1 TimeSpanUnit

Folgend ist die Dokumentation der Aufgabe 1 der Übung 5 beschrieben, welche das Verpacken des Programms TimeSpan in eine eigene Unit verlangt.

1.1 Lösungsansatz

Es sollen alle Hauptfunktionalitäten des Programms TimeSpan in eine Unit verpackt werden, die Testmethoden sowie die Fehlerauswertung sollen hierbei weggelassen werden, da sich um die Fehlerbehandlung der Aufrufer kümmern soll. Lediglich ein String mit der Fehlernachricht soll enthalten sein, der die Art des Fehlers beschreibt. Um sicherzustellen, dass die Funktionalität weiter besteht sollen erneut Tests durchgeführt werden.

Die neue Funktion ShorterThan soll die bereits implementierte Funktion TimeSpanToSeconds verwenden, um die beiden gegebenen Zeitdarstellungen der beiden TimSpan Instanzen zuerst in Sekundendarstellung zu konvertieren und anschließend die Sekundenwerte gegeneinander zu vergleichen. Hierbei soll als Ergebnis False zurückgegeben werden, wenn der Sekundenwert der ersten TimSpan Instanz kleiner gleich des Sekundenwertes der zweiten Instanz ist, oder wenn einer der beiden Instanzen einen ungültigen Zeitwert beinhaltet, ansonsten soll True zurückgegeben werden. Per Spezifikation ist keine gesonderte Fehlerbehandlung vorgesehen.

Die Funktion TimeSpanToString soll die Zeitdarstellung einer TimeSpan Instanz in einen String mit der Syntax 23:59:59 umwandeln. Hierbei wird nicht geprüft ob die TimeSpan Instanz einen gültigen Zeitwert beinhaltet. Es werden also auch ungültige Zeitwerte auf der Konsole ausgebenen.

Die bereits implementierte Funktion SecondsToTimeSpan soll nunmehr als Übergabeparameter den Datentyp LONGINT und nicht mehr eine TimeSpan Instanz verwenden, da dies logischer für einen Aufrufer sein sollte.

1.2 Source

Folgend ist der Source der TimeSpanUnit und seiner Tests aufgeführt.

