Assignment 04: Bug Hunt

CS 140 with Dr. Sam Schwartz

Due: Sunday, October 5 at 11:59pm via Canvas Upload

1 Purpose

The purpose of this assignment is to:

1. Build comfort reading unfamiliar code and distinguishing syntax vs. logic errors.
2. Practice input validation using `Scanner` and `try/catch`.
3. Apply conditionals and Boolean expressions to produce correct, well-explained behavior.

2 Tasks – High Level Overview

1. Copy the starter code below into a file named `BugHunt.java`.
2. Get it to compile by fixing syntax errors.
3. Correct the behavior by fixing logic errors through meeting the specifications described below.
4. Add input validation using `try/catch`.
5. Add a short reflection comment at the top of the file describing the bugs you found and how you fixed them.
6. Run test cases and collect screenshots showing correct behavior.
7. Submit your `.java` file and screenshots to Canvas.

3 Tasks – Detailed Requirements

3.1 Starter Code (copy into `BugHunt.java`)

This code is intentionally buggy. It includes both *syntax* and *logic* errors drawn from topics covered so far (variables, data types, I/O with `Scanner`, conditionals, Boolean expressions, debugging, and try/catch). Your job is to repair it and meet the specification in the next subsection.

```
// This code intentionally contains syntax and logic errors.
// Your job: make it compile, behave correctly, and add try/catch input validation.
// Then add a top-of-file comment reflecting on bugs you found and how you fixed them.
import java.util.Scanner
public class BugHunt {
 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in)
  System.out.print("Choose mode [grade/temp]: ");
  String mode = sc.nextLine();

  if (mode == "grade") {
   runGrade(sc);
  } else if (mode == "temp") {
   runTemp(sc)
  } else {
   System.out.println("Unrecognized mode: " + mode);
  }

  sc.close();
 }
 static void runGrade(Scanner sc) {
  System.out.print("Enter numeric score 0-100: ");
  int score = sc.nextDouble();
  if (score > 100 || score < 0) {
   System.out.println("OK");
  }
  if (score >= 60) {
   System.out.println("Letter grade: D");
  } else if (score >= 70) {
   System.out.println("Letter grade: C");
  } else if (score >= 80) {
   System.out.println("Letter grade: B");
  } else if (score >= 90) {
   System.out.println("Letter grade: A");
  } else {
   System.out.println("Letter grade: F");
  }
 }
 static void runTemp(Scanner sc) {
  System.out.print("Enter Fahrenheit temperature: ");
  double f = sc.nextInt();
  double c = (f - 32) * 1.8;
  System.out.println("Celsius: " + String.format("%.2f", c));

  if (f < 32 && f > 212) {
   System.out.println("Phase: liquid");
  } else if (f <= 32) {
   System.out.println("Phase: gas");
  } else {
   System.out.println("Phase: solid");
  }
 }
}
```

3.2 Specification you must satisfy

Mode selection. The program prompts for a mode `[grade/temp]` and should accept either spelling regardless of case and with stray whitespace handled (e.g., `" Grade"`, `"TEMP"`, etc.). Unrecognized modes must be rejected gracefully with a clear message (no crash).

Grade mode (`grade`). Prompt for an integer score in the closed interval $[0, 100]$ and print the letter grade with these standard boundaries:

A: $[90, 100]$, B: $[80, 90)$, C: $[70, 80)$, D: $[60, 70)$, F: $[0, 60)$.

Out-of-range values (e.g., -4 or 123) must be handled gracefully (clear error message). Use a clear, non-overlapping `if / else if / else` chain. Include at least one compound Boolean expression (e.g., `&&` or `||`) somewhere in your program.

Temperature mode (`temp`). Prompt for a real number (double) in Fahrenheit, compute Celsius with

$$\texttt{celsius} = \frac{\texttt{fahrenheit} - 32}{1.8},$$

print Celsius to two decimal places, *and* classify the phase of water at that Fahrenheit temperature as:

- `solid` for $\leq 32°$F,
- `liquid` for $> 32°$F and $< 212°$F,
- `gas` for $\geq 212°$F.

