Introduction to Programming

Exam Project 2025

Magnus Madsen

magnus@cs.au.dk

Formalia

You are to solve all of the programming problems described below. You must write several Java programs and document them using program comments. You should put each program in its own package. You should also describe any simplifications, omissions, or design choices you make, and these explanations should be included as program comments rather than in a separate report.

Requirements. The programs must be written in Java and should be well-written, well-structured, and clearly explained. You are not permitted to include any code from external sources, including books, papers, the internet, or fellow students. You *are* specifically permitted to use code from the textbook, e.g. `StdDraw`. Your unique anonymous identifier must be included at the top of every file.

- You must hand-in the *source code* of your Java programs.
- Your Java programs must compile.

GenAI. Use of GenerativeAI is not permitted.

Individuality. You must write your programs alone. You are welcome to *discuss* the problems with fellow students. Discussion means talking, not programming.

Workload. The expected workload is three days.

Submission. You must submit a zip-file that contains the source code of your programs. Let me repeat. A single *zip-file*, not any other type of archive file.

If you use Intellij IDEA, you are encouraged to include the entire project folder.

WiseFlow. You must submit your project using WiseFlow. If you are unable to submit your project, you must immediately contact:

```
Studieservice.Nat-Tech@au.dk
```

You cannot submit your project by writing to the lecturer.

1 Tic-Tac-Toe

Objective: You are to implement a simple tic-tac-toe game that runs in the terminal.

Problem 1.1. Write a class `TicTacToe` with a `main` method that expects a single command line argument n which is the size of the tic-tac-toe board. You should check that n is an integer and lies in the interval $[1, 8]$. Otherwise report an error to the user.

Note: The board is between 1×1 and 8×8 in size.

Problem 1.2. Add a method to print a tic-tac-toe board. Use `x` and `o` to represent the two players of the game. See below for an example. Show the initial empty tic-tac-toe board when the program starts.

Problem 1.3. Use `StdIn` to read a move from the player, i.e. a row and column position on the board where an `x` should be placed. Show the updated board.

Problem 1.4. Add checks to ensure that a move is valid:

- The position must unoccupied (i.e. no `x` or `o` is already there.)
- The position must be within bounds of the board.
- If a move is invalid, print an error to the user.

Problem 1.5. Add a loop that repeatedly asks the `x` player for a move. After each move, check if the player has won or lost, or it is a draw (i.e. the board is full).

Note: A player has won if they have 3 consecutive pieces in any direction.

Problem 1.6. Add a simple AI opponent. You may choose any strategy, such as:

- Randomly placing a piece on the board.
- Using an optimal winning strategy.

Ultimately, it should be possible to play tic-tac-toe from the terminal. For example:

```
$ java TicTacToe 4
. . . .
. . . .
. . . .
. . . .
Your move (row col): 0 0
x . . .
. . . .
. . . .
. . . .
AI plays: (1, 1)
x . . .
. o . .
. . . .
. . . .
Your move (row col): 0 1
x x . .
. o . .
. . . .
. . . .
AI plays: (2, 2)
x x . .
. o . .
. . o .
. . . .
Your move (row col): 0 2
x x x .
. o . .
. . o .
. . . .
You win!
```

2 Bouncing Ball

Objective: You are to implement a bouncing ball animation.

Problem 2.1. Write a program `Animation` that implements a bouncing ball animation using the `StdDraw` API. A single ball is placed in the middle of the canvas and moves in a random direction. When the ball hits one of the walls, i.e. the boundaries of the canvas, it bounces away from the wall.

- The canvas must be at least 800×600 pixels.
- Use double buffering to ensure that the animation is smooth.
- The ball must move in a random direction for each run of the program.

You are welcome to use the textbook example as a starting point.

Problem 2.2. Extend the program such that the "Window Title" displays the number of times the ball has collided with a wall. For example: `"Collisions: 17"`.

Problem 2.3. Draw a trail, like a comet, behind the ball. The trail should (a) have the same color as the ball and gradually fade-out, (b) get smaller and smaller, and (c) ultimately disappear entirely.

Hint: Use a `LinkedList` to store the last n positions of the ball.

