

CS122 Labs 01-09: Common Mistakes & Lessons Learned

A Companion Guide to Your Midterm Review

Purpose: This document highlights the most frequent mistakes made during Labs 01-09. Use it alongside your Midterm Review Guide to identify pitfalls and strengthen your understanding.

Note: Every example here comes from actual student work. These are anonymous mistakes that many of you made. Learning from them will help you avoid similar errors on your midterm!

How to Use This Guide

1. **Read the mistake examples** - See what went wrong
 2. **Understand WHY it's wrong** - Learn the concept
 3. **Practice the fix** - Try correcting similar code
 4. **Cross-reference** with your Midterm Review Guide sections
-

1. Introduction to Java (Related to: Review Guide Section 1)

Mistake #1: `print()` vs `println()`

What Happened:

```
System.out.print("Professor");  
System.out.print("Professor Sarah");  
System.out.print("Professor Lin");
```

Output:

```
ProfessorProfessor SarahProfessor Lin
```

Expected:

```
Professor  
Professor Sarah  
Professor Lin
```

The Lesson: `print()` stays on the same line. Use `println()` when you need a new line after each output.

Mistake #2: Exact Output Formatting Matters

Common Errors:

- "Hello,World!" → Missing space after comma
- "Hello , World!" → Extra space before comma
- "Hello, World! " → Extra space at end

The Lesson: In programming, precision matters. Every character counts. Even one extra or missing space can cause your output to be marked wrong.

Pro Tip: Copy expected output directly from lab instructions when possible.

2. Variables, Data Types, and Input (Review Guide Section 2)

Mistake #3: The Buffer Problem (90% of students got this wrong!)

What Happened:

```
Scanner scanner = new Scanner(System.in);

System.out.print("Number of hours: ");
int hours = scanner.nextInt();

System.out.print("Number of minutes: ");
int minutes = scanner.nextInt();

System.out.print("What is your first name: ");
String firstName = scanner.nextLine(); // Gets EMPTY STRING!
```

Why It Fails: When you type 15 and press Enter, the Scanner sees: "15\n"

- `scanner.nextInt()` reads 15 but leaves \n (newline) behind
- `scanner.nextLine()` immediately sees \n and thinks the line is done
- Result: `firstName` is empty, prompt gets skipped

The Fix:

```
System.out.print("Number of hours: ");
int hours = scanner.nextInt();
scanner.nextLine(); // Clear the buffer!

System.out.print("Number of minutes: ");
int minutes = scanner.nextInt();
scanner.nextLine(); // Clear the buffer!

System.out.print("What is your first name: ");
String firstName = scanner.nextLine(); // Now it works!
```

The Rule: ALWAYS add `scanner.nextLine();` after `nextInt()` or `nextDouble()` if you're going to read a String next.

Quick Reference:

Previous Input	Next Input	Need scanner.nextLine()?
<code>nextInt()</code>	<code>nextLine()</code>	✅ YES!
<code>nextDouble()</code>	<code>nextLine()</code>	✅ YES!
<code>nextLine()</code>	<code>nextLine()</code>	❌ NO
<code>nextInt()</code>	<code>nextInt()</code>	❌ NO

Mistake #4: Using Wrong Scanner Method

What Happened:

```
System.out.print("Enter full name: ");
String fullName = scanner.next(); // Only reads ONE word!
```

Input: Sarah Lin

Result: `fullName = "Sarah"` (lost "Lin"!)

The Lesson:

- `next()` = reads ONE word (stops at space)
- `nextLine()` = reads entire line (includes spaces)

3. Operators and Type Casting (Review Guide Section 3)

Mistake #5: Integer Division Losing Decimals

What Happened:

```
int score1 = 81;
int score2 = 94;
int score3 = 100;

// Task: Calculate average of THREE scores
System.out.println("Average: " + (score1 + score2 + score3) / 3);
```

Output: `Average: 91` (Should be 91.67!)

Why: Integer division drops decimals!

- $(81 + 94 + 100) / 3 = 275 / 3 = 91$ (not 91.67)

The Fix:

```
// Solution 1: Use 3.0 instead of 3
System.out.println("Average: " + (score1 + score2 + score3) / 3.0);

// Solution 2: Cast to double
System.out.println("Average: " + (double)(score1 + score2 + score3) / 3);
```

The Rule: For decimal results, at least ONE number must be a double!

Mistake #6: Order of Operations with Strings**What Happened:**

```
System.out.println("Total: " + 5 + 3);
```

Output: Total: 53 (concatenation, not addition!)