Input validation with `try/catch`. For both modes, add `try/catch` blocks to handle non-numeric input gracefully (e.g., user enters `abc` when a number is expected). You may use either parsing (e.g., `Double.parseDouble`). Re-prompting is not required; a clear error message is sufficient.

Decomposition and style. Keep the class name `BugHunt`. Use at least three methods (e.g., `main`, `runGrade`, `runTemp`; adding additional helper methods is encouraged). Use descriptive identifiers, consistent brace/indent style, and professional output phrasing.

Reflection (comment at top of file). At the very top of `BugHunt.java`, add a multi-line comment containing:

- Your name,
- A brief reflection identifying *at least three* distinct bugs you found (e.g., syntax, logic, input/exception),
- A one-sentence note on how you fixed each of those bugs.

3.3 Test cases to demonstrate (for screenshots)

Grade mode: Show at least three runs, including boundary cases such as `59` → F, `60` → D, `90` → A, `100` → A.
Temp mode: Show at least three runs that clearly hit `solid` (e.g., `20`), `liquid` (e.g., `70`), and `gas` (e.g., `220`).

3.4 What to Upload

- Your finalized `BugHunt.java` file (compiles and runs).
- Screenshots demonstrating the required test cases for both modes (you may combine multiple runs into one screenshot per mode).

4 Grading Criteria

Scale

Elements of nearly all assignments in this class will be broken down into a 0-1-2 scale. “2” means, “Nailed it!” “1” means, “Umm, kinda got it, but not quite.” “0” means, “Didn’t get it.” The points are averaged out and multiplied by 100. The ceiling becomes the final score. (If you get a 92.01%, we “round up” to 93%.)

Rubric

Validity

The author submitted a `BugHunt.java` file to Canvas	2	1	0.
... which included a top-of-file reflection comment with name and bug summary	2	1	0.
... compiles and runs without crashing for valid inputs	2	1	0.
... trims and handles mode input case-insensitively (`grade`/`temp`)	2	1	0.
... rejects unrecognized mode gracefully (clear message)	2	1	0.
... uses at least three methods (decomposition beyond `main`)	2	1	0.
... replaces == with appropriate string comparison (e.g., `equalsIgnoreCase`)	2	1	0.
... fixes obvious syntax issues (imports/semicolons/braces)	2	1	0.
... adds `try/catch` to handle non-numeric input in grade mode	2	1	0.
... adds `try/catch` to handle non-numeric input in temp mode	2	1	0.
... validates score is in $[0, 100]$ and rejects out-of-range with a clear message	2	1	0.
... implements correct letter-grade mapping with correct boundaries	2	1	0.
... uses a clear, non-overlapping `if/else-if/else` chain for grades	2	1	0.
... reads Fahrenheit as `double` and uses the correct conversion $c = (f - 32)/1.8$	2	1	0.
... prints Celsius to two decimal places	2	1	0.
... classifies water phase correctly with inclusive boundaries at $32°$F and $212°$F	2	1	0.
... demonstrates at least one compound Boolean (`&&` or `||`)	2	1	0.
... avoids the leftover-newline pitfall when mixing `nextInt/nextDouble` and `nextLine`	2	1	0.

Screenshot

The author included a screenshot showing multiple grade-mode runs (including boundaries)	2	1	0.
... and a screenshot showing multiple temp-mode runs (solid, liquid, gas)	2	1	0.

Readability

The reflection identifies at least three distinct bug types and fixes	2	1	0.
... identifiers and formatting are consistent and professional	2	1	0.
... output messages are clear, labeled, and consistently formatted (units, punctuation)	2	1	0.
... and contained a sufficient number of comments explaining and documenting the code, each of which was written in an appropriate register of professional English	2	1	0.

Fluency

The deliverable would not feel “weird-in-a-bad-way” to an experienced practitioner	2	1	0.

Subtotal (of 50) ....

Total ($\lceil 100 \cdot \frac{\textbf{subtotal}}{50} \rceil$) ....