1.2.1 TimeSpanUnit

Folgend ist der Source der TimeSpanUnit angeführt.

```
{
  This unit hanldes time spans and is able to convert them into second representation
  and visa versa.
}
UNIT TimeSpanUnit;

{ ######################### Interface part ######################### }
INTERFACE
  TYPE
    {
      Record which holds the information of the time span representations and a eventually
occurred error message as well.
    }
    TimeSpan = RECORD
        hour: INTEGER;
        minute: INTEGER;
        second: INTEGER;
        timeInSeconds: LONGINT;
        error: STRING;
    END;

  {
     This function is used to create a TimeSpan instance with the given values for the defined
attributes.
     error attribute will be set to an empty string which indicates that no error has occured.

     @param
        h: the hour to be set on the instance
     @param
        m: the minute to be set on the instance
     @param
        s: the second to be set on the instance
     @return
         the created TimeSpan instance
  }
  FUNCTION CreateTimeSpan(h, m, s: INTEGER): TimeSpan;

  {
   Function which converts the given seconds to hour, minute and second values,
   which will be set on the returned TimeSpan instance.
   If an error occured thenthe error attribute will not be an empty string, but will contain
the occured error description.

   @param
       seconds: the seconds to convert into a time spend returned by a TimeSpan instance
   @return
       the TimeSpan instance containing the calculated time representation of the set
'timeInSeconds' value,
       or an set error message if an error occured
  }
  FUNCTION SecondsToTimeSpan(seconds: LONGINT): TimeSpan;

  {
    Function which calculates the second representation of the given time 'hh:mm:ss' set on
the TimeSpan instance.
    If an error occurs then the error attribute will contain the the error description.

    @param
        span: the TimeSpan instance which holds the time representation.
    @return
        the TimeSpan instance with the set calculated 'timeInSeconds' value or the description
of the occured error
    @see ValidateRange(value, range: INTEGER): BOOLEAN;
  }
  FUNCTION TimeSpanToSeconds(span: TimeSpan): TimeSpan;

  {
```

```
    Function which calculates the time difference in seconds between the two given TimeSpan
    instances.

    @param
        before: the TimeSpan instance representing the lower time value
    @param
        after: the TimeSpan instance representing the higher time value
    @return
        calcResult: the TimeSpan instance which holds the calculated time difference, or the
set error
                    which would occur if 'before TimeSpan ' has a second value which is after
the 'after TimeSpan' second value.
  }
  FUNCTION TimeDifference(before, after:TimeSpan): TimeSpan;

  {
    Function which answers the question if the before TimeSpan instance has a lower
    time value than the after TimeSpan.

    @param
        span1: the before TimeSpan instance
    @param
        span2: the after TimeSpan instance
    @return
        true if the span1 time span instance has a lower time value the the span2
        TimeSpan instance, false otherwise. If a error occured this function will return
        false by default, because it has no way to populate the error to the caller.
  }
  FUNCTION IsShorterThan(span1, span2: TimeSpan):BOOLEAN;

  {
    Converts a TimeSpan isntance to its string representation.

    @param
        span: the TimeSpan instance which time span values shall be converted to a string
    @return
        the converted time span values of the given TimeSpan instance
  }
  FUNCTION TimeSpanToString(span: TimeSpan): STRING;

IMPLEMENTATION

TYPE
  {
    Enumeration which specifies the types of the time constants and is used to act as an array
index.
    These enumerations specify the time attributes which do have constants settings.
  }
  TimeConstantType = (HOURS_PER_DAY, MINUTES_PER_HOUR, SECONDS_PER_MINUTE);

  {
    The array type which specifies an array with an index of the type 'TimeConstantType'
    Holds every time constants used for the conversion actions.
  }
  TimeConstantArray = ARRAY [TimeConstantType] OF INTEGER;

  {
    Constants which define the time constants.
    The values are accessible via the 'TimeConstantType' enum values acting as the index of
this array type.
    These constants are used for the time conversion actions.
    ATTENTION: The order of the values must fit the 'TimeConstantType' enumeration
specification.
  }
  CONST
    timeConstant: TimeConstantArray = (24, 60, 60);


  { ######################### Private function and procedures ######################### }
  {
    Function which validates the range of the given 'value'.

    @param
        value: the value which range shall be validated
    @param
        bottom: the bottom border of the given 'value'
```

```
    @param
        top: the top border of the given 'value'
    @return
        true if the value is within the defined borders, false otherwise or if bottom > top
  }
  FUNCTION ValidateRange(value, bottom, top: INTEGER): BOOLEAN;
  BEGIN

      ValidateRange := false;

      IF (bottom <= top) AND (value >= bottom) AND (value <= top) THEN
        ValidateRange := true

  END;

  {
    Converts a integer to a string representation.
    Used to have a function whic returns the converted integer instead of a procedure.
    WriteLn('h:', IntToStr(1), '');

    @param value:
              the integer to be converted to a string
    @return
        the string representation of the given integer value
  }
  FUNCTION IntToStr(value: LONGINT): STRING;
  VAR
    s: STRING;
  BEGIN

    Str(value, s);
    IntToStr := s

  END;

  { ######################### Public procedure and functions ######################### }
  { Initializes the TimeSpan instance }
  FUNCTION CreateTimeSpan(h, m, s: INTEGER): TimeSpan;
  VAR
    span: TimeSpan;
  BEGIN

    span.hour := h;
    span.minute := m;
    span.second := s;
    span.timeInSeconds := -1;
    span.error := '';

    CreateTimeSpan := span;

  END;

  { Convert seconds to time span }
  FUNCTION SecondsToTimeSpan(seconds: LONGINT): TimeSpan;
  VAR
    result: TimeSpan;
  BEGIN
    result := CreateTimeSpan(-1, -1, -1);
    IF seconds >= 0 THEN BEGIN
      result.hour := seconds DIV (timeConstant[SECONDS_PER_MINUTE] *
timeConstant[MINUTES_PER_HOUR]);
      result.minute := (seconds - (result.hour * (timeConstant[SECONDS_PER_MINUTE] *
timeConstant[MINUTES_PER_HOUR]))) DIV timeConstant[MINUTES_PER_HOUR];
      result.second := (seconds - (result.hour * (timeConstant[SECONDS_PER_MINUTE] *
timeConstant[MINUTES_PER_HOUR])) - (result.minute * timeConstant[MINUTES_PER_HOUR]));
    END
    ELSE
     result.error := 'Invalid Seconds: [' + IntToStr(seconds) + ']';

    SecondsToTimeSpan := result;

  END;

  { converts time span to seconds }
  FUNCTION TimeSpanToSeconds(span: TimeSpan): TimeSpan;
  VAR
```

```
    message: STRING;
    error: BOOLEAN;
  BEGIN
    message := '';
    error := false;

    IF span.hour < 0 THEN BEGIN
      message := message + '[' + IntToStr(span.hour) + ']';
      error := true;
    END
    ELSE
      message := message + IntToStr(span.hour);
    IF NOT ValidateRange(span.minute, 0, (timeConstant[MINUTES_PER_HOUR] - 1)) THEN BEGIN
      message := message + ':[' + IntToStr(span.minute) + ']';
      error := true;
    END
    ELSE
      message := message + ':' + IntToStr(span.minute);
    IF NOT ValidateRange(span.second, 0, (timeConstant[SECONDS_PER_MINUTE] - 1)) THEN BEGIN
      message := message + ':[' + IntToStr(span.second) + ']';
      error := true;
    END
    ELSE
      message := message + ':' + IntToStr(span.second);

    IF NOT error THEN
      span.timeInSeconds := (span.hour * timeConstant[MINUTES_PER_HOUR] *
timeConstant[SECONDS_PER_MINUTE]) + (span.minute * timeConstant[SECONDS_PER_MINUTE]) +
span.second
    ELSE
      span.error := 'Invalid time span: ' + message;

    TimeSpanToSeconds := span;

  END;

  { Reimplement the seconds difference conversion }
  FUNCTION TimeDifference(before, after:TimeSpan): TimeSpan;
  VAR
    result: TimeSpan;
    message: STRING;
  BEGIN
    message := '';
    result := CreateTimeSpan(-1, -1, -1);
    before := TimeSpanToSeconds(before);
    after := TimeSpanToSeconds(after);

    { Handle detected error on time span }
    IF before.error <> '' THEN BEGIN
      message := 'Before: ' + before.error;
    END;
    IF after.error <> '' THEN BEGIN
      IF before.error <> '' THEN BEGIN
        message := message + ' | ';
      END;
      message := message + 'After: ' + after.error;
    END;

    { Handle overflow }
    IF after.timeInSeconds < before.timeInSeconds THEN BEGIN
      message := 'after overflows before time span'
    END;

    IF message = '' THEN BEGIN
      result.timeInSeconds := after.timeInSeconds - before.timeInSeconds;
    END
    ELSE BEGIN
      result.error := message;
    END;

    TimeDifference := result;
  END;

  { Check for shorter time span }
  FUNCTION IsShorterThan(span1, span2: TimeSpan):BOOLEAN;
  BEGIN
```

```
    span1 := TimeSpanToSeconds(span1);
    span2 := TimeSpanToSeconds(span2);

    IF (span1.error = '') AND (span2.error = '') AND (span1.timeInSeconds <
span2.timeInSeconds) THEN BEGIN
      IsShorterThan := true
    END
    ELSE
      IsShorterThan := false;
  END;

  { Convert to string }
  FUNCTION TimeSpanToString(span: TimeSpan): STRING;
  BEGIN
    TimeSpanToString := IntToStr(span.hour) + ':' + IntToStr(span.minute) + ':' +
IntToStr(span.second);
  END;
END.
```

