# Programmering og Problemløsning
# Datalogisk Institut, Københavns Universitet
# Arbejdsseddel 10 - gruppeopgave

Jon Sporring

9. december - 22. december.
Afleveringsfrist: torsdag d. 22. december kl. 22:00.

Denne arbejdsseddel dækker de følgende små 2 uger. I perioden skal vi udvide vores forståelse af objektorienteret programmering med hierakiske klassemodeller, og vi skal kigge på metoder til designe større programmer med objektorienteret design. Endeligt skal vi kigge på afprøvning fra et lidt mere systematisk perspektiv.

Denne arbejdsseddels læringsmål er:

- At bruge klasser, som er organiseret i et hierarki,
- At designe objektorienterede programmer med navne- og udsagnsordsmetoden,
- At kunne beskrive et klasse og objekt-design ved hjælp af UML-diagrammer,
- At kunne afprøve programmer set fra en brugers og en udviklers synspunkt.

Opgaverne er opdelt i øve- og afleveringsopgaver. I denne periode skal I arbejde i grupper med jeres afleveringsopgaver. Regler for gruppe- og individuelle afleveringsopgaver er beskrevet i “Noter, links, software m.m.” → “Generel information om opgaver”.

## Øveopgaver (in English)

10ø1 Write a `Person` class with data properties for a person’s name, address, and telephone number. Next, write a class named `Customer` that is a subclass of the `Person` class. The `Customer` class should have a positive integer data property for a unique customer number and a boolean data property indicating whether the customer wishes to be on a mailing list. Write a small program, which makes an instance of the `Customer` class.

10ø2 (a) Write an `Employee` class that keeps data properties for the following pieces of information:

- Employee name

- Employee number

Next, write a class named `ProductionWorker` that is a subclass of the `Employee` class. The `ProductionWorker` class should keep data properties for the following information:

- Shift number (an integer, such as 1 or 2)
- Hourly pay rate

The workday is divided into two shifts: day and night. The shift property will hold an integer value representing the shift that the employee works. The day shift is shift 1 and the night shift is shift 2. Write the appropriate methods for each class.

When you have written these classes, write a program that prompts the user to enter the relevant data for a production worker, and create an object of the `ProductionWorker` class. Then use the object's methods to retrieve the relevant data and display it on the screen.

(b) Extend the previous exercise as follows: Let a shift supervisor be a salaried employee who supervises a shift. In addition to salary, the shift supervisor earns a yearly bonus when his or her shift meets production goals. Write a `ShiftSupervisor` class that is a subclass of the `Employee` class you created in the previous exercise. The `ShiftSupervisor` class should keep a data property for the annual salary and a data property for the annual production bonus that a shift supervisor has earned. Demonstrate the class by writing a program that uses a `ShiftSupervisor` object.

(c) **(Extra difficult).** Considering that production during night shifts is reduced by 5% compared to production during day shifts, and that the hourly pay rate during night shifts is double the hourly pay rate during day shifts, compute the best possible worker & shift allocation over the period of 12 months. You need to think how to measure productivity and salary cost, and then find their best tradeoff in the period of 12 months.

10ø3 Extend `Customer` in Exercise 10ø1 to implement the `comparison` interface. The comparison method should compare customers based on their customer number. Create a list of several Customers and use List.sort to sort them.

---

10ø4 War is a card game for two players. A simplified version can be described as follows:

> War is a card game for two players using the so-called French-suited deck of cards. The deck is initially divided equally between the two players, which is organized as a stack of cards. A turn is played by each player showing the top of their stack. The player with the highest card wins the hand. Aces are the highest. The won cards are placed at the bottom of the winner's stack. When one player has all the cards, then that player wins the game.

Use the nouns-and-verbs method to identify possible classes and their interactions and write 1-2 lines of description for each.

