

Building Flexible Java Programs

**With Generics, Lists,
and Maps**



**Peerzada Anzar Azmat
Software Engineer, Eurofins**

Welcome! Today's Agenda

- Quick Warm-up: Prerequisites & Core OOP Recap.
- The Problem: Uncover the hidden dangers of "flexible" code.
- The Solutions: Dive into Generics, Lists, & Maps.
- Live Build: Create "DevConnect," a mini social feed, from scratch.
- Q&A: Your questions, my answers.

My aim is to show you how to write code that doesn't just work, but lasts.



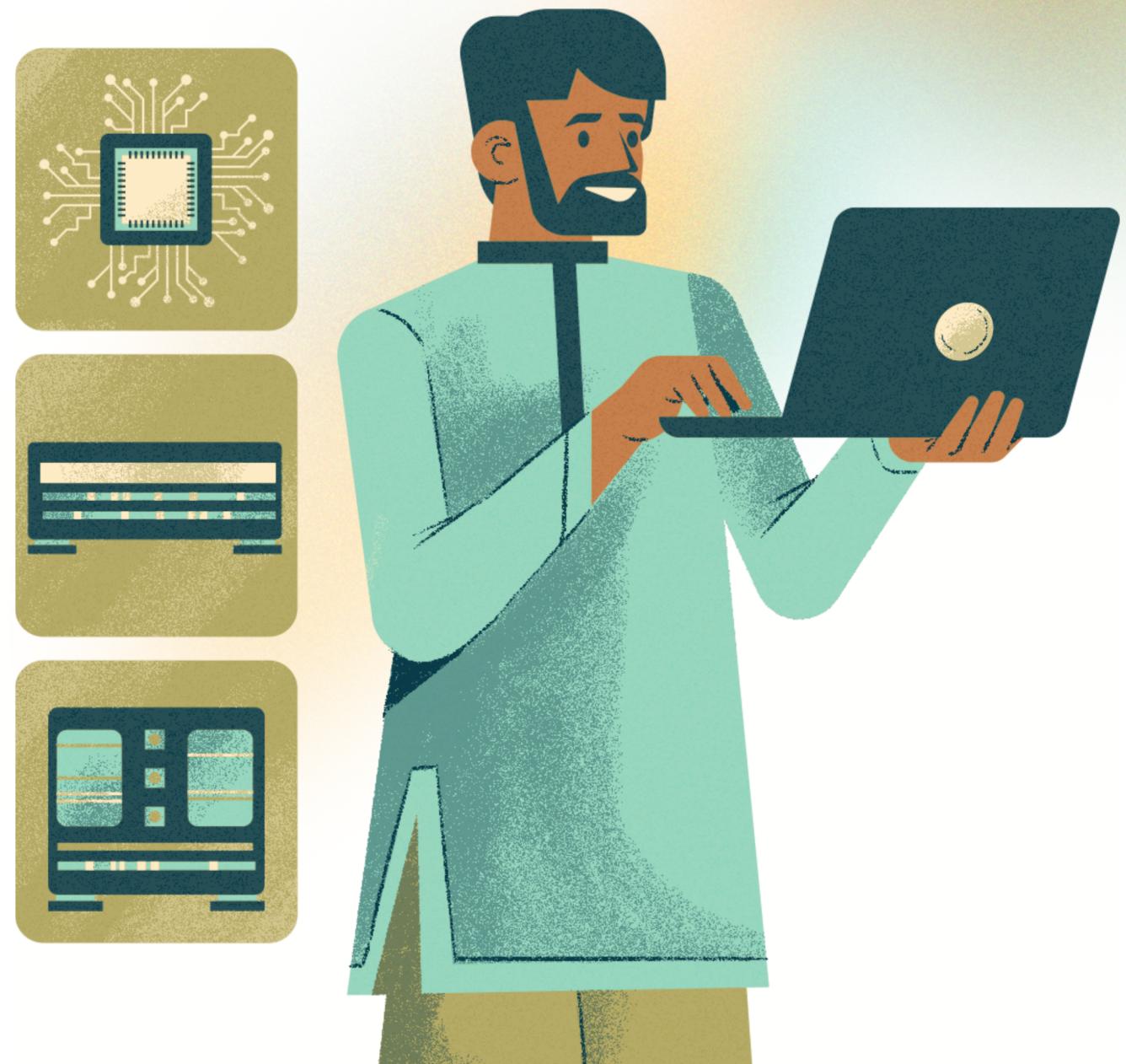
Prerequisites (Warm-up)