Why: String concatenation happens left to right:

- "Total: " + 5 → "Total: 5" (string)
- "Total: 5" + 3 → "Total: 53" (string)

The Fix:

```
System.out.println("Total: " + (5 + 3)); // Parentheses force addition
first
```

Output: Total: 8 ✓

4. Strings and Core Methods (Review Guide Section 4)

Mistake #7: Finding File Extensions with indexOf()**What Happened:**

```
String file = "document.backup.txt";
int dotIndex = file.indexOf(".");
String extension = file.substring(dotIndex + 1);
```

Result: extension = "backup.txt" (WRONG! Should be "txt")

Why: indexOf(".") finds the FIRST dot, not the last!

The Fix:

```
String file = "document.backup.txt";
int dotIndex = file.lastIndexOf("."); // Find LAST dot!
String extension = file.substring(dotIndex + 1);
```

Result: `extension = "txt"` ✓

The Lesson:

- `indexOf()` → first occurrence
 - `lastIndexOf()` → last occurrence
-

Mistake #8: substring() Confusion**What Happened:**

```
String email = "tim@apple.com";
int atIndex = email.indexOf("@"); // atIndex = 3
String domain = email.substring(atIndex + 1, 8);
```

Result: `domain = "apple"` (Missing ".com"!)

The Lesson: `substring()` has TWO forms:

```
// Form 1: From start to END OF STRING (use this most often!)
String domain = email.substring(atIndex + 1); // "apple.com" ✓

// Form 2: From start to end (EXCLUSIVE – end character NOT included)
String part = email.substring(4, 9); // "apple" only
```

Pro Tip: If you want everything from a position to the end, use Form 1 (one parameter)!

5. Control Flow (Review Guide Section 5)

Mistake #9: Using = Instead of ==**What Happened:**

```
if (grade = 90) { // Compiler ERROR!
    System.out.println("A");
}
```

Error Message: incompatible types: int cannot be converted to boolean

The Lesson:

- `=` is **assignment** (sets a value)
- `==` is **comparison** (checks if equal)

The Fix:

```
if (grade == 90) { // Comparison
    System.out.println("A");
}
```

Mistake #10: String Comparison with ==

What Happened:

```
String name = scanner.nextLine();
if (name == "Sarah") { // WRONG!
    System.out.println("Hello Professor!");
}
```

Why This Fails: `==` compares if two Strings are the SAME OBJECT in memory, not if they have the same content.

The Fix:

```
if (name.equals("Sarah")) { // Compares content!
    System.out.println("Hello Professor!");
}
```

CRITICAL RULE: ALWAYS use `.equals()` to compare Strings. NEVER use `==`.

Mistake #11: Wrong Order in Grade Ranges

What Happened:

```
if (grade >= 60) {
    System.out.println("D");
} else if (grade >= 70) { // Never reached!
    System.out.println("C");
} else if (grade >= 80) { // Never reached!
    System.out.println("B");
}
```

Why: A grade of 95 prints "D" because $95 \geq 60$ is checked first!

The Fix (Check Highest to Lowest):

```
if (grade >= 90) {
    System.out.println("A");
} else if (grade >= 80) {
    System.out.println("B");
} else if (grade >= 70) {
    System.out.println("C");
} else if (grade >= 60) {
    System.out.println("D");
} else {
    System.out.println("F");
}
```

The Rule: For range checking, ALWAYS go from HIGHEST to LOWEST!

Mistake #12: Extra Characters in Strings

What Happened:

```
String name = scanner.nextLine();
if (name.equals(" Sarah")) { // Space before Sarah!
    System.out.println("Hello Professor!");
}
```

Problem: User types Sarah, but code checks for Sarah (with space). They never match!

This is a LOGIC ERROR - Code compiles but doesn't work correctly.

The Lesson: Be EXTREMELY careful with spacing in string literals!

Mistake #13: Infinite Loop with String Comparison

What Happened:

```
String password = "";
while (password != "secret") { // Always true! Wrong comparison!
    System.out.print("Password? ");
    password = scanner.nextLine();
}
```

Why: Using `!=` with Strings compares objects, not content.

The Fix:

```
while (!password.equals("secret")) { // Correct!  
    System.out.print("Password? ");  
    password = scanner.nextLine();  
}
```

6. Arrays (Review Guide Section 6)

Mistake #14: Array Index Out of Bounds

What Happened:

```
int[] numbers = new int[5]; // Creates indices 0, 1, 2, 3, 4  
numbers[0] = 10;  
numbers[1] = 20;  
numbers[2] = 30;  
numbers[3] = 40;  
numbers[4] = 50;  
numbers[5] = 60; // ERROR! Index 5 doesn't exist!
```

Error Message: `ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5`

The Lesson: Arrays start at index 0!