10ø5 Checkers also known as draughts is a ancient board game. A simplified version can be described as follows:

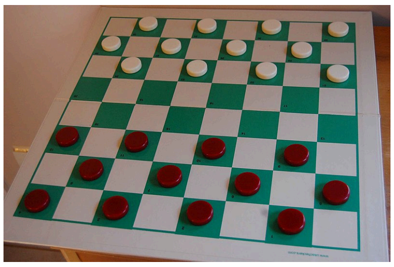
Figure 1: The starting position in Checkers [`https://commons.wikimedia.org/wiki/File:CheckersStandard.jpg`]

> Checkers is a turn-based strategy game for two players. The game is (typically) played on an $8 \times 8$ checkerboard of alternating dark- and light-colored squares. Each player starts with 12 pieces, where player one's pieces are light, and player two's pieces are dark in color, and the initial position of the pieces is shown in Figure 1. Players take turns moving one of their pieces. A player must move a piece if possible, and when one player has no more pieces, then that player has lost the game.
>
> A piece may only move diagonally into an unoccupied adjacent square. If the adjacent square contains an opponent's piece and the square immediately beyond is vacant, then the piece jumps over the opponent's piece and the opponent's piece is removed from the board.

Use the nouns-and-verbs method to identify possible classes and their interactions and write 1-2 lines of description for each.

---

10ø6 Draw the UML diagram for the following programming structure: A `Person` class has data property for a person's name, address, and telephone number. A `Customer` has data property for a customer number and a Boolean data property indicating whether the customer wishes to be on a mailing list.

10ø7 Draw an UML class diagram for the following structure:

A `Employee` class that keeps data properties for the following pieces of information:

- Employee name
- Employee number

A subclass `ProductionWorker` that is a subclass of the `Employee` class. The `ProductionWorker` class should keep data properties for the following information:

- Shift number (an integer, such as 1 or 2)
- Hourly pay rate

A class `Factory` which has one or more instances of `ProductionWorker` objects.

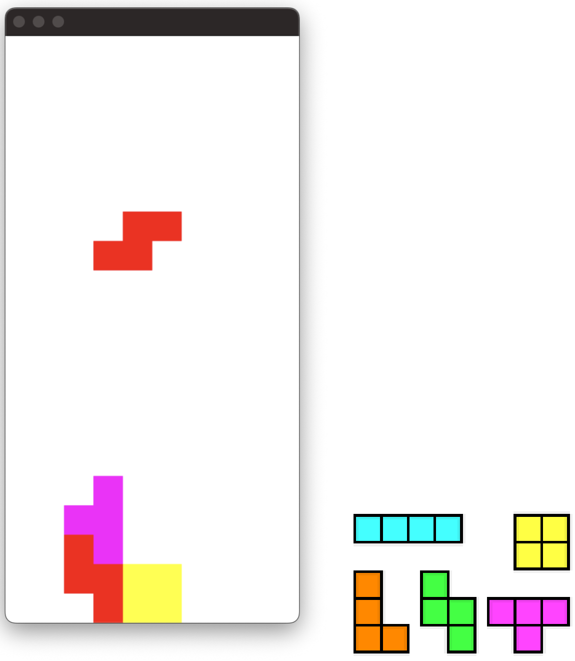
(a) (b)

Figure 2: Left: A tetris board in mid-game. Right: The available pieces (courtesy of wikipedia/Anypdetos). The L- and Z-pieces exists in two versions: Those shown here and their mirrored counterparts which are blue and red respectively.

## Afleveringsopgaver (in English)

This assignment is about a simple version of the game Tetris. Tetris was invented by Alexey Pajitnov in 1984. The game consists of a rectangular board of square fields and pieces of different colors. An example is shown in Exercise 2. You are to work with a game which has the following rules:

1. The game starts with a random piece position randomly and touching the top of the board. This is the first active piece.
2. The game takes turns by the user pressing a key.
3. The user can press the left, down, right, and space keys.
4. Left moves the piece one down and one to the left. Down moves the piece one down. Right moves the piece one down and one to the right. Space rotates the piece clock-wise 90 degrees.
5. Pieces cannot overlap.
6. If an active piece has no legal places to move by the user, then the game makes a new random piece placed at the top.
7. If the game no longer can place new pieces at the top, then the game is finished.
8. The players score is the number of pieces placed on the board.

