```
import tkinter as tk
from tkinter import ttk, messagebox
import tkinter.font as font
import random


def set_dpi_awareness():
    """
    Set DPI awareness for improved resolution on high-DPI displays
(Windows-specific)
    """
    try:
        from ctypes import windll
        windll.shcore.SetProcessDpiAwareness(1)
    except:
        pass

# Apply DPI awareness
set_dpi_awareness()

# Main application class
class LinearEquationTest(tk.Tk): # Application Window which inherits
from tk.Tk
    def __init__(self, *args, **kwargs):
        # Initialise the tkinter application
        super().__init__(*args, **kwargs) # Makes self essentially equal
to the tk object

        # Set the title of the application window
        self.title("Linear Equation Test")

        # Create and display the Test frame
        frame = Test(self, padding=(100, 60))
        frame.grid()

        # Bind the Enter key to start and check answers
        self.bind("<Return>", frame.start_test)
        self.bind("<Return>", frame.check_answer)

# Frame class containing test logic and tkinter widgets
class Test(ttk.Frame):
    def __init__(self, container, **kwargs):
        # Initialise the frame within the container
        super().__init__(container, **kwargs)

        # Declare variables for test state and GUI updates

        self.equation = tk.StringVar() # Current equation to display
        self.answer_entry = tk.StringVar() # User's answer input
        self.solution = tk.StringVar() # Correct solution for the
current eqaution
        self.progress_display = tk.StringVar() # Displays question
progress
        self.current_question = None # Placeholder for current question
```

```
logic
        self.correct_count = 0 # Counter for correct answers
        self.total_questions = 0 # Counter for total questions answered
        self.feedback_display = tk.StringVar() # Feedback massage for
user
        self.score_display = tk.StringVar() # Dsiplays score percentage
        self.max_questions = tk.IntVar(value=5) # Maximum number of
questions

        # Create and configure widgets for the test

        test_label = ttk.Label(self, text="Linear Equation Test", font=
('Helvetica', 18, 'bold'))
        start_button = ttk.Button(self, text="Start Test",
command=self.start_test)
        equation_label = ttk.Label(self, text="Equation", font=
("Helvetica", 15)) # command=generate_equation
        equation_display = ttk.Label(self, textvariable=self.equation,
font=("Helvetica", 15))
        user_answer_label = ttk.Label(self, text="Answer to two decimal
places: ", font=("Helvetica", 15))
        user_answer_var = ttk.Entry(self, width=10,
textvariable=self.answer_entry, font=("Helvetica", 15))
        self.calc_button = ttk.Button(self, text="Calculate",
command=self.check_answer)
        feedback_label = ttk.Label(self, text="Solution: ", font=
("Helvetica", 15))
        feedback_display =ttk.Label(self,
textvariable=self.feedback_display, font=("Helvetica", 15))
        progress_label = ttk.Label(self, text="Question: ", font=
("Helvetica", 15))
        progress_display = ttk.Label(self,
textvariable=self.progress_display, font=("Helvetica", 15))
        score_label = ttk.Label(self, text="Score: ", font=("Helvetica",
15, 'bold'))
        score_display = ttk.Label(self, textvariable=self.score_display,
font=("Helvetica", 15))

        # Define grid layout for widgets

        test_label.grid(column=0, row=0, sticky="EW")

        start_button.grid(column=0, row=1, sticky="EW")

        equation_label.grid(column=0, row=2, sticky="EW")
        equation_display.grid(column=1, row=2, sticky="EW")

        user_answer_label.grid(column=0, row=3, sticky="EW")
        user_answer_var.grid(column=1, row=3, sticky="EW")
        self.calc_button.grid(column=2, row=3, sticky="EW") # Calculate
button

        user_answer_var.focus()  # Set focus to answer entry field

        feedback_label.grid(column=0, row=4, sticky="EW")
```

```
        feedback_display.grid(column=1, row=4, sticky="EW")

        progress_label.grid(column=0, row=5, sticky="EW")
        progress_display.grid(column=1, row=5, sticky="EW")

        score_label.grid(column=0, row=6, sticky="EW")
        score_display.grid(column=1, row=6, sticky="EW")

        # Apply consistent padding to all child widgets
        for child in self.winfo_children():
            child.grid_configure(padx=15, pady=15)

    def start_test(self):
        """
        Reset test variables and start a new test.

