Балтийский государственный технический университет «Военмех» им. Д. Ф. Устинова

Кафедра И5
«Информационные системы и программная инженерия»

ЛАБОРАТОРНАЯ РАБОТА № 3

По дисциплине **«Визуальное программирование»**

На тему

«КОМПОНЕНТ LISTVIEW. КОЛЛЕКЦИИ НА ПРИМЕРЕ СПИСКОВ»

Вариант № 4

Выполнил:
Студент Васильев Н. А.
Группа И967

Преподаватель:
Ракова И. К.

Санкт-Петербург
2018

Цель работы:

Получить навыки работы с компонентом ListView, научиться работать с коллекциями на примере объектов класса TObjectList.

Задание:

1. Разработать и реализовать иерархию классов (ядро системы) в соответствии с вариантом, выданным преподавателем, позволяющую моделировать указанную систему без привязки к интерфейсу пользователя. Обязательно применение класса TStringList (TObjectList);
2. Разработать интерфейс пользователя и согласовать его с преподавателем.
3. Реализовать интерфейс пользователя с необходимой функциональностью, используя разработанную ранее иерархию классов.

Необходимо провести моделирование процесса выделения памяти под совокупность задач. На входе пользователь задает объем памяти (в байтах), список задач с соответствующими атрибутами. На выходе необходимо отобразить список всех задач; список задач, загруженных в память в данный момент времени; список задач, не загруженных вовремя в память; список отработавших задач; карту памяти (например, memo-поле из нулей и единиц, где 0 – свободный байт, 1 – занятый байт). Связный список – самый подходящий участок. Данный алгоритм выполняет поиск по всему списку и выбирает наименьший по размеру подходящий свободный фрагмент. Затем этот участок делится на две части: одна отдается задаче, а другая остается неиспользуемой. Данный алгоритм должен просматривать весь список от начала и до конца.

Текст программы:

task_unit;

```
unit task_unit;
interface
uses
  Classes, SysUtils, ExtCtrls,
Contnrs, memoryBlock_unit, Dialogs;
type Task = class
  TaskName: string;
  TaskSize: integer;
  TaskStartTime: string;
  TaskTotalTime: integer;
  TaskStatus: string;
  TaskTimer: TTimer;
  TaskUsedMemoryBlock: integer;
  constructor Create(Name: string;
Size: integer; TotalTime: integer);
  destructor Free;
  function isInQueue(): boolean;
  function isComplete(): boolean;
  procedure onTimerTick(Sender:
TObject);
  procedure TaskStart(fromTimer:
integer; MemBlock: MemoryBlock);
  procedure
FreeMemoryBlock(MemBlock:
MemoryBlock);
end;
implementation
```

```
constructor Task.Create(Name:
string; Size: integer; TotalTime:
integer);
begin
  TaskName := Name;
  TaskSize := Size;
  TaskStartTime := '-';
  TaskTotalTime := TotalTime;
  TaskStatus := 'В очереди';
  TaskTimer := TTimer.Create(nil);
  TaskTimer.Interval := 1000;
  TaskTimer.OnTimer := @OnTimerTick;
  TaskTimer.Enabled := False;
end;
destructor Task.Free;
var i: integer;
begin
  TaskTimer.Free;
end;
procedure Task.onTimerTick(Sender:
TObject);
begin
  if TaskStatus = 'Завершена' then
Exit;
  dec(TaskTotalTime);
  if (TaskTotalTime = 0) then
  begin
    TaskTimer.Enabled := False;
    TaskStatus := 'Завершена';
  end;
end;
procedure Task.TaskStart(fromTimer:
integer; MemBlock: MemoryBlock);
var i, place: integer;
begin
  place :=
MemBlock.findPlace(TaskSize);
  if place <> -1 then begin
    TaskUsedMemoryBlock := place;
    i := 0;
    while (i < TaskSize) do begin
      MemBlock.MbBytes[place + i] :=
'1';
      i := i + 1;
    end;
    TaskTimer.Enabled := True;
    TaskStartTime :=
IntToStr(fromTimer);
    TaskStatus := 'Выполняется';
  end;
end;
function Task.isInQueue(): boolean;
begin
  if TaskStatus = 'В очереди' then
    isInQueue := True
  else
    isInQueue := False;
end;
function Task.isComplete(): boolean;
begin
   if TaskStatus='Завершена' then
     isComplete := True
   else
     isComplete := False;
end;
procedure
Task.FreeMemoryBlock(MemBlock:
MemoryBlock);
var i:integer;
begin
    if TaskUsedMemoryBlock = -1 then
Exit;
    i := 0;
    while (i < TaskSize) do begin
MemBlock.MbBytes[TaskUsedMemoryBlock
+ i] := '0';
      i := i + 1;
    end;
    TaskUsedMemoryBlock := -1;
end;
end.
```

