Programmering og Problemløsning

15.2: Skak

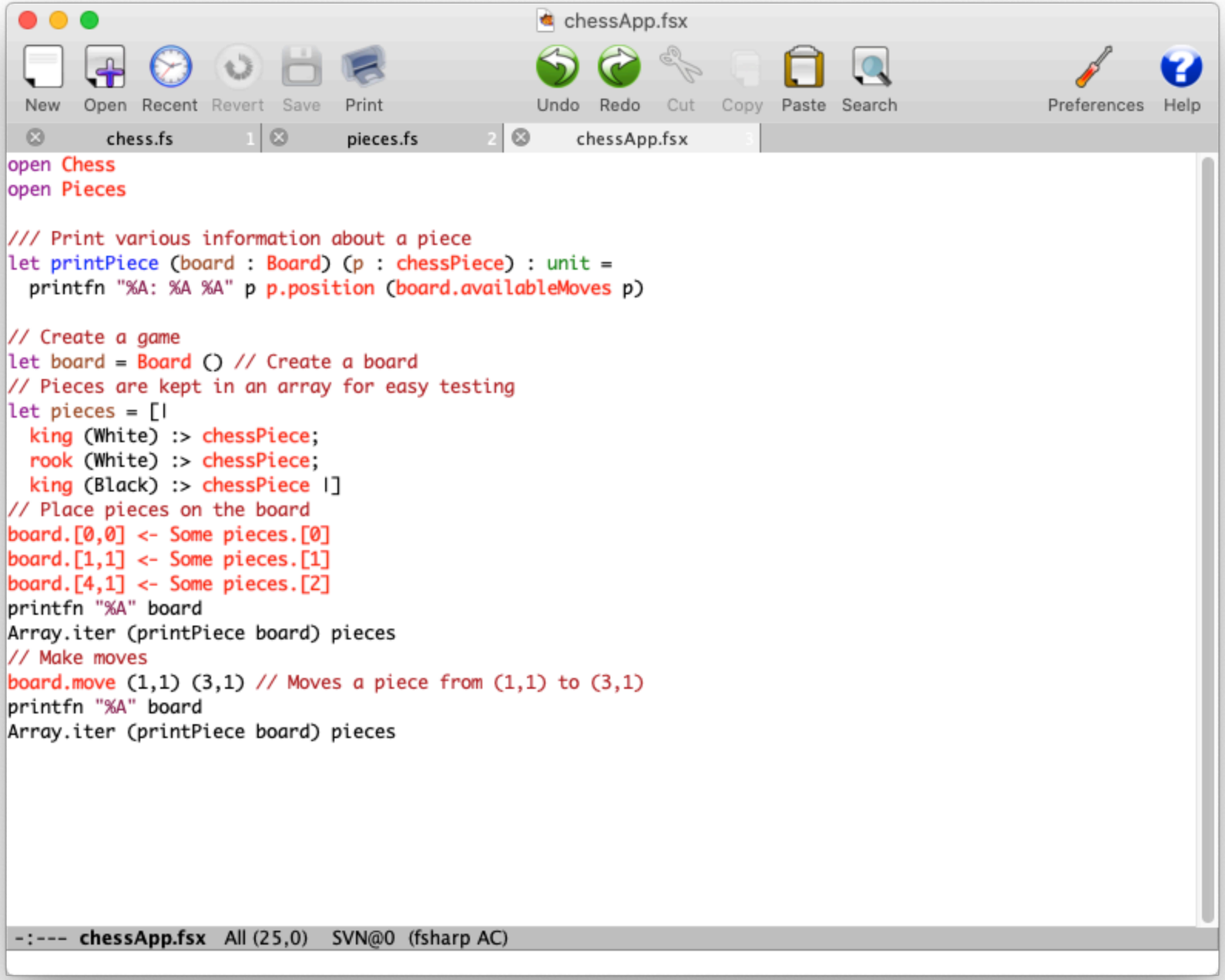

Jon Sporring - Computer (New Game)
8
7
6
5
4
3
2
1
A B C D E F G H

Opgave: chess.fs, pieces.fs, chessApp.fsx

Trin til at forstå koden:
1. Oversæt og kør. Virker den? Ser output ok ud?
2. Skab overblik over koden.
3. Forstå detaljer i koden.

1. Oversæt og kør f.eks. med:
% fsharpc chess.fs pieces.fs chessApp.fsx
% mono chessApp.exe

2. Overblik
- Hvilke klasser er der, hvilke medlemmer (members) har de?
- Tegn et UML diagram med relationer og medlemmer


chessApp.fsx
New
Open
Recent
Revert
Save
Print
Undo
Redo
Cut
Copy
Paste
Search
Preferences
Help
chess.fs
pieces.fs
chessApp.fsx
open Chess
open Pieces
/// Print various information about a piece
let printPiece (board : Board) (p : chessPiece) : unit =
printfn "%A: %A %A" p p.position (board.availableMoves p)
// Create a game
let board = Board () // Create a board
// Pieces are kept in an array for easy testing
let pieces = [|
king (White) :> chessPiece;
rook (White) :> chessPiece;
king (Black) :> chessPiece |]
// Place pieces on the board
board.[0,0] <- Some pieces.[0]
board.[1,1] <- Some pieces.[1]
board.[4,1] <- Some pieces.[2]
printfn "%A" board
Array.iter (printPiece board) pieces
// Make moves
board.move (1,1) (3,1) // Moves a piece from (1,1) to (3,1)
printfn "%A" board
Array.iter (printPiece board) pieces
-:--- chessApp.fsx All (25,0) SVN@0 (fsharp AC)

