 1 fare: 20

Trip 2 fare: 30

Trip 3 fare: 25

📤 Output:

Total after discount: 75

**10. Online Shopping Cart – Bulk Discount**

A user adds multiple items to their cart. Each item has a price and quantity. If quantity > 5, apply a 10% discount on that item.

📥 Input:

Enter number of items: 2

Item 1 price: 100

Item 1 quantity: 6

Item 2 price: 50

Item 2 quantity: 3

📤 Output:

Item 1 total after discount: 540.0

Item 2 total: 150.0

Final bill: 690.0

**11. Pyramid Pattern**

Print a symmetrical pyramid of stars.

📥 Input:

Enter number of rows: 5

📤 Output:

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

**12. Hollow Square**

Print a square border with hollow inside.

📥 Input:

Enter size: 5

📤 Output:

\*\*\*\*\*

\* \*

\* \*

\* \*

\*\*\*\*\*

**13. Write a program to print numbers from 1 to 10 using for, while & do while loop**

**14. Write a program to print prime numbers from 1 to N**📥 Input:

Enter N: 10

📤 Output:

2,3,5,7