memoryBlock_unit:

```
unit memoryBlock_unit;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, Dialogs;
type
  MemoryBlock = class
       MbTotal: integer;
       MbBytes: string;
       constructor
Create(n:integer);
       function ByteLine(): string;
       function findPlace(n:
integer): integer;
  end;
implementation
  constructor
MemoryBlock.Create(n:integer);
  var i: integer;
  begin
       MbTotal := n;
       MbBytes := StringOfChar('0',
MbTotal);
  end;
  function MemoryBlock.ByteLine():
string;
  begin
       Result := MbBytes;
  end;
  function MemoryBlock.findPlace(n:
integer): integer;
  var i, j, bestPlace,
currentBlockStart, bestSize,
currentBlockSize: integer;
    c, curType: char;
  begin
       bestPlace := -1;
       bestSize := -1;
       currentBlockSize := 0;
       i := 1;
       while (i <= MbTotal) do begin
         if (MbBytes[i] = '0') then
begin
           c := '0';
           j := i;
           currentBlockSize := 0;
           currentBlockStart := i;
           while ((MbBytes[j] = '0')
and (j <= MbTotal)) do begin
             currentBlockSize :=
currentBlockSize + 1;
             j := j + 1;
           end;
           if ((currentBlockSize <
bestSize) or (bestSize = -1)) and
(currentBlockSize >= n) then begin
             bestPlace :=
currentBlockStart;
             bestSize :=
currentBlockSize;
           end;
           i := currentBlockStart +
currentBlockSize + 1;
         end
         else
         i := i + 1;
       end;
       Result := bestPlace;
  end;
end.
```

unit1:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil,
Forms, Controls, Graphics, Dialogs,
StdCtrls,
  ExtCtrls, ComCtrls, task_unit,
memoryBlock_unit, Contnrs, Math;
type
  { TForm1 }
```

```
  TForm1 = class(TForm)
    btnMemory: TButton;
    btnAddTask: TButton;
    editName: TEdit;
    editSize: TEdit;
    editMemory: TEdit;
    editDuration: TEdit;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    TimeLabel: TLabel;
    Label4: TLabel;
    ListView1: TListView;
    Memo1: TMemo;
    StartButton: TButton;
    Switcher: TRadioGroup;
    Timer1: TTimer;
    procedure
btnAddTaskClick(Sender: TObject);
    procedure btnMemoryClick(Sender:
TObject);
    procedure FormClose(Sender:
TObject; var CloseAction:
TCloseAction);
    procedure FormCreate(Sender:
TObject);
    procedure FormResize(Sender:
TObject);
    procedure
StartButtonClick(Sender: TObject);
    procedure SwitcherClick(Sender:
TObject);
    procedure Timer1Timer(Sender:
TObject);
    function isNum(str: string):
boolean;
    procedure printMemory(Place:
TStrings);
    procedure printTasks(status:
string);
    procedure updateTasks;
  private
    sw: string;
    Queue: TObjectList;
    Memory: MemoryBlock;
    taskCount: integer;
  public
  end;
var
  Form1: TForm1;
implementation
{$R *.lfm}
{ TForm1 }
function TForm1.isNum(str: string):
boolean;
var num: integer;
begin
  try
    num := StrToInt(str);
  except
    On EConvertError do begin
    ShowMessage('Введите число!');
    Result := False;
    Exit(Result);
    end;
  end;
  if (num > 0) then
    Result := True
  else begin
    ShowMessage('Введите число
больше нуля!');
    Result := False;
  end;
end;
procedure TForm1.FormCreate(Sender:
TObject);
var i: integer;
begin
ListView1.Columns.Add.Caption:='Зада
ча';
ListView1.Columns.Add.Caption:='Разм
ер';
ListView1.Columns.Add.Caption:='Врем
```