Problem 2.4. Support multiple balls. The balls should have different colors and the animation should start with two balls. For now, balls are allowed to overlap and pass through each other.

Hint: Add a `Ball` class with with fields for the Ball's position, color, velocity, and trail. You can then store a list of balls.

Problem 2.5. When the mouse is pressed, create a new ball with a random color and moving in a random direction.

Hint: You may want to use `StdDraw.isMousePressed`, `StdDraw.mouseX`, and `StdDraw.mouseY`.

Problem 2.6. Remove one ball when two balls collide (i.e. overlap). You may choose which ball to remove.

3 Texas Hold 'Em

Objective: You are to implement a *simplified* version of the Texas Hold 'Em card game.

In Texas Hold 'Em, we have two players, each with two *hole cards*, and there are five *community cards* shared for everyone.

Problem 3.1. We use a simplified deck of cards with 48 cards, each uniquely identified by two attributes:

- `rank` – the value of the card, which is a number between 1 and 12 (inclusive).
- `suit` – the suite of the card. The deck has four suits: `Earth`, `Air`, `Fire`, and `Water`.

Write a class `Card` that stores the `rank` and `suit` of a card. Ensure that your class has an appropriate constructor, appropriate getter methods, and and `toString` method.

You may add additional data types, if you want.

Problem 3.2. Write a program `TexasHoldEm` to read a text file with the structure:

```
<CommunityCard1> ... <CommunityCard5>
<HoleCard1> <HoleCard2>
```

that prints out the description of the cards. For example, if `hands.txt` contains:

```
A9 W7 E5 A7 F6
E4 A3
```

Running `java TexasHoldEm < hands.txt` should print:

```
Community cards: 9 of Air, 7 of Water, 5 of Earth, 7 of Air, 6 of Fire
Hole cards: 4 of Earth, 3 of Air
```

You can assume that the input is always valid. The cards can be printed in any order.

Problem 3.3. The cards in our modified deck has a total order. We order them first by its rank (with 1 being the lowest and 12 being the highest), and then by suit (with earth being the lowest, followed by air, fire, and finally water being the highest).

- Add `equals` and `hashCode` methods to the `Card` class.
- Implement the `Comparable` interface for the `Card` class using the above total order.

Modify the `TexasHoldEm` program such that the community cards and hole cards are printed in increasing order. For example, for the previous input you should now print:

```
Community cards: 5 of Earth, 6 of Fire, 7 of Air, 7 of Water, 9 of Air
Hole cards: 3 of Air, 4 of Earth
```

If the community cards and/or hole cards contain duplicates print an error message to the user instead.

Problem 3.4. In Texas Hold 'Em, players try to make the best possible five-card hand from the best combination of cards chosen from the hole and community cards. In our game, we have four kinds of poker hands in decreasing order of ranking:

1. A *Flush* is a 5-card hand all of the same suit
2. A *Straight* is a 5-card hand with sequential rank
3. A *Pair* is a hand with two cards of the same rank
4. A *High Card* is a hand that does not fall into any other category

Write methods to recognize poker hands given some number of cards. You can assume that the `Card[]` array is never null, but might contain fewer or more than 5 cards.

- `Card getFlushWithHighCard(Card[] a)` Returns `null` if there is no flush. Otherwise returns the highest card in the flush. If there are multiple flushes, return the highest ranked flush card.
- `Card getStraightWithHighCard(Card[] a)` Returns `null` if there is no straight. Otherwise returns the highest card in the straight. If there are multiple straights, return the highest ranked straight card.
- `Card getPairWithHighCard(Card[] a)` Returns `null` if there is no pair. Otherwise returns the highest card in the pair. If there are multiple pairs, return the highest ranked pair card.
- `Card getHighCard(Card[] a)` Returns the highest card in the hand, or `null` otherwise.

Extend your program `TexasHoldEm` to also output the description of the best poker hand formed by choosing five cards from the hole and community cards. For example:

```
Community cards: 5 of Earth, 6 of Fire, 7 of Air, 7 of Water, 9 of Air
Hole cards: 3 of Air, 4 of Earth
Classification: Straight with 7 of Water
```