# Diamond Operator for Anonymous Inner Classes in Java

A COMPARISON OF LEGACY AND MODERN APPROACHES IN JAVA

# Introduction

- The diamond operator (`<>`) was introduced in Java 7 to simplify the use of generics.
- Legacy Approach (Pre-Java 7)
  - - Verbose generic type declarations
  - - Explicit type parameters in anonymous inner  classes
  - - Complex collection initialization

# Legacy Approach - Code Example

```
Map<String, List<Transaction>> transactions =
    new HashMap<String, List<Transaction>>();

Comparator<Transaction> comparator = new Comparator<Transaction>() {
    @Override
    public int compare(Transaction t1, Transaction t2) {
        return t1.getAmount().compareTo(t2.getAmount());
    }
};

Map<String, Set<Payment>> payments = new HashMap<String, Set<Payment>>() {{
    put("USD", new HashSet<Payment>());
    put("EUR", new TreeSet<Payment>());
}};
```

# Modern Approach (Java 7+ and Java 9+)

- - Simplified generic declarations
- - Diamond operator with anonymous inner class (Java 9+)
- - Streamlined collection initialization

# Modern Approach - Code Snippet

```
Map<String, List<Transaction>> transactions = new HashMap<>();

Comparator<Transaction> comparator = new Comparator<>() {
    @Override
    public int compare(Transaction t1, Transaction t2) {
        return t1.getAmount().compareTo(t2.getAmount());
    }
};

Map<String, Set<Payment>> payments = new HashMap<>() {{
    put("USD", new HashSet<>());
    put("EUR", new TreeSet<>());
}};
```

# Benefits of the Diamond Operator

- - Reduces code verbosity
- - Improves readability
- - Avoids redundant type declarations
- - Works well with anonymous inner classes in Java 9+

# Summary

- The diamond operator (`<>`)
  - simplifies the use of generics
  - making Java code cleaner and more readable
  - Since Java 9, it also supports anonymous inner classes, further enhancing code maintainability