        Parameters: Self
        """
        self.correct_count = 0 # Reset correct answer score
        self.total_questions = 0 # Reset total question counter
        self.max_questions_value = self.max_questions.get() # Get
maximum question limit
        self.calc_button['state'] = 'normal' # Enable the Calculate
button
        self.score_display.set("") # Clear score display
        self.generate_equation()    # Generate the first equation

    def generate_equation_data(self):
        """
        Generate the equation and solution based on random parameters.

        Parameters: self
        """
        coefficient = random.randint(2, 5)
        constant = random.randint(1, 9)
        result = random.randint(1, 25)
        equation_type = random.choice(["basic", "add", "sub"])

        # Generate simple linear equation in different forms and their
solutions
        if equation_type == "basic":
            result = coefficient * random.randint(1, 25)
            equation = f"{coefficient}x = {result}"
            solution = result / coefficient
        elif equation_type == "add":
            equation = f"{coefficient}x + {constant} = {result}"
            solution = (result - constant) / coefficient
        else:  # equation_type == "sub"
            equation = f"{coefficient}x - {constant} = {result}"
            solution = (result + constant) / coefficient

        return equation, solution

    def generate_equation(self):
        """
```

```
        Generate a new equation and update the GUI.

        Parameters: self
        """
        equation, solution = self.generate_equation_data() # Get
equation and solution
        self.equation.set(equation) # Display the equation
        self.solution.set(solution) # Store the correct solution
        self.total_questions += 1 # Increment question counter
        self.progress_display.set(f"
{self.total_questions}/{self.max_questions.get()}")
        self.answer_entry.set("") # Clear the answer entry
        self.feedback_display.set("") # Clear feedback

    def check_answer(self, *args):
        """
        Check if the user's answer is correct.

        Parameters: self, *args
        """
        try:
            user_answer = float(self.answer_entry.get()) # Get user's
input as a float
            solution = float(self.solution.get()) # Get the correct
solution

            if round(user_answer, 2) == round(solution, 2):  # Use
rounding for precision
                self.feedback_display.set("Correct! 👍🙌👍")
                self.correct_count += 1 # Increment correct answer
counter

            else:
                self.feedback_display.set(f"The answer is
{round(solution, 2)}\nBetter luck next time! 🍀🍀🍀")

            # Update score display
            self.score_display.set(f"{int((self.correct_count /
self.total_questions) * 100)}%")

            if self.total_questions == self.max_questions_value: # Check
if the test is complete
                self.finish_quiz()  # Finish the test
                self.calc_button.state(['disabled'])  # Disable
Calculate button

            else:
                self.after(3000, self.generate_equation)  # Generate
next question after delay

        except ValueError: # Handle invalid input
            messagebox.showerror("❌❌❌Error❌❌❌", "Please enter a
valid number")
```

```
    def finish_quiz(self):
        """
        Display final score.

        Parameters: self
        """
        final_score = int((self.correct_count/self.total_questions)*100)
        self.answer_entry.set("")
        self.feedback_display.set("")
        self.progress_display.set("")
        self.equation.set("")
        messagebox.showinfo("TEST OVER",
                            f"👏👏👏👏Test complete👏👏👏👏\n\nYou
scored {self.correct_count} out of {self.total_questions}
correct.\nFinal score: {final_score}%")
        self.score_display.set("") # Clear score display


if __name__ == "__main__":
    # Create an instance of the application
    root = LinearEquationTest()

    # Set default font configuration
    font.nametofont("TkDefaultFont").configure(size=15)

    # Column configuration for the root
    root.columnconfigure(0, weight=1)

    # Run the main application loop
    root.mainloop()
```