Department Mathematik/Informatik, Abteilung Informatik
Software & System Engineering
Weiterführende Konzepte der Programmierung, SS 2025

Niklas Bockholt, Chiara Hafner, Alicja Bezpalko,
Jiyan Alis, Dilara Günay,
Simon Wolf, Adrian Bajraktari, Dr. Mersedeh Sadeghi

Homework 7. Recap

Registration Deadline: 25.05.2025, 23:59
Hand-in Deadline: 28.05.2025, 23:59

In this exercise sheet, you will implement a simple Hangman game. Hangman is a classic word-guessing game where one player thinks of a word, and the other tries to guess it letter by letter. Each incorrect guess brings the player closer to losing the game.

In Task 2, you will enhance the game by adding a timer, and in Task 3, you will improve the structure of your code.

In the first in-class task, you will write a method that selects a random word from a text file. You may optionally integrate this method into your project.

Exercise 1. Hangman Game

To improve usability, we will provide a graphical user interface (GUI) for the game. A GUI is a user interface that includes graphical elements—such as buttons, text fields, and drop-down menus—making the program more accessible and user-friendly than a purely text-based interface. The GUI code is already provided and should not be modified unless explicitly stated.

Create a package `hangman` and a class Game. Inside:

Create `private boolean hasWon()`:

- Return true if every letter of the String `currentWord` has already been guessed, else return false. The guessed letters are supposed to be stored in the ArrayList `guessedLetters`.

Create `protected void endGame(boolean won)`:

- Check whether won is true or false and print an appropriate message to the console that also includes the target word.
- Disable further user input (using `inputField.setEnabled(false)`);
- End the whole program.

Create `protected void handleGuess(char guess)`:

- Check if the letter has already been guessed (using `guessedLetters`). If so, print a hint for the user and exit early.
- Otherwise, add the letter to the set of guessed letters.
- If the letter is not part of the target word `currentWord`, reduce the number of remaining attempts `attemptsLeft`.

- Update the current display of the word (hint: Use `updateDisplay()`. There are no modifications needed.)
- End the game if the player has either guessed the word completely or used up all attempts. Use `hasWon()` and `endGame(boolean won)` to check and finish the game.

Your task is now to add an action listener to the input field (a text box) so that the program reacts when the player presses the Enter key. Specifically create `public void processInput()`:

- Retrieve the text from the input field `inputField` and convert it to lowercase.
- Clear the input field after reading the input.
- Check that the input is exactly one character and that it is a letter.
  - If the input is invalid, print the message `"Please enter exactly one letter."` to the console and return without doing anything else.
- If the input is valid, call the `handleGuess()` method with the entered character.

At this point, you should have a fully working version of the Hangman game where the player can guess a fixed word using keyboard input through the GUI. The input is validated, incorrect guesses reduce the number of attempts, and the game ends automatically when the word is guessed correctly or the attempts run out. In the next tasks, we will further extend the functionality by adding a countdown timer to increase the challenge, improving the code structure, and optionally in the in-class tasks, enabling the possibility of picking a random word to guess.

Use the following code as a starting point:

```
package hangman;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public class Game extends JFrame {
    // GUI components
    protected JLabel wordLabel, attemptsLabel, timeLabel;
    protected JTextField inputField;

    // The word the player has to guess (fixed for now)
    protected String currentWord = "example";

    // Stores the letters guessed by the player
    protected ArrayList<Character> guessedLetters = new ArrayList<>();

    // Number of tries left
    protected int attemptsLeft = 6;

    // Time left (not yet functional - stays at 60)
    protected int timeLeft = 60;

    // Sets up the window and its components
    public Game() {
        setTitle("Hangman Game");
```

```
        setSize(400, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new GridLayout(6, 1));

        // Button to start a new game
        JButton startButton = new JButton("Start Game");
        add(startButton);

        // Label showing the guessed word with _ for unknown letters
        wordLabel = new JLabel("Word: ", SwingConstants.CENTER);
        add(wordLabel);

        // Label showing how many tries are left
        attemptsLabel = new JLabel("Attempts left: " + attemptsLeft,
        SwingConstants.CENTER);
        add(attemptsLabel);

        // Label showing the time left (not yet changing)
        timeLabel = new JLabel("Time left: " + timeLeft,
        SwingConstants.CENTER);
        add(timeLabel);

        // Input field where the player types their guess
        inputField = new JTextField();
        inputField.setHorizontalAlignment(JTextField.CENTER);
        inputField.setEnabled(false); // initially inactive
        add(inputField);

        // Starts the game when button is clicked and disables restart
        startButton.addActionListener(e -> {
            startGame();
            startButton.setEnabled(false);
        });

        // You will implement this method
        processInput();

        setVisible(true);
    }

    // Resets game state when starting a new game
    protected void startGame() {
        guessedLetters.clear();
        attemptsLeft = 6;
        timeLeft = 60;
        updateDisplay();
        inputField.setEnabled(true);
        inputField.requestFocus();
    }
```

```
    // Updates the word, attempts, and time on the screen
    protected void updateDisplay() {
        StringBuilder display = new StringBuilder();
        for (char c : currentWord.toCharArray()) {
            if (guessedLetters.contains(c)) {
                display.append(c).append(" ");
            } else {
                display.append("_ ");
            }
        }
        wordLabel.setText("Word: " + display.toString());
        attemptsLabel.setText("Attempts left: " + attemptsLeft);
        timeLabel.setText("Time left: " + timeLeft);
    }

    /* TODO Task 1: Add ActionListener for keyboard input on
    inputField and process the input */
    public void processInput() {
        // You will implement this
    }

    /* TODO Task 1: Check if the guessed letter is in the
    word and handle result */
    protected void handleGuess(char guess) {
        // You will implement this
    }

    /* TODO Task 1: Check if the player has guessed
    all letters correctly */
    private boolean hasWon() {
        // You will implement this
        return false;
    }

    // TODO Task 1: End the game and show a message
    protected void endGame(boolean won) {
        // You will implement this
    }
}
```

```
package hangman;
import javax.swing.SwingUtilities;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new Game();
        });
```

```
    }
}
```

Note: The words in the file are well-defined and unique except for differences in capitalization. When processing the words, you do not need to worry about case sensitivity—treat uppercase and lowercase letters as equivalent.

