Sockets: ”server”

```
with Ada.Command_Line; use Ada.Command_Line;
with Ada.Exceptions;   use Ada.Exceptions;
with Ada.Text_IO;      use Ada.Text_IO;

with TJa.Sockets;      use TJa.Sockets;

procedure Server is

   -- Servern behöver en Listener_Type för
   -- att ta emot inkommande anslutningar
   Lyssnare  : Listener_Type;

   -- Socket_type används för att kunna
   -- kommunicera med en klient
   Socket    : Socket_Type;

   -- Text - används för att lagra en textrad
   -- Textlangd - radens längd
   -- Antal_E - antal ’E’ i Text
   Text                : String(1..100);
   Textlangd, Antal_E : Natural;

begin
   -- Kontrollerar att programmet startas med
   -- en parameter (portnumret i detta fall).
   -- Annars kastas ett fel.
   -- Starta programmet enligt:
   --   server 3400
   if Argument_Count /= 1 then
      Raise_Exception(Constraint_Error’Identity,
                      "Usage: " & Command_Name &
                     " port");
   end if;
```

```
  -- Initierar lyssnaren ”på” en port
  Initiate(Lyssnare,
           Natural’Value(Argument(1)));

  -- Väntar tills en anslutning kommer,
  -- krävs att en klient kör connect
  Wait_For_Connection(Lyssnare, Socket);

  -- Nu har en anslutning skapats och vi kan
  -- börja använda den ...
  Put_Line("Klient ansluten...");
  loop
    --Väntar på en sträng från klienten
    Get_Line(Socket, Text, Textlangd);
    --Letar rätt på antalet ’E’ i denna text
    Antal_E := 0;
    for I in 1 .. Textlangd loop
      if Text(I) = ’E’ then
        Antal_E := Antal_E + 1;
      end if;
    end loop;
    --Skickar resultatet tillbaka
    Put_Line(Socket, Antal_E);
  end loop;
  Close(Socket); -- Kommer aldrig hit, men ...

exception  --Lite felhantering.
  when Constraint_Error =>
    Put("Du matade inte in en paramter ”);
    Put_Line(”innehållande portnummer.");
  when others =>
    -- Kanske End_Error eller Socket_Error,
    -- det betyder att klienten kopplat ned
    -- förbindelsen. Stängas även härifrån.
    Put_Line("Nu dog klienten");
    Close(Socket);
end server;
```

Sockets: ”klient”

```
with Ada.Command_Line; use Ada.Command_Line;
with Ada.Exceptions;   use Ada.Exceptions;
with Ada.Text_IO;      use Ada.Text_IO;

with TJa.Sockets;      use TJa.Sockets;

procedure Klient is

   -- Socket_Type används för att kommunicera
   -- med en server
   Socket : Socket_Type;

   -- Text - används för att lagra en textrad
   -- Textlangd - radens längd
   -- Resultat - resultatet från servern
   Text      : String(1..100);
   Textlangd : Natural;
   Resultat  : Natural;

begin
   -- Kontrollerar att programmet startas med
   -- två parametrar (serverdator och portnummer
   -- i detta fall). Annars kastas ett fel.
   -- Starta programmet enligt:
   --   klient zaza12 3400
   --   klient localhost 3400
   if Argument_Count /= 2 then
    Raise_Exception(Constraint_Error'Identity,
                    "Usage: " & Command_Name &
                    " serverhost serverport");
   end if;
```

```
   -- Initierar en socket, detta krävs för att
   -- kunna ansluta denna till servern.
   Initiate(Socket);

   -- Ansluter till servern
   Connect(Socket, Argument(1),
           Positive'Value(Argument(2)));

   loop
     --Läser in en sträng från användaren
     Put_Line("Mata in, ""exit"" => stop");
     Get_Line(Text,Textlangd);
     if Textlangd=100 then
       Skip_Line;
     end if;

     exit when Text(1..4) = "exit";

     -- Skicka strängen till servern
     Put_Line(Socket,Text(1..Textlangd));
     -- Vänta på att få tillbaka resultatet
     Get(Socket,Resultat);
     Skip_Line(Socket);
     -- Skriv ut på skärmen
     Put("Strängen innehöll");
     Put(Integer'Image(Resultat));
     Put_Line(" st E");
   end loop;

   -- Innan programmet avslutar stängs socketen
   -- (bryts kontakten), detta genererar ett
   -- undantag (exception) hos servern.
   Close(Socket);
   -- På motsvarande sätt skulle denna klient
   -- kunna få ett undantag om servern bryter
   -- förbindelsen
end Klient;
```

Hur kompilera in TJa-biblioteket?

På kurshemsidan (under projekt) finns det en länk till TJa-bibliotekets dokumentation (studentversionen). Där finner ni de paketspecifikationer som visar vad de olika paketen i biblioteket innehåller.

När ni använder saker ur TJa-biblioteket (d.v.s. när ni skrivit något likt följande i ert program)

```
with TJa.Xyz;
```

skall själva kompileringen se lite annorlunda ut:

```
gnatmake '~TDDD11/lib/TJa/bin/tja_config'
         program.adb
```

OBS! Man skall absolut inte kopiera några filer från TJa-biblioteket. Dessa inkluderas på rätt sätt när man utför ovanstående.

Om man ändå råkat kopiera några delar från hemsidan kommer kompileringen att ge felmeddelanden av typen

```
Missing file tja-xyz.adb
```

Tag då bort de filer som ni kopierat så löser sig problemet.