Dobbelt List.map

Alle kombinationer af bogstaver og tal:

```
> List.map (fun d -> "a"+d) ["1"; "2"; "3"];;
val it : string list = ["a1"; "a2"; "a3"]

> List.map (fun l -> List.map (fun d -> l+d) ["1";"2";"3"]) ["a"; "b"];;
val it : string list list = [["a1"; "a2"; "a3"]; ["b1"; "b2"; "b3"]]
```

Alle kombinationer af funktioner og tal:

```
> let indToRel = [
  fun elm -> (elm,0); // South by elm
  fun elm -> (-elm,0); // North by elm
  fun elm -> (0,elm); // West by elm
  fun elm -> (0,-elm) // East by elm
  ]
- List.map (fun e -> List.map e [1..7]) indToRel;;
val it : (int * int) list list =
  [[(1, 0); (2, 0); (3, 0); (4, 0); (5, 0); (6, 0); (7, 0)];
   [(-1, 0); (-2, 0); (-3, 0); (-4, 0); (-5, 0); (-6, 0); (-7, 0)];
   [(0, 1); (0, 2); (0, 3); (0, 4); (0, 5); (0, 6); (0, 7)];
   [(0, -1); (0, -2); (0, -3); (0, -4); (0, -5); (0, -6); (0, -7)]]
```

Mere om dobbelt List.map

Alle kombinationer af funktioner og tal:

```
> let indToRel = [
  fun elm -> (elm,0); // South by elm
  fun elm -> (-elm,0); // North by elm
  fun elm -> (0,elm); // West by elm
  fun elm -> (0,-elm) // East by elm
  ]
- List.map (fun e -> List.map e [1..7]) indToRel;;
```

Kombinationer med alternative List.map function og currying:

```
> let altMap lst e = List.map e lst
- List.map (altMap [1..7]) indToRel;;
val it : (int * int) list list =
  [[(1, 0); (2, 0); (3, 0); (4, 0); (5, 0); (6, 0); (7, 0)];
   [(-1, 0); (-2, 0); (-3, 0); (-4, 0); (-5, 0); (-6, 0); (-7, 0)];
   [(0, 1); (0, 2); (0, 3); (0, 4); (0, 5); (0, 6); (0, 7)];
   [(0, -1); (0, -2); (0, -3); (0, -4); (0, -5); (0, -6); (0, -7)]]
```

Mere om dobbelt List.map

Kombinationer med alternative List.map function og currying:

```
> let altMap lst e = List.map e lst
- List.map (altMap [1..7]) indToRel;;
val it : (int * int) list list =
  [[(1, 0); (2, 0); (3, 0); (4, 0); (5, 0); (6, 0); (7, 0)];
   [(-1, 0); (-2, 0); (-3, 0); (-4, 0); (-5, 0); (-6, 0); (-7, 0)];
   [(0, 1); (0, 2); (0, 3); (0, 4); (0, 5); (0, 6); (0, 7)];
   [(0, -1); (0, -2); (0, -3); (0, -4); (0, -5); (0, -6); (0, -7)]]
```

Kombinationer med swap og currying:

```
> let swap f a b = f b a
- List.map (swap List.map [1..7]) indToRel;;
val it : (int * int) list list =
  [[(1, 0); (2, 0); (3, 0); (4, 0); (5, 0); (6, 0); (7, 0)];
   [(-1, 0); (-2, 0); (-3, 0); (-4, 0); (-5, 0); (-6, 0); (-7, 0)];
   [(0, 1); (0, 2); (0, 3); (0, 4); (0, 5); (0, 6); (0, 7)];
   [(0, -1); (0, -2); (0, -3); (0, -4); (0, -5); (0, -6); (0, -7)]]
```

Upcasting og downcasting med [<AbstractClass>]

Upcasting i pieces.fs

```
let pieces = [|
  king (White) :> chessPiece;
  rook (White) :> chessPiece;
  king (Black) :> chessPiece |]
```

Upcasting:

+ vi kan maskere forskellige brikker som same type i listen
- Når vi skal bruge brikkernes specielle træk skal vi downcaste, men til hvilken type?

Skal brættet bruge downcasting for at få adgang til candiateRelativeMoves?

Nej! Abstrakte metoder beholder deres implementation ved upcasting

Hvad gør?

```
 member this.availableMoves (piece : chessPiece) : (Position list * chessPiece
list)  =
    match piece.position with
      None ->
        ([],[])
      | Some p ->
        let convertNWrap =
          (relativeToAbsolute p) >> this.getVacantNOccupied
        let vacantPieceLists = List.map convertNWrap piece.candiateRelativeMoves
        // Extract and merge lists of vacant squares
        let vacant = List.collect fst vacantPieceLists
        // Extract and merge lists of first obstruction pieces and filter out own
pieces
        let opponent =
          vacantPieceLists
          |> List.choose snd
        (vacant, opponent)(*//§\label{chessBoardEnd}§*)
```

Spørgsmål til opgave 11g?