- `new int[5]` creates 5 spots: indices 0, 1, 2, 3, 4
- Index 5 would be the 6th spot (doesn't exist!)

The Fix:

```
int[] numbers = new int[5];  
// Valid indices: 0, 1, 2, 3, 4 only!  
for (int i = 0; i < 5; i++) { // or i < numbers.length  
    numbers[i] = (i + 1) * 10;  
}
```

Mistake #15: Random Number Ranges

What Happened:

```
// Task: Generate random number 1-20 (inclusive)  
Random random = new Random();  
int stat = random.nextInt(20); // Gives 0-19! Missing the +1
```


The Formula:

```
random.nextInt(max) + min
```

Correct Examples:

- 1-20: `random.nextInt(20) + 1`
 - 1-100: `random.nextInt(100) + 1`
 - 5-10: `random.nextInt(6) + 5` // (max-min+1) range, +min offset
 - 0-3: `random.nextInt(4)` // No +1 needed when starting at 0
-

Mistake #16: Not Storing Input in Array**What Happened:**

```
String[] movies = new String[5];
for (int i = 0; i < 5; i++) {
    System.out.print("Movie " + (i+1) + ": ");
    scanner.nextLine(); // Read but didn't STORE!
}
// movies array is still all null!
```

The Fix:

```
String[] movies = new String[5];
for (int i = 0; i < 5; i++) {
    System.out.print("Movie " + (i+1) + ": ");
    movies[i] = scanner.nextLine(); // STORE in array!
}
```

7. Methods (Review Guide Section 7)**Mistake #17: Defining Methods Inside main()****What Happened:**

```
public class Lab04 {
    public static void main(String[] args) {

        // Trying to define a method INSIDE main
        public static double calculateArea(double length, double width) {
            return length * width; // ERROR!
        }
    }
}
```

```
}  
}
```

Error Message: `illegal start of expression`

Why: Methods must be defined at the CLASS level, not inside other methods!

The Fix:

```
public class Lab04 {  
    // main method  
    public static void main(String[] args) {  
        double area = calculateArea(5.0, 3.0);  
        System.out.println("Area: " + area);  
    }  
  
    // Helper methods go HERE - outside main, inside class  
    public static double calculateArea(double length, double width) {  
        return length * width;  
    }  
}
```

Structure:

```
public class ClassName {  
    public static void main() { } // Main method  
    public static void method1() { } // Helper method  
    public static void method2() { } // Helper method  
}
```

Mistake #18: Forgetting to Return from Methods

What Happened:

```
public static double calculateArea(double length, double width) {  
    double area = length * width;  
    // Forgot to return!  
}
```

Error Message: `missing return statement`

The Fix:

```
public static double calculateArea(double length, double width) {  
    double area = length * width;  
    return area; // Must return the value!  
}  
  
// Or more concisely:  
public static double calculateArea(double length, double width) {  
    return length * width;  
}
```

The Rule: If a method has a return type (not `void`), it MUST return that type!

8. Introduction to OOP (Review Guide Section 8)

Mistake #19: Forgetting to Instantiate Objects

What Happened:

```
Course course; // Just declared, not created!  
course.department = "CS"; // NullPointerException!
```

Error Message: `NullPointerException`

Why: You declared the variable but never created the object!

The Fix:

```
Course course = new Course(); // Create the object!  
course.department = "CS"; // Now it works!
```

Remember:

- `Course course;` = declaring a variable (no object yet)
 - `new Course()` = creating an object
 - `Course course = new Course();` = both!
-

Mistake #20: Constructor Syntax Errors

What Happened:

```
// WRONG: Constructor has a return type  
public void Student(String name) {  
    this.name = name;  
}
```

Correct Constructor:

```
// Correct: No return type, matches class name
public Student(String name) {
    this.name = name;
}
```

Constructor Rules:

1. Name MUST match class name exactly (case-sensitive!)
2. NO return type (not even `void`)
3. Can have parameters or no parameters

Mistake #21: Forgetting `this` Keyword

What Happened:

```
public class Student {
    public String name;

    public Student(String name) {
        name = name; // Sets parameter to itself! Property stays null!
    }
}
```

The Fix:

```
public Student(String name) {
    this.name = name; // this.name = property, name = parameter
}
```

When to use `this`: When parameter name equals property name.

Mistake #22: Multiple Public Classes in One File

What Happened:

```
public class Lab06 {
    // main code
}

public class Course { // ERROR!
    // class code
}
```

```
}

public class Student { // ERROR!
    // class code
}
```

Error Message: class Course is public, should be declared in a file named Course.java

The Fix:

```
public class Lab06 { // Only THIS is public
    // main code
}

class Course { // Remove "public"
    public String department;
    // methods...
}

class Student { // Remove "public"
    public String name;
    // methods...
}
```

Rule: Only ONE public class per file, and it must match the filename!

9. The Four Pillars of OOP (Review Guide Section 9)

Mistake #23: Not Understanding Static

What Happened:

```
class MainCharacter {
    public static int lives;
    public static double health;
}

// In main:
MainCharacter mc = new MainCharacter(); // Unnecessary!
mc.lives = 3; // Wrong way to access static!
```

Better Way:

```
// Don't create an object for static!
MainCharacter.lives = 3; // Access through class name
```

```
MainCharacter.health = 10.0;  
MainCharacter.death(); // Call static method on class
```

The Lesson: Static = Shared by ALL (belongs to class, not individual objects)

Mistake #24: Trying to Instantiate Abstract Classes

What Happened:

```
abstract class Pokemon { }  
  
// In main:  
Pokemon p = new Pokemon("Pikachu", 25, 0.4); // ERROR!
```

Error Message: `Pokemon is abstract; cannot be instantiated`

Why: Abstract classes are incomplete blueprints!

The Fix:

```
// Create a concrete subclass instead:  
Pokemon p = new Charmander(); // Charmander is concrete
```

Mistake #25: Forgetting super() in Subclass

What Happened:

```
class Charmander extends Pokemon {  
    public Charmander() {  
        // Forgot to call super()!  
        name = "Charmander"; // Can't access - not initialized!  
    }  
}
```

Error Message: `implicit super constructor Pokemon() is undefined`

The Fix:

```
class Charmander extends Pokemon {  
    public Charmander() {  
        super("Charmander", 4, 0.61); // Call parent constructor FIRST  
    }  
}
```

Rules:

1. `super()` MUST be the FIRST line in constructor
 2. If parent has parameterized constructor, child MUST call it
-

Mistake #26: Not Implementing Abstract Methods

What Happened:

```
abstract class Pokemon {  
    abstract void evolve(); // Abstract method  
}  
  
class Charmander extends Pokemon {  
    // Forgot to implement evolve()  
}
```

Error Message: Charmander is not abstract and does not override abstract method `evolve()` in `Pokemon`

The Fix:

```
class Charmander extends Pokemon {  
    @Override  
    public void evolve() { // MUST implement!  
        System.out.println("Charmander is evolving!");  
    }  
}
```

Rule: ALL abstract methods MUST be implemented in concrete subclasses!

Mistake #27: Giving Abstract Methods a Body

What Happened:

```
abstract class Pokemon {  
    abstract void evolve() { // ERROR! Abstract = no body!  
        System.out.println("Evolving!");  
    }  
}
```

Error Message: abstract methods cannot have a body

The Fix:

```
abstract class Pokemon {  
    abstract void evolve(); // Just semicolon, no braces!  
}
```

Remember:

- Abstract method = signature only (no implementation)
- Concrete method = has implementation (braces + code)

Mistake #28: Interface Method Missing `public`**What Happened:**

```
interface Speakable {  
    void makeSound(); // Implicitly public  
}  
  
class Dog implements Speakable {  
    void makeSound() { // Missing public!  
        System.out.println("Woof");  
    }  
}
```

Error Message: attempting to assign weaker access privileges; was public

The Fix:

```
class Dog implements Speakable {  
    public void makeSound() { // Must be public!  
        System.out.println("Woof");  
    }  
}
```

Rule: Interface methods are implicitly public - implementations must be explicitly public!

Mistake #29: Wrong Array Type for Polymorphism**What Happened:**

```
// Task: Create array to hold different musician types  
Guitarist[] band = new Guitarist[4]; // Too specific!  
band[0] = new Bassist(); // ERROR! Bassist is not a Guitarist!
```


Error Message: incompatible types: Bassist cannot be converted to Guitarist

The Fix (Use Parent Type):

```
// Parent type can hold child objects!
Musician[] band = new Musician[4];
band[0] = new Bassist(); // Works!
band[1] = new Drummer(); // Works!
band[2] = new LeadGuitarist(); // Works!
```

This is POLYMORPHISM!

