**1) Quick hook (1–2 minutes)**

Ask: “If 200 students are enrolled at COMSATS, where should we store **the number 200**—inside every Student object, or once, somewhere central?”  
Students will usually say “once”. That’s your doorway to **static (class) members** vs **non-static (instance) members**.

**2) Purpose of each feature (why it exists in OOP)**

* **Static (class) fields**  
  A single shared piece of data that belongs to the **class itself**, not any one object. Use it for things like counters, configuration, or constants that are common to all objects (e.g., universityName, totalStudents).
* **Non-static (instance) fields**  
  Data that is unique **per object** (e.g., each student’s name, email, currentCredits). This is the core of modeling real-world instances.
* **Static (class) methods**  
  Behavior that doesn’t need a specific object’s state (no this). Use for validation, utility operations, or operations purely about the class (e.g., isValidComsatsEmail, renameUniversity). They can read/modify static fields, but **cannot** access instance fields directly.
* **Non-static (instance) methods**  
  Behavior that *does* use the specific object’s state (e.g., enrollCourse changes that student’s credits).
* **final class fields**  
  Make a field assignable **only once**.
  + static final → a constant shared by all (e.g., MAX\_CREDITS\_PER\_SEM).
  + instance final → a value fixed for that object after construction (e.g., rollNo).  
    (If it’s a reference type, final means the **reference** can’t change, but the object it points to could still be mutable.)

**3) Real-life, relatable example (runnable Java)**

public class UniversityStudent {  
  
 *// ---------- Static (class) fields ----------* private static String universityName = "COMSATS Lahore";  
 private static int totalStudents = 0;  
  
 *// A constant (same for everyone) — public for teaching convenience* public static final int MAX\_CREDITS\_PER\_SEM = 18;  
  
 *// ---------- Instance (non-static) fields ----------* private final String rollNo; *// fixed for each student after construction* private String name;  
 private String email;  
 private int currentCredits;  
  
 *// ---------- Constructor ----------* public UniversityStudent(String rollNo, String name, String email) {  
 this.rollNo = rollNo;  
 this.name = name;  
 this.email = email;  
 totalStudents++; *// updating shared class state* }  
  
 *// ---------- Instance (non-static) methods ----------* public void enrollCourse(int credits) {  
 if (credits <= 0) throw new IllegalArgumentException("credits must be positive");  
 if (currentCredits + credits > MAX\_CREDITS\_PER\_SEM) {  
 throw new IllegalStateException("Exceeds semester credit limit of " + MAX\_CREDITS\_PER\_SEM);  
 }  
 currentCredits += credits;  
 }  
  
 public void dropCourse(int credits) {  
 if (credits <= 0) throw new IllegalArgumentException("credits must be positive");  
 if (currentCredits - credits < 0) throw new IllegalStateException("Cannot drop below 0 credits");  
 currentCredits -= credits;  
 }  
  
 public String profile() {  
 return String.format("[%s] %s | %s | Credits: %d | %s",  
 rollNo, name, email, currentCredits, universityName);  
 }  
  
 *// ---------- Static (class) methods ----------* public static boolean isValidComsatsEmail(String email) {  
 return email != null && email.toLowerCase().endsWith("@comsats.edu.pk");  
 *// Note: can't do `name` or `currentCredits` here (no `this` in static context)* }  
  
 public static void renameUniversity(String newName) {  
 if (newName == null || newName.isBlank()) throw new IllegalArgumentException("name required");  
 universityName = newName;  
 }  
  
 public static int getTotalStudents() {  
 return totalStudents;  
 }  
  
 *// ---------- Getters/Setters for instance fields (selective) ----------* public String getRollNo() { return rollNo; } *// final → no setter* public String getName() { return name; }  
 public void setName(String name) { this.name = name; }  
 public String getEmail() { return email; }  
 public void setEmail(String email) {  
 if (!*isValidComsatsEmail*(email)) throw new IllegalArgumentException("Use COMSATS email");  
 this.email = email;  
 }  
  
 *// ---------- Tiny demo ----------* public static void main(String[] args) {  
 UniversityStudent s1 = new UniversityStudent("23-CS-001", "Ayesha", "ayesha@comsats.edu.pk");  
 UniversityStudent s2 = new UniversityStudent("23-CS-002", "Bilal", "bilal@comsats.edu.pk");  
  
 s1.enrollCourse(12);  
 s2.enrollCourse(15);  
  
 System.*out*.println(s1.profile());  
 System.*out*.println(s2.profile());  
  
 System.*out*.println("Total students so far (static): " + UniversityStudent.*getTotalStudents*());  
  
 UniversityStudent.*renameUniversity*("COMSATS University Islamabad (Lahore Campus)");  
 System.out.println("After rename:");  
 System.out.println(s1.profile()); *// both show new uni name* System.out.println(s2.profile());  
  
 *// Uncomment to see validation fail:  
 // s2.setEmail("bilal@gmail.com"); // throws: Use COMSATS email* }  
}

**4) More comprehensive term (instead of saying static/non-static)?**

Use **class members** (for static) and **instance members** (for non-static).  
You can also say **static context** vs **instance context**.

**🔹 Exercise: Static vs Instance + Final**

public class REPLDemo {  
 public static void main(String[] args) {  
  
 *// 1. Create two students* UniversityStudent s1 = new UniversityStudent("23-CS-001", "Ayesha", "ayesha@comsats.edu.pk");  
 UniversityStudent s2 = new UniversityStudent("23-CS-002", "Bilal", "bilal@comsats.edu.pk");  
  
 *// 2. Enroll some courses* s1.enrollCourse(9);  
 s2.enrollCourse(12);  
  
 *// --- Exercise A ---* System.*out*.println(s1.profile());  
 System.*out*.println(s2.profile());  
 System.*out*.println("Total students: " + UniversityStudent.getTotalStudents());  
 *// Q: Why is totalStudents = 2 instead of 1 or 0?  
  
 // --- Exercise B ---* UniversityStudent.renameUniversity("COMSATS University Lahore Campus");  
 System.*out*.println("After renaming university...");  
 System.*out*.println(s1.profile());  
 System.*out*.println(s2.profile());  
 *// Q: Why do \*both\* objects reflect the new university name automatically?  
  
 // --- Exercise C ---* System.*out*.println("Max credits allowed: " + UniversityStudent.MAX\_CREDITS\_PER\_SEM);  
 *// Q: What happens if we try UniversityStudent.MAX\_CREDITS\_PER\_SEM = 20?  
  
 // --- Exercise D ---  
 // Uncomment this and run:  
 // s2.setEmail("bilal@gmail.com");  
 // Q: Why does it throw an exception?  
  
 // --- Exercise E ---  
 // Uncomment this and run:  
 // s1.rollNo = "24-CS-009"; // ERROR!  
 // Q: Why does compiler stop us here but not with email or name?  
  
 // --- Challenge ---  
 // Predict output:* UniversityStudent s3 = new UniversityStudent("23-CS-003", "Fatima", "fatima@comsats.edu.pk");  
 System.*out*.println("Total students: " + UniversityStudent.getTotalStudents());  
 }  
}

**🎯 Questions**

* **Exercise A:** Explain why totalStudents increases even though it wasn’t defined in the constructor’s object fields.
* **Exercise B:** Explain why renameUniversity affects **all** objects.
* **Exercise C:** What makes MAX\_CREDITS\_PER\_SEM different from totalStudents?
* **Exercise D:** Why does setEmail("gmail") fail while setName("Ali") works?
* **Exercise E:** Why can’t we reassign rollNo, but we can change email or name?
* **Challenge:** Predict the new value of totalStudents after adding s3.