```
я';
ListView1.Columns.Add.Caption:='Стар
т';
ListView1.Columns.Add.Caption:='Стат
ус';
  Queue := TObjectList.Create;
  sw := 'all';
  for i:=0 to ListView1.ColumnCount-
1 do
ListView1.Columns[i].Width:=Floor(Fo
rm1.Width/(ListView1.ColumnCount));
end;
procedure TForm1.FormResize(Sender:
TObject);
var i: integer;
begin
  for i:=0 to ListView1.ColumnCount-
1 do
ListView1.Columns[i].Width:=Floor(Fo
rm1.Width/(ListView1.ColumnCount));
end;
procedure TForm1.printMemory(Place:
TStrings);
begin
    Place[0] :=
MemoryBlock(Memory).ByteLine;
end;
procedure
TForm1.btnMemoryClick(Sender:
TObject);
begin
  if (isNum(editMemory.Text) = True)
then begin
     Memory :=
MemoryBlock.Create(StrToInt(editMemo
ry.Text));
     printMemory(Memo1.Lines);
     btnAddTask.Enabled := True;
     btnMemory.Enabled := False;
     editMemory.Enabled := False;
     btnMemory.Caption := 'Память
выделена';
     StartButton.Enabled := True;
  end;
end;
procedure TForm1.FormClose(Sender:
TObject; var CloseAction:
TCloseAction);
begin
  Queue.Free;
  Memory.Free;
end;
procedure
TForm1.btnAddTaskClick(Sender:
TObject);
var addedTask: Task;
begin
  if ((isNum(editSize.Text) = True)
and (isNum(editDuration.Text)) =
True) then begin
     if (editName.Text = '') then
       ShowMessage('Введите имя
задачи!')
     else begin
Queue.Add(Task.Create(editName.Text,
StrToInt(editSize.Text),
StrToInt(editDuration.Text)));
       if StartButton.Enabled =
False then
          Task(Queue[Queue.Count -
1]).TaskStart(StrToInt(TimeLabel.Cap
tion), Memory);
       printTasks(sw);
     end;
     printMemory(Memo1.Lines);
  end;
end;
procedure TForm1.printTasks(status:
string);
var i: integer;
ListItem:TListItem;
begin
  ListView1.Clear;
  if status = 'all' then begin
```

```
    for i:= 0 to Queue.Count-1 do
begin
      ListItem :=
ListView1.Items.Add;
ListItem.Caption:=Task(Queue[i]).Tas
kName;
ListItem.SubItems.Add(inttostr(Task(
Queue[i]).TaskSize));
ListItem.SubItems.Add(inttostr(Task(
Queue[i]).TaskTotalTime));
ListItem.SubItems.Add(Task(Queue[i])
.TaskStartTime);
ListItem.SubItems.Add(Task(Queue[i])
.TaskStatus);
    end;
  end
  else begin
      for i:=0 to Queue.Count-1 do
    begin
      if((Task(Queue[i]).TaskStatus
= status) and (status = sw)) then
      begin
       ListItem :=
ListView1.Items.Add;
ListItem.Caption:=Task(Queue[i]).Tas
kName;
ListItem.SubItems.Add(inttostr(Task(
Queue[i]).TaskSize));
ListItem.SubItems.Add(inttostr(Task(
Queue[i]).TaskTotalTime));
ListItem.SubItems.Add(Task(Queue[i])
.TaskStartTime);
ListItem.SubItems.Add(Task(Queue[i])
.TaskStatus);
       end;
    end;
  end;
end;
procedure TForm1.UpdateTasks;
var i: integer;
    ListItem: TListItem;
begin
  for i:=0 to Queue.Count-1 do begin
    ListItem := ListView1.Items[i];
    if ListItem <> nil then begin
     ListItem.SubItems[1] :=
inttostr(Task(Queue[i]).TaskTotalTim
e);
     ListItem.SubItems[3] :=
Task(Queue[i]).TaskStatus;
     if(Task(Queue[i]).isComplete())
then
       begin

Task(Queue[i]).FreeMemoryBlock(Memor
y);
       end;
    end;
    if Task(Queue[i]).isInQueue()
then
Task(Queue[i]).TaskStart(StrToInt(Ti
meLabel.Caption), Memory);
  end;
  printTasks(sw);
  printMemory(Memo1.Lines);
end;
procedure
TForm1.StartButtonClick(Sender:
TObject);
var i: integer;
begin
  if Queue.Count = 0 then begin
     ShowMessage('Введите задачу!');
     Exit;
  end;
  Timer1.Enabled := True;
  StartButton.Enabled := False;
  if Queue <> nil then
     for i := 0 to Queue.Count-1 do

Task(Queue[i]).TaskStart(StrToInt(Ti
meLabel.Caption), Memory);
  StartButton.Caption := 'Процесс
запущен';
```

```
end;
procedure
TForm1.SwitcherClick(Sender:
TObject);
begin
  case Switcher.ItemIndex of
  0 : sw := 'all';
  1 : sw := 'Выполняется';
  2 : sw := 'В очереди';
  3 : sw := 'Завершена';
  end;
  printTasks(sw);
  printMemory(Memo1.Lines);
end;
procedure TForm1.Timer1Timer(Sender:
TObject);
begin
  Memo1.Lines.Clear;
  printMemory(Memo1.Lines);
  updateTasks;
  TimeLabel.Caption :=
IntToStr(StrToInt(TimeLabel.Caption)
+ 1);
end;
end.
```

Результаты работы программы: