ObjectIO
Clean Warm-Up

The goal of ObjectIO

- interactive objects
  - windows
  - menus
  - timers
  - receivers
  - tools for drawing
- interactive processes
  - communication with channels

ObjectIO
Clean Warm-Up

The structure of interactive objects

- every interactive object is located in the I/O state
- `ls` is the user-defined local state of the component
- together they form the process state

```
:: *IOSt st
:: PSt l
 = { ls : !l
   , io : !*IOSt l
   }
```

ObjectIO
Clean Warm-Up

The structure of interactive objects

- the objects themselves are represented as algebraic data types
  - the most trivial is `Void`
  - the names of the type constructors and the data constructors are the same

```
:: ButtonControl ls pst
 = ButtonControl
     String
     [ControlAttribute *(ls, pst)]
```

ObjectIO
Clean Warm-Up

The structure of interactive objects

- an object is interactive if it has an effect
  - e.g. a button can be pressed
  - when the effect is invoked, a callback function is called, which makes the state change as `(ls, pst) ↦ (ls, pst)`
  - these are often attributes of the object

```
:: ControlAttribute st
 = ... | ControlFunction (st -> st)
```

ObjectIO
Clean Warm-Up

The structure of interactive objects

- it is possible to glue objects together
  - composite objects can be formed
  - user interfaces are made this way
  - universal glue

```
:: :+: t1 t2 ls pst
 = (:+:) infixr 9
         (t1 ls pst) (t2 ls pst)
```

ObjectIO
Clean Warm-Up

The structure of interactive objects

- it is possible to glue objects together
  - you can define which objects can be glued using type constructor classes
  - the following enables gluing Controls together

```
instance Controls
  ButtonControl,
  ...
  :+: c1 c2 | Controls c1 &
              Controls c2
```

ObjectIO
Clean Warm-Up

The life cycle of interactive objects

- opening an interactive object

```
class Dialogs ddef where
  openDialog ::
    .ls
    !(ddef .ls !(PSt .l)) !(PSt .l) ->
    (!ErrorReport, !PSt .l)
```

ObjectIO
Clean Warm-Up

The life cycle of interactive objects

- opening an interactive object

```
(error, new_process_state)
  = openDialog
      local_dialog_state
      (Dialog
        “Title”
        (TextControl “Text”
                     [])
        [])
      previous_process_state
```

ObjectIO
Clean Warm-Up

The life cycle of interactive objects

- modifying an interactive object
  - the object to be modified has to be identified

```
(error, new_process_state)
  = openDialog
      local_dialog_state
      (Dialog
        “Title”
        (TextControl “Text”
                     [ControlId cId]) [])
      previous_process_state
```

ObjectIO
Clean Warm-Up

The life cycle of interactive objects

- modifying an interactive object
  - we perform the modification by a function of type `(IOSt l) -> (IOSt l)`
  - should we need to also change the local state, we would use `(PSt l) -> (PSt l)`

```
changeText process_state=:{io}
  = {process_state &
     io=setControlText cId
                       “Example 2”
                       io}
```

ObjectIO
Clean Warm-Up

The World

- I/O programs are of type `*World -> *World`
- in order to have an interactive object to work with, we have to create it using `startIO`

```
:: IdFun st :== st -> st
:: ProcessInit pst :== IdFun st

startIO :: !DocumentInterface
           !.l (ProcessInit (PSt .l))
           [(ProcessAttribute (PSt .l)]
           !*World -> *World
```

ObjectIO
Clean Warm-Up

```
module helloWorld
import StdEnv, StdIO

Start w = startIO NDI Void init [] w
where
  init pst
  # (e, pst) = openDialog Void hello pst
  | e <> NoError = closeProcess pst
  | otherwise = pst

  hello = Dialog “” (TextControl “HW” [])
                  [WindowClose (noLS closeProcess)]
```

ObjectIO
Clean Warm-Up

The World

- the interactive process has to be closed, too

```
closeProcess :: !(PSt .l) ->
                !(PSt .l)
```

ObjectIO
Clean Warm-Up

More on object identification

- before they can be used, the identifiers have to be created

```
openId  ::       !*env -> (!Id   !*env)
openIds :: !Int !*env -> (![Id] !*env)
```