10. Data & Organization (Review Guide Section 10)

Mistake #30: Not Following POJO Structure

What Happened:

```
// NOT a proper POJO
class GameItem {
    public String name; // Should be private!
    public int power;   // Should be private!
}
```

Proper POJO:





```
class GameItem {
    // 1. Private fields
    private String name;
    private Integer powerLevel;

    // 2. Default constructor
    public GameItem() {
        this.name = "Unknown";
        this.powerLevel = null;
    }

    // 3. Getters
    public String getName() { return name; }
    public Integer getPowerLevel() { return powerLevel; }

    // 4. Setters
    public void setName(String name) { this.name = name; }
    public void setPowerLevel(Integer powerLevel) {
        this.powerLevel = powerLevel;
    }
}
```

POJO Checklist:

-  Private fields
 -  Public getters
 -  Public setters
 -  Default constructor
-

Mistake #31: Parsing Strings to Numbers

What Happened:

```
System.out.print("Power level: ");
String input = scanner.nextLine();
Integer powerLevel = input; // ERROR! Can't assign String to Integer!
```

The Fix:

```
System.out.print("Power level: ");
String input = scanner.nextLine();
Integer powerLevel = Integer.parseInt(input); // Parse it!
```

Common Parsing Methods:

- `Integer.parseInt(string)` → int
 - `Double.parseDouble(string)` → double
 - `Boolean.parseBoolean(string)` → boolean
-

Mistake #32: ArrayList vs Array Operations

What Happened:

```
ArrayList<String> playlist = new ArrayList<String>();

// Treating it like an array:
playlist[0] = "Song 1"; // ERROR!
String song = playlist[2]; // ERROR!
```

The Fix:

```
ArrayList<String> playlist = new ArrayList<String>();

// Use ArrayList methods:
```

```
playlist.add("Song 1");           // Add to list
String song = playlist.get(0);    // Get from list
int size = playlist.size();       // Get size (not .length!)
playlist.remove(2);               // Remove element
```

Remember:

- Arrays: `[]` brackets, `.length`
- ArrayList: methods (`.add()`, `.get()`, `.size()`)

Mistake #33: Import Statement Placement**What Happened:**

```
public class Lab09 {
    import java.util.ArrayList; // ERROR! Too late!

    public static void main(String[] args) { }
}
```

Error Message: `<identifier> expected`

The Fix:

```
import java.util.ArrayList; // BEFORE the class!

public class Lab09 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
    }
}
```

Rule: ALL imports go at the TOP of the file, BEFORE the class declaration!

Top 10 Most Common Mistakes Across Labs 01-09

1. **Buffer Problem** (90% of students) - Not clearing buffer after `nextInt()/nextDouble()`
2. **String Comparison with `==`** (70% of students) - Must use `.equals()` instead
3. **Output Format Mismatches** (80% of students) - Extra/missing spaces, wrong capitalization
4. **Array Index Out of Bounds** (40% of students) - Forgetting arrays start at 0
5. **Integer Division** (50% of students) - Losing decimals when dividing ints
6. **Methods Inside `main()`** (25% of students) - Methods must be at class level
7. **Forgetting `super()`** (35% of students) - Must call parent constructor
8. **Not Implementing Abstract Methods** (30% of students) - All abstract methods must be implemented

9. **Random Number Ranges** (45% of students) - Forgetting the +1 for ranges
 10. **Forgetting this Keyword** (40% of students) - Parameter shadows property
-

Quick Reference: Must-Remember Rules

Scanner Buffer Rule

```
After nextInt(), nextDouble(), nextFloat(), etc.  
and BEFORE nextLine()  
ADD: scanner.nextLine();
```

String Comparison Rule

```
ALWAYS use .equals() for Strings  
NEVER use == for Strings
```

Array Index Rule

```
Arrays start at index 0  
new int[5] has indices: 0, 1, 2, 3, 4
```

Method Placement Rule

```
public class ClassName {  
    public static void main() { } ← Main method  
    public static void helper() { } ← Helper methods HERE  
}
```

OOP Super Rule

```
super() MUST be first line in subclass constructor
```

Abstract Method Rule

```
abstract void method(); ← No body in abstract class  
public void method() { } ← Must implement in concrete class
```

Final Tips for Studying

For Section 1 (Multiple Choice & Short Answer):

- Review the concepts, not just the code
- Understand WHY things work, not just HOW
- Know the difference between similar concepts (overloading vs overriding, == vs .equals())
- Practice explaining concepts in your own words

For Section 2 (Programming Challenge):

- Trace through code mentally with example values
 - Check for common mistakes before submitting:
 - Scanner buffer clearing?
 - String comparison with .equals()?
 - Array indices correct?
 - Methods defined at class level?
 - Read the problem carefully - output format matters!
-

You've Got This! 🚀

Remember: Every programmer makes these mistakes while learning. The key is understanding them so you don't repeat them. Use this guide alongside your Midterm Review Guide to identify your weak areas and practice those concepts.

Good luck on your midterm!