Lambda Expression: Problem Sets

1. Filename sorter with policy tiers
    a. You've a list of file names, for example: ["README.md", "Main.java", "notes.txt", "index.md", "build.gradle", "App.java", "todo.md"]
    b. Sort them using a single lambda-based Comparator with this policy:
        i. Priority by extension: .md first, then .java, then everything else
        ii. Within the same extension, shorter names first
        iii. If still tied, reverse alphabetical
2. Loop-capture bug hunt: threads and indices
    a. Start three background threads in a for-loop; each prints its own index exactly once:
    b. Thread-0 prints "0", Thread-1 prints "1", Thread-2 prints "2".
3. Score combiner via custom functional interface
    a. Define a @FunctionalInterface named IntCombiner with int apply(int a, int b).
    b. You have two partial scores for a player: base and bonus.
    c. Implement a method calc(int base, int bonus, IntCombiner rule) and call it with three distinct rules:
        i. sum rule: base + bonus
        ii. cap rule: minimum of the two - 100 or base + 2*bonus
        iii. penalty rule: if bonus < 0, return base + bonus*3; else base + bonus
    d. Call calc method with sample bonus & base values and an instance of IntCombiner
4. Comparator chaining
    a. Given a record: record Task(String name, int priority, int durationMins) {}
    b. Sort a List<Task> by
        i. priority ascending
        ii. then durationMins descending
        iii. then name case-insensitive ascending
    c. Do this with method references or lambdas via Comparator.comparing(...) chaining. Show one call site that performs the sort
5. Username policy with a denylist and mixed rules
    a. Given:
        i. A Set deny = {"root","admin","system"}
        ii. A List candidates = ["root","jo","jo_1","Asha99","bob","bob_","SYSTEM","jo1"]
    b. Write three boolean lambdas (you may use Predicate if you want) that check:
        i. length at least 3
        ii. contains only letters, digits, underscore
        iii. not in denylist, case-insensitive
6. Leaderboard updates without TreeMap
    a. Given an initial Map<String,Integer> score and these updates in order:
    b. +10 "ann", +5 "bob", +2 "ann", +7 "ann", +6 "bob", +3 "cara"
    c. Use Map.merge with a lambda to apply updates. Then print the scoreboard as:

        i. 1. name points
        ii. 2. name points
        iii. 3. name points
    d. ordered by points descending, ties broken by name ascending.

7. Retry helper for checked exceptions with lambdas
    a. Design a tiny retry utility for IO-like actions using a lambda that can throw a checked exception.
    b. Create @FunctionalInterface CheckedAction { void run() throws IOException; }
    c. Implement runWithRetry(CheckedAction action, int attempts) that:
        i. tries to run the action up to attempts times
        ii. if it succeeds, return normally
        iii. if it fails every time, throw a RuntimeException wrapping the last IOException
    d. Show how a caller would use it to “write a file” lambda that sometimes throws
8. Statefulness trap
    a. You want to increment an external counter each time a Consumer<String> is called:
        i. int hits = 0;
        ii. Consumer<String> log = s -> { hits++; System.out.println(s); };
    b. Explain why this is a problem with lambdas in Java.
    c. Make it compile, and then rewrite it to avoid external mutation while preserving the behavior contract. Show the safer alternative