- there are also receiver IDs and receiving/answering ones
- `World, IOSt .l` and `PSt .l` are identifier instances

ObjectIO
Clean Warm-Up

More on object identification

- two strategies for creating IDs
  - create IDs, then pass them on to the callback
  - store the created IDs in a record, then reference the record from the callback

ObjectIO
Clean Warm-Up

Drawing

- the environment of drawing is a `*Picture`
  - it has a coordinate system

```
viewDomainRange :==
  { corner1 = {x=0-(2^30), y=0-(2^30)}
  , corner2 = {x=   2^30 , y=   2^30 }
  }
```

  - it uses a pen, which defines the position, colour and font of the figure to be drawn

ObjectIO
Clean Warm-Up

Drawing

```
class Drawables figure where
  draw   ::          !figure !*Picture -> *Picture
  drawAt :: !Point2 !figure !*Picture -> *Picture
undraw   ::          !figure !*Picture -> *Picture
undrawAt :: !Point2 !figure !*Picture -> *Picture

// Fillables and Hilites are similar
```

- all of the above : boxes, rectangles
- above two : ovals, curves, polygons
- only Drawables : strings, vectors, bitmaps

ObjectIO
Clean Warm-Up

Drawing

```
:: PenAttribute = PenSize Int
                | PenPos Point2
                | PenColour Colour
                | PenBack Colour
                | PenFont Font

:: Point2 = {x :: !Int, y :: !Int}

instance zero Point2 where
  zero = { x = 0, y = 0 }

:: Colour = Black | ... | RGB RGBColour
:: RGBColour ={ r :: !Int, g :: !Int , b :: !Int}
```

ObjectIO
Clean Warm-Up

Drawing

```
:: FontDef = { fName   :: !FontName
             , fStyles :: ![FontStyle]
             , fSize   :: !FontSize
             }

openFont :: !FontDef !*Picture
         -> ( !( !Bool, !Font ), !*Picture )

openDefaultFont :: !*Picture -> (!Font,!*Picture)
openDialogFont  :: !*Picture -> (!Font,!*Picture)
```

- Bool: was the font found? if not, closest match

ObjectIO
Clean Warm-Up

Drawing

```
openBitmap :: !{#Char} !*env
           -> ( !Maybe Bitmap, !*env )
                                  | FileSystem env

resizeBitmap  :: !Size !Bitmap -> Bitmap
getBitmapSize ::       !Bitmap -> Size
```

- it is possible to make only “temporary changes” instead of permanent drawing

```
appXorPicture :: !.(IdFun *Picture) *Picture
              -> *Picture
```

ObjectIO
Clean Warm-Up

Drawing

- if we need only part of a picture, it can be easier to draw the whole picture, and clip out the rest

```
class toRegion area :: !area -> Region

instance toRegion Rectangle
instance toRegion [r]           | toRegion r
instance toRegion (:^: r1 r2) | toRegion r1 &
                                  toRegion r2

appClipPicture :: !Region !.(IdFun *Picture)
                  !*Picture     -> *Picture
```

ObjectIO
Clean Warm-Up

Making windows

- there are two basic types of windows, similar in construction: windows and dialogues

```
:: Window c ls pst
   = Window Title ( c ls pst )
            [WindowAttribute *( ls, pst )]

:: Dialog c ls pst
   = Dialog Title ( c ls pst )
            [WindowAttribute *( ls, pst )]
```

ObjectIO
Clean Warm-Up

Making windows

- creating and closing dialogues and windows

```
class Windows wdef where
  openWindow ::
    .ls !(wdef .ls !(PSt .l)) !(PSt .l) ->
    (!ErrorReport, !PSt .l)

closeWindow ::   !Id !(PSt .l) -> PSt .l
closeActiveWindow :: !(PSt .l) -> PSt .l
```

- remember: `WindowClose (noLS closeActiveWindow)`

ObjectIO
Clean Warm-Up

Making windows

- windows have multiple layers
  - low level: the document to be displayed
  - connection in between: controls
  - top level: the window frame
- there are two ways to change the contents of the window
  - indirect method: change the document, and it changes the visible part of its `Picture`
  - direct method: change the `Picture` itself

ObjectIO
Clean Warm-Up

Making windows

- indirect drawing

```
:: WindowAttribute = ...
                   | WindowLook       Bool Look
                   | WindowViewDomain ViewDomain
:: Look :== SelectState -> UpdateState ->
            *Picture -> *Picture

:: UpdateState = { oldFrame :: !ViewFrame
                 , newFrame :: !ViewFrame
                 , updArea  :: !UpdateArea }
:: ViewFrame :== Rectangle
:: UpdateArea :== [ViewFrame]
```

ObjectIO
Clean Warm-Up

Making windows

- indirect drawing
  - the Look callback renders the window

```
setWindowLook ::
  !Id !Bool !(!Bool, !Look) !(IOSt .l) -> IOSt .l

getWindowLook ::
  !Id !(IOSt .l) -> (!Maybe (Bool, Look), IOSt .l)
```

- direct drawing

```
appWindowPicture ::
  !Id !.(IdFun *Picture) !(IOSt .l) -> IOSt .l
```

ObjectIO
Clean Warm-Up

Handling control events

- window attributes include controls for keyboard and mouse events
- keyboard events

```
:: KeyboardState
     = CharKey Char KeyState
     | SpecialKey SpecialKey Keystate Modifiers
     | KeyLost

:: KeyState = KeyDown Bool
            | KeyUp
```

ObjectIO
Clean Warm-Up

Handling control events

- mouse events

```
:: MouseState = MouseMove Point2 Modifiers
              | MouseDown Point2 Modifiers
              | MouseDrag Point2 Modifiers Int
              | MouseUp Point2 Modifiers
              | MouseLost
```

ObjectIO
Clean Warm-Up

Modal dialogues

- blocking dialogue: the user has to fully handle the dialogue before proceeding

```
class Dialogs ddef where
  openModalDialog ::
        .ls  !(ddef .ls !(PSt .l))  !(PSt .l)
    -> (!(!ErrorReport, !Maybe .ls), !PSt .l)
```

ObjectIO
Clean Warm-Up

Controls

- `ButtonControl`
  - `CustomButtonControl`
- `CheckControl`
- `EditControl`
- `PopUpControl`
- `RadioControl`
- `SliderControl`
- `TextControl`

- organising the controls
  - `LayoutControl`
  - `CompoundControl`

ObjectIO
Clean Warm-Up

Controls

- putting controls together
  - `:+:`
  - lists can be more appropriate

```
:: ListLS t ls cs = ListLS [t ls cs]
:: NilLS    ls cs = NilLS
```

ObjectIO
Clean Warm-Up

Layout

- layout is defined with attributes
  - at fixed positions : (`Fix`, *position*)
  - in a corner: `LeftTop, RightTop, LeftBottom, RightBottom`
  - along lines: `Left, Center, Right`
  - relative to the previous object: (`X`, *position*), where `X` is `RightToPrev, LeftOfPrev, AbovePrev` or `BelowPrev`
  - the above can be modified by offsets

ObjectIO
Clean Warm-Up

Menus

- opening menus
  - pop-up menus cannot have submenus

```
class Menus mdef where
  openMenu :: .ls (!mdef .ls (PSt .l)) !(PSt .l)
           -> ( !ErrorReport, !PSt .l )

instance Menus (Menu      m)| MenuElements      m
instance Menus (PopUpMenu m)| PopUpMenuElements m

:: Menu m ls pst = Menu Title (m ls pst)
                        [MenuAttribute *(ls, pst)]
:: PopUpMenu m ls pst = PopUpMenu (m ls pst)
```

ObjectIO
Clean Warm-Up

Menus

- menu attributes

```
:: MenuAttribute   st
 = MenuId                Id
 | MenuSelectState       SelectState
 | MenuIndex             Int
 | MenuInit              (IdFun st)
 | MenuFunction          (IdFun st)
 | MenuMarkState          MarkState
 | MenuModsFunction      (ModifiersFunction st)
 | MenuShortKey          Char
```

ObjectIO
Clean Warm-Up

Menus

- constructing the menu

```
:: MenuItem ls pst
   = MenuItem Title [MenuAttribute *(ls, pst)]

:: MenuSeparator ls pst
   = MenuSeparator [MenuAttribute *(ls, pst)]

:: SubMenu m ls pst
   = SubMenu Title (m ls pst)
             [MenuAttribute *(ls, pst)]
```

... also: menu glue `:+:` and `NilLS` and `ListLS`