10g0 Make a Canvas drawing function

```
draw: w: int -> h: int -> s: state -> Canvas.canvas
```

which on a canvas of size 300x600 pixels draws an Array2D of size 10x20 of squares. The Array2D must be of type `Color option[,]` where `Color` is defined as

```
type Color =
  | Yellow
  | Cyan
  | Blue
  | Orange
  | Red
  | Green
  | Purple
```

For example,

```
type board (w: int, h: int) =
  let _board = Array2D.create w h None
  do _board.[0,1] <- Some Green
  member this.width = w
  member this.height = h
  member this.board with get() = _board
type state = board
let draw (w: int) (h: int) (s: state) =
  // insert your definition of draw here

let b = board(10, 20)
let C = draw 300 600 b
show C "testing"
```

should draw a green square of size 30 by 30 one field below the top left corner.

10g1 Make the classes `square`, `straight`, `t`, `l`, `z`, where `l` and `z` takes a boolean argument for whether they should be their mirrored version or not. All classes must take the offset as the instantiation parameter. Further, all classes must inherit from the parent `tetromino` which must have the following type:

```
  type position = int*int
  type tetromino =
  /// The constructor with its initial shape, its final color,
   and its inital offset
  new: a: bool[,] * c: Color * o: position -> tetromino
  /// Make a string representation of this piece
  override ToString: unit -> string
  /// Make a deep copy of this piece
  member clone: unit -> tetromino
  /// Rotates the piece 90 degrees clock-wise such that its
   left-top offset is maintained
  member rotateRight: unit -> unit
  /// The piece' color
  member col: Color
  /// The present height of the shape
  member height: int
  /// The piece' present shape
  member image: bool[,]
  /// The piece' present offset
  member offset: position
  /// The present width of the shape
  member width: int
```

Hint: the keyword `base` is used to access the method of the parent class. E.g., `base.image` in a child class refers to the parent's member `image`. For example, instantiating a non-mirrored z-piece at position `(0,0)` as `z(false, (0,0))` should result in

```
val it: z =
  Image: [[false; true; true]
          [true; true; false]]
Color: Green
Offset: (0, 0)
```

while its mirrored counterpart `z(true, (0,0))` should result in

```
val it: z =
  Image: [[true; true; false]
          [false; true; true]]
Color: Red
Offset: (0, 0)
```

Use `draw` from Exercise 10g0 to ensure the pieces are as desired.

10g2 Make the `board` class which includes the game-mechanics of this version of tetris.:

```
type board =
  /// The constructor of a boad of w x h fields and which
   creates the first active piece at the top
  new: w: int * h: int -> board
  /// Make a string representation of this board
  override ToString: unit -> string
  /// Make a new piece and put it on the board if possible.
   Returns the piece or None
  member newPiece: unit -> tetromino option
  /// Put a piece on the board if possible. Returns true if
   successful
  member put: t: tetromino -> bool
  /// Take the active piece from the board. Returns a piece or
   None if no piece is active
  member take: unit -> tetromino option
  /// Return the board
  member board: Color option[,]
  /// The number of fields on the board vertically
  member height: int
  /// The number of fields on the board horizontally
  member width: int
```

Use `draw` from Exercise 10g0 and pieces from Exercise 10g1 to verify that your code works as expected.

10g3 Put all pieces together with a `react` function

```
react: s: state -> k: Canvas.key -> state option
```

with which you should be able to play this version of Tetris by

```
runApp "Tetris" 300 600 draw react (board (10, 20))
```

The `react` function must

- react to the user's key-presses
- move the active piece if possible according to the user's wishes
- if the active piece cannot be moved any longer, create a new piece

10g4 **(Optional)** A typical Tetris game will have an additional rule:

9. After the active piece no longer can be moved, any rows on the board which have no empty fields are removed, and an empty row is added at the top.

Add a method

```
removeRows: unit -> unit
```

to the `board` class which updates the board according til this additional rule, and add a call to it appropriately, e.g., in `react`.

10g5 Write a black- and white-box test of the `tetromino` classes and the `board` class

## Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `10g.zip`
- en opgavebesvarelse i pdf-format.

Zip-filen skal indeholde:

- en `Tetris` mappe med følgende og kun følgende filer:

  tetris.fsproj, tetris.fsi, tetris.fs, og tetrisApp.fsx

- pdf-dokumentet skal være lavet med LaTeX, benytte `report.tex` skabelonen, ganske kort dokumentere jeres løsning og besvare evt. ikke-programmeringsopgaver.

God fornøjelse.