Exercise 2. Timer

In this task, you will implement a countdown timer that starts when the game begins. The timer should decrease the time by one second every second. If the timer reaches zero, the game should end automatically with a corresponding message.

- Implement the `Runnable` interface: Create a private class (e.g., `TimerRunnable`) that implements the `Runnable` interface. Inside the `run()` method, write a loop that decrements the time left every second until it reaches zero or the timer is stopped.
- `Start the timer thread:` In the `startTimer()` method, instantiate a new `Thread` with the `TimerRunnable` and start it.
- Update the GUI safely: Use `SwingUtilities.invokeLater()` to update the timer display on the Swing event dispatch thread.
- Handle timer end: When the timer reaches zero, stop the timer and call `endGame(false)` via `SwingUtilities.invokeLater()`.
- Thread interruption: Handle `InterruptedException` inside the `run()` method to allow for safe thread termination.
- To run your extended implementation with a countdown timer, replace the original `game instance` with your new `Timer` class in the main method

Use the following code as a starting point:

```
package hangman;

public class Timer extends Game {
    protected Thread timerThread;
    protected boolean timerRunning;

    // TODO: Override startGame() to also start the timer
    // TODO: Implement startTimer() to create and start the timer thread
}
```

```
package Hangman;

import javax.swing.SwingUtilities;

public class TimerRunnable implements Runnable {
    private final Timer game;

    public TimerRunnable(Timer game) {
        this.game = game;
```

```
    }

    @Override
    public void run() {
        // TODO: Implement the countdown loop
        // - Decrease timeLeft every second
        // - Update the time label using SwingUtilities.invokeLater()
        // - End the game if time runs out
    }
}
```

Exercise 3. Using Interfaces for Game Logic and Timer

To make your Hangman code more structured and easier to maintain, you will create two interfaces that separate the game logic and the timer.

Part 1: Create two interfaces

- Create an interface `GameLogic` with the following methods: `void handleGuess(char guess);`, `void hasWon(); void endGame(boolean won);`
- Create an interface `TimerLogic` with this method: `void startTimer();`

Part 2: One class implements both interfaces

- Create a class `HangmanGame` that implements both interfaces. If you already have a base class like `Game` or `GameGUI`, then HangmanGame should extend that class.
- Move the methods `handleGuess`, `hasWon`, `endGame` and `startTimer` in this class.

Base class given: `GameGUI`
Use the provided `GameGUI` class as a superclass. It already contains core GUI components and utility methods like `updateDisplay()`, `setupInput()` etc. You need to implement processInput() again here like in the first exercise and you will override and implement the abstract methods.

After refactoring, briefly answer the following questions:

- Which SOLID principles do you recognize in your solution?
- How does using interfaces improve the flexibility of your design?
- What benefits does this modular design using interfaces offer in larger, more complex projects?

In the following the abstract class GameGUI is provided:

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public abstract class GameGUI extends JFrame {
    protected JLabel wordLabel, attemptsLabel, timeLabel;
    protected JTextField inputField;
    protected String currentWord = "example";
    protected ArrayList <Character> guessedLetters = new ArrayList <>();
    protected int attemptsLeft = 6;
```

```
    protected int timeLeft = 60;

    public GameGUI() {
        setTitle("Hangman Game");
        setSize(400, 250);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new GridLayout(6, 1));

        JButton startButton = new JButton("Start Game");
        add(startButton);

        wordLabel = new JLabel("Word: ", SwingConstants.CENTER);
        add(wordLabel);

        attemptsLabel = new JLabel("Attempts left: " + attemptsLeft,
SwingConstants.CENTER);
        add(attemptsLabel);

        timeLabel = new JLabel("Time left: " + timeLeft,
SwingConstants.CENTER);
        add(timeLabel);

        inputField = new JTextField();
        inputField.setHorizontalAlignment(JTextField.CENTER);
        inputField.setEnabled(false);
        add(inputField);

        startButton.addActionListener(e -> startGame());
        processInput ();

        setVisible(true);
    }



    protected void startGame() {
        guessedLetters.clear();
        attemptsLeft = 6;
        timeLeft = 60;
        updateDisplay();
        inputField.setEnabled(true);
        inputField.requestFocus();
    }

    protected void updateDisplay() {
        StringBuilder display = new StringBuilder();
        for (char c : currentWord.toCharArray()) {
            if (guessedLetters.contains(c)) {
                display.append(c).append(" ");
```

```
            } else {
                display.append("_ ");
            }
        }
        wordLabel.setText("Word: " + display.toString());
        attemptsLabel.setText("Attempts left: " + attemptsLeft);
        timeLabel.setText("Time left: " + timeLeft);
    }

public void processInput() {
//implement again
}
    protected abstract void handleGuess(char guess);
    protected abstract boolean hasWon();
    protected abstract void endGame(boolean won);
}
```