TJa-biblioteket (studentversion)

De olika delarna i TJa-biblioteket som ni studenter kan använda är följande:

```
TJa.Calendar

TJa.File_IO

TJa.Keyboard              key_codes
TJa.Keyboard.Keys

TJa.Lists
TJa.Lists.Unchecked.Double_Linked.
        General_List.Checked_Data
        Sorted_List.Checked_Data
        Unsorted_List.Checked_Data

TJa.Sockets

TJa.Window.Elementary
TJa.Window.Text
TJa.Window.Graphic

TJa.Misc
```

Trevliga saker om strängar

Det finns en strängtyp som heter `Unbounded_String`. Denna anpassar sig efter vad man lagrar i den. Kan vara trevligt att använda i vissa fall.

För att hantera detta i samband med de vanliga strängarna kan det vara bra att kolla in de rutiner som hör till denna datatyp. Kolla i paketet `Ada.Strings.Unbounded`.

Några bra saker (till att börja med) är:

```
   S  : String(1 .. 10);
   US : Unbounded_String;

begin
   ...
   US := To_Unbounded_String(S);
   S := To_String(US);
   L := Length(US);
   ...
end ...;
```

Vill man hantera vanliga String lite mer kan man titta i paketet `Ada.Strings.Fixed`.

Några specialla saker som hör till stränghanteringen finns också i paketet `Ada.Strings`.

Grafiska delen i TJa-biblioteket

```
with Ada.Text_IO;              use Ada.Text_IO;
with TJa.Window.Elementary;    use ...
with TJa.Window.Text;          use ...
with TJa.Window.Graphic;       use ...

procedure Test_TJa is

   Msg  : constant String := " Hello world ";
   X_UL : constant Integer := 10;
   Y_UL : constant Integer := 2;

begin
   Reset_Colours;  -- Standard (black & white)
   Clear_Window;
   Set_Graphical_Mode(On);

   -- Draw a rectangle on screen ...
   Goto_XY(X_UL, Y_UL);
   Put(Upper_Left_Corner);
   Put(Horisontal_Line, Times => Msg'Length);
   Put(Upper_Right_Corner);

   Goto_XY(X_UL, Y_UL + 1);
   Put(Vertical_Line);
   Goto_XY(X_UL + Msg'Length + 1, Y_UL + 1);
   Put(Vertical_Line);

   Goto_XY(X_UL, Y_UL + 2);
   Put(Lower_Left_Corner);
   Put(Horisontal_Line, Times => Msg'Length);
   Put(Lower_Right_Corner);

   Set_Graphical_Mode(Off);
```

```
   -- Prints the message inside the rectangle ...
   Goto_XY(X_UL + 1, Y_UL + 1);

   Set_Background_Colour(Blue);
   Set_Foreground_Colour(White);
   Set_Bold_Mode(On);

   Put(Msg);

   Reset_Colours;
   Reset_Text_Modes;  -- Resets bold mode ...

   Goto_XY(1, Y_UL + 4);
end Test_TJa;
```

OBS! Detta är inget äkta grafiskt paket utan en utökning av den vanliga texthanteringen i ett terminalfönster med t.ex. färger, några grafiska tecken m.m.

Varje grafiskt tecken motsvarar ett vanligt tecken, men det ser ut som ett vågrätt/lodrätt streck eller ett hörn.

Det man kan få fram är dock figurer som innehåller räta linjer horisontellt och vertikalt. Ett exempel skulle kunna vara att man kan rita ett vanligt husfönster.

Tangenthanteringsdelen i TJa-biblioteket

För att göra det lite roligare och användarvänligare i projekten är det meningen att ni skall undvika saker som att användaren matar in koordinater för att placera ut skepp, kryss m.m.

Användaren skall kunna köra ert projekt enkelt. T.ex. genom att använda sig av piltangenter för att flytta sig till en ruta man vill placera ut ett kryss på ett luffarschacks-bräde eller vinkla någon kanon så att man kan skjuta i en viss vinkel.

För detta behövs något för att hantera tangentbordet lite smidigare. Till detta använder ni `TJa.Keyboard`-paketet.

```
with Ada.Text_IO;              use Ada.Text_IO;
with TJa.Window.Elementary;    use ...
with TJa.Keyboard;             use ...

procedure Move_Cursor is
  X, Y : Positive := 10;
  Key  : Key_Type;
begin
  loop
    Goto_XY(X, Y);
    Get_Immediate(Key);
    exit when Is_Esc(Key) or (X = 1);
    if Is_Left_Arrow(Key) then
      X := X - 1;
    end if;
  end loop;
end Move_Cursor;
```

Innan ni kommer allt för långt i ert projekt

VIKTIGT! Hur skall ni lagra era data i projektet? Vad har ni för olika möjligheter?

Enstaka variabler
Poster
Fält
Listor
Filer

Kombinationer av ovanstående

Hur skall ni strukturera ert program? Skall det finnas paket/underprogram/...? En tanke man bör ha är att man skall se om det finns saker som används både i server och klient. Isåfall är det praktiskt att lägga sådana saker i ett (eller flera) paket. Vad kan det vara?

Nätverksdelen (socketshanteringen)
Lagring/hantering av spelplan
Interaktion med användaren

Att dela upp projektet i olika delar gör att ni kan arbeta parallellt. T.ex. en person tar hand om ett paket, en annan tar hand om hur man spelar själva spelet och en ser till att användardialogen blir snygg och funktionell.

VIKTIGT! Planera så att ni vet hur ni skall anropa de olika sakerna i paketen och så att ni inte gör så att ni inte får ihop sakerna på slutet.