Before We Start: Quick Check-in

(Let's confirm our foundations!)

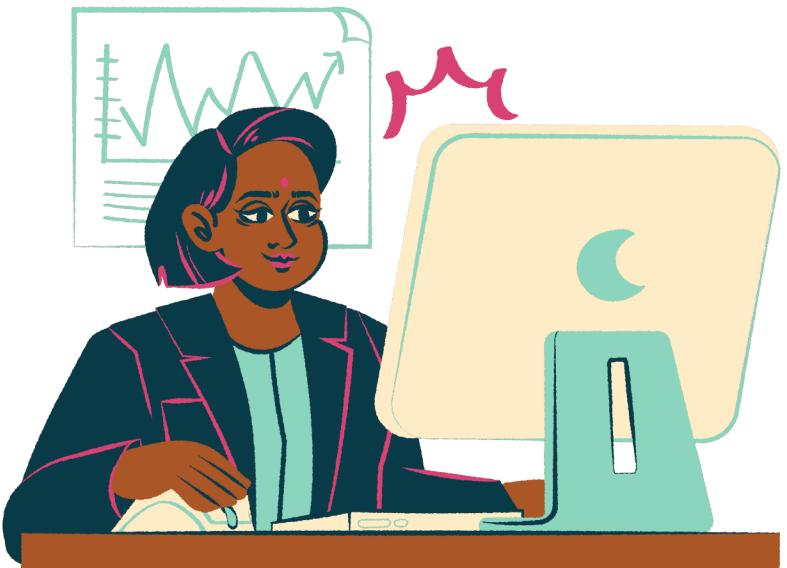
This session will build upon your understanding of:

1. **Basic Java Syntax:** Variables, loops, methods, `main` method.
2. **Object-Oriented Programming (OOP):**
 - **Objects:** `class` as a blueprint, `object` as an instance.
 - **Inheritance:** `extends` keyword, how a `Dog` **is-a** `Animal`.
 - The Object **Class:** The ultimate superclass for **everything** in Java.



Act I: The Problem - The Flexible, But Fragile Bag

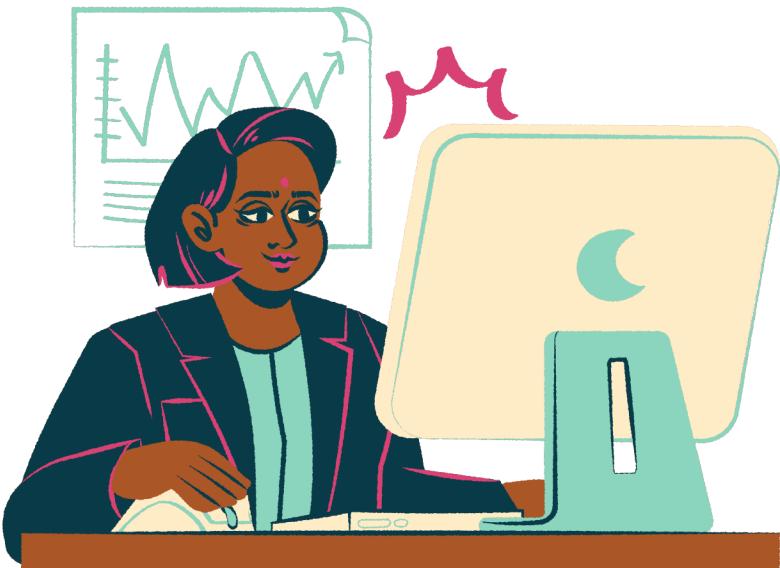
When "Smart" Code Goes Horribly Wrong



We'll see how Java's core flexibility can ironically lead to runtime crashes.

Act II: The Solutions – Our Heroes Arrive

Building Robustness & Efficiency

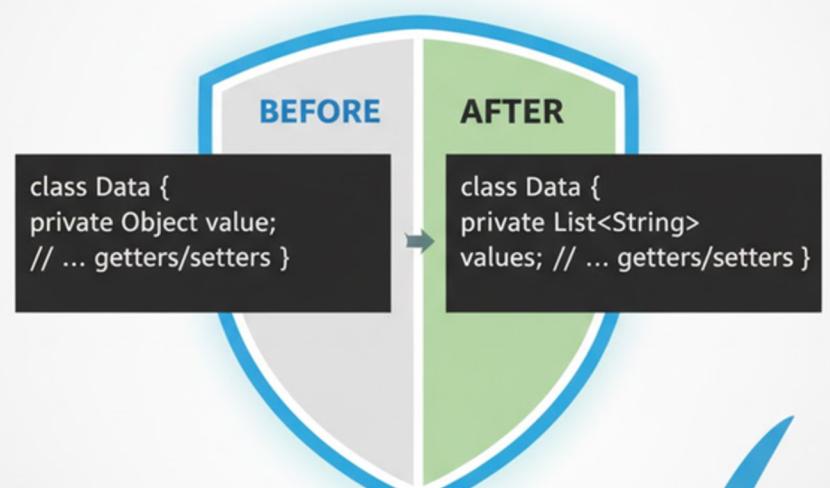


- **Generics:** The [Type Safety Guardian](#)
- **Lists:** The **Ordered Organizer** ([ArrayList](#) vs [LinkedList](#))
- **Maps:** The **High-Speed Directory** ([HashMap](#))

These tools are how professional Java developers prevent crashes and build flexible, performant systems.

Solution 1: Generics <T> Your Compile-Time Guardian

BEFORE & AFTER



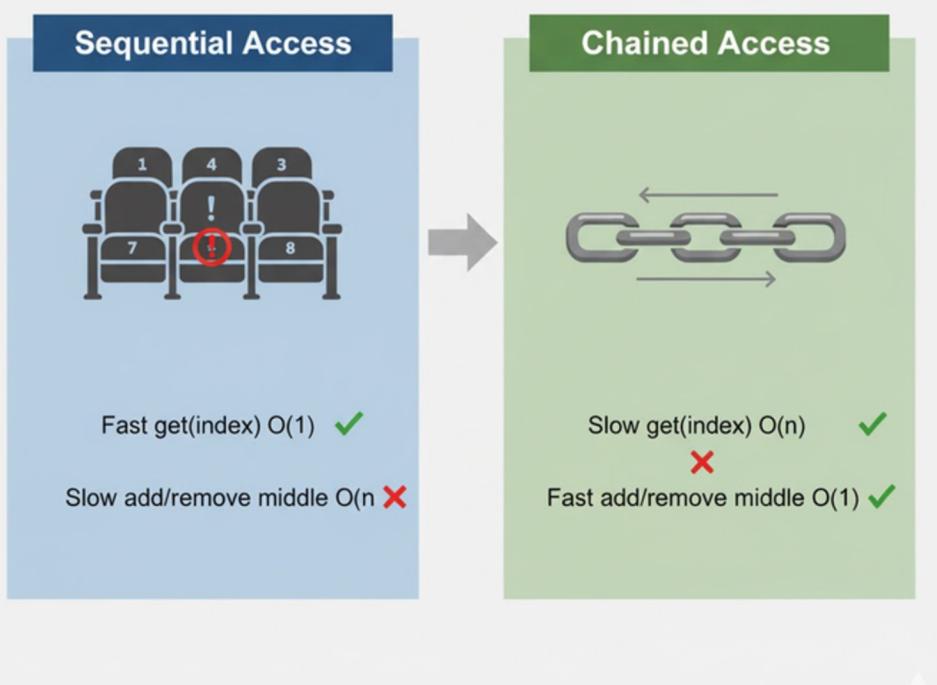
- **What it does:** Allows you to define classes/methods that operate on types as parameters.
- **The Benefit:** Moves errors from **Runtime** (crash!) to **Compile-time** (IDE error!).
- **Example:** `List<String>` means "A list that **only holds Strings**."

Generics enforce '**contracts**' in your code, making it dramatically safer.

Solution 2: Lists

The Ordered Organizer

Data Structure Comparison

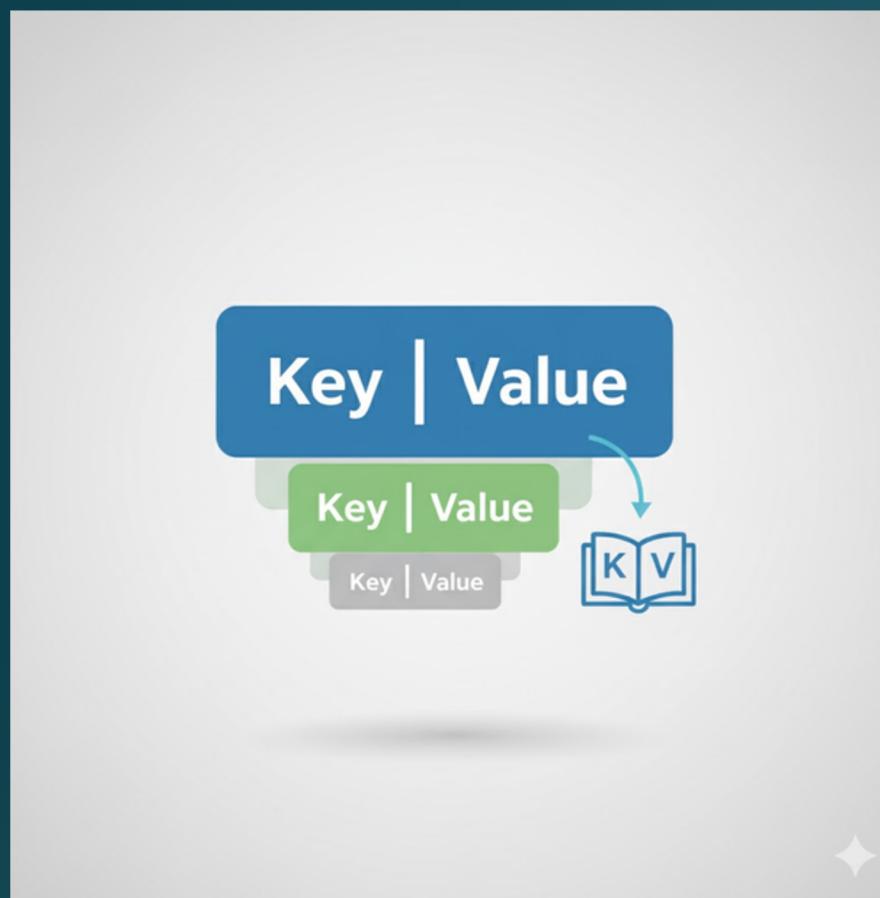


ArrayList vs. LinkedList: A Performance Duel

Choosing the right *List* implementation has a massive impact on your application's speed and efficiency.

Solution 3: Maps

The High-Speed Directory



Key-Value Storage for Lightning-Fast Lookups

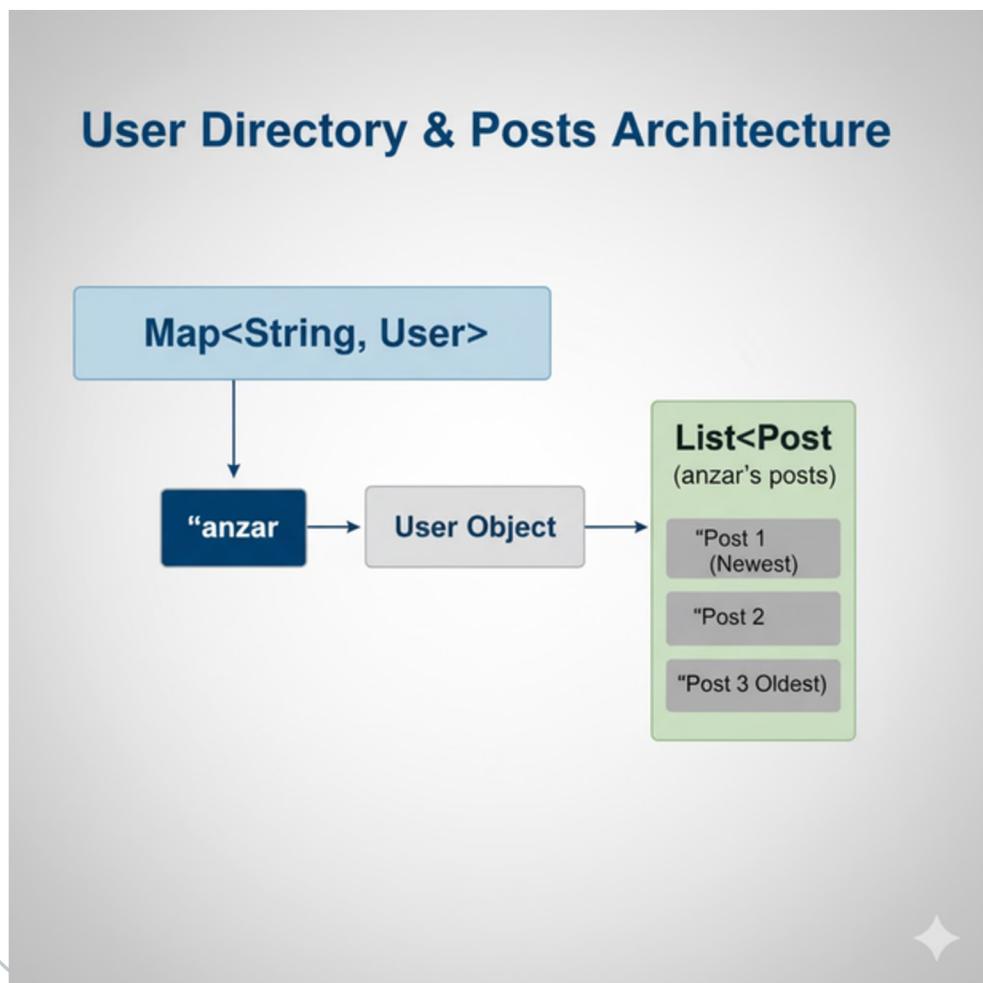
- **What it does:** Stores data as (Key, Value) pairs.
- **Key Property:** Keys must be *unique*.
- **Implementation:** `HashMap<K, V>` is the most common.
- **Performance:** Incredibly fast `get (key)` and `put (key, value)` - $O(1)$ (Constant Time).

When you need to find something fast by a unique identifier, Map is your go-to.

Act III: The Live Build

"DevConnect"

Bringing Generics, Lists, & Maps Together



We'll build a mini social feed where every piece highlights our heroes.

Your New Toolkit (Key Takeaways)

- **Generics:** Your **Safety Net**. Catches errors at compile-time.
- **Lists:** Your **Organizer**. For ordered collections. `ArrayList` (fast reads), `LinkedList` (fast middle-inserts/deletes).
- **Maps:** Your **Directory**. For instant O(1) lookups by key. `HashMap` is your friend.

Writing flexible, robust, and performant Java is not magic – it's a choice you make by mastering these fundamental tools.





QnA Time!

Questions ?

Peerzada Anzar Azmat
Software Engineer, Eurofins
peerzada.anzar@ieee.org