1.2.2 TimeSpanUnitTest

Folgend ist der Source der Tests der TimeSpanUnit aufgeführt, welche die Funktionalitäten der TimeSpanUnit testen.

```
{
  Programm which test the TimeSpanUnit.
}
PROGRAM TimeSpanUnitTest;

USES TimeSpanUnit;

{
  Tests the TimeDifference function.

  @param
      before: the before TimeSpan instance
  @param
      after: the after TimeSpan instance
}
PROCEDURE TestTimeDifference(before, after: TimeSpan);
VAR
  result: TimeSpan;
BEGIN
  result := TimeDifference(before, after);
  WriteLn('before:          ', TimeSpanToString(before));
  WriteLn('after:           ', TimeSpanToString(after));
  WriteLn('result (h:m:s): ', TimeSpanToString(result));
  WriteLn('result (sec):    ', result.timeInSeconds);
  WriteLn('error:  ', result.error);
  WriteLn('');
END;

{
  Test the TimeSpanToSeconds function.

  @param
      span: the TimeSpan instance
}
PROCEDURE TestTimeSpanToSeconds(span: TimeSpan);
VAR
  result: TimeSpan;
BEGIN
  result := TimeSpanToSeconds(span);
  WriteLn('time span:     ', TimeSpanToString(result));
  WriteLn('result (sec): ', result.timeInSeconds);
  WriteLn('error:         ', result.error);
  WriteLn('');
END;

{
  Test the SecondsToTimeSpan function.

  @param
      seconds: the seconds to convert
}
```

```
PROCEDURE TestSecondsToTimeSpan(seconds: LONGINT);
VAR
  result: TimeSpan;
BEGIN
  result := SecondsToTimeSpan(seconds);
  WriteLn('seconds:   ', seconds);
  WriteLn('time span: ', TimeSpanToString(result));
  WriteLn('error:     ', result.error);
  WriteLn('');
END;

{
  Test the IsShorterThan function.

  @param
      before: the before TimeSpan instance
  @param
      after: the after TimeSpan instance
}
PROCEDURE TestIsShorterThan(before, after: TimeSpan);
BEGIN
  WriteLn('before: ', TimeSpanToString(before));
  WriteLn('after:  ', TimeSpanToString(after));
  WriteLn('result: ', IsShorterThan(before, after));
  WriteLn('');
END;

BEGIN
  { Test convert span to seconds }
  WriteLn('################ TimeSpanToSeconds ################');
  TestTimeSpanToSeconds(CreateTimeSpan(-1,-1,-1));
  TestTimeSpanToSeconds(CreateTimeSpan(1,60,60));
  TestTimeSpanToSeconds(CreateTimeSpan(1,59,60));
  TestTimeSpanToSeconds(CreateTimeSpan(1,1,1));

  { Test convert seconds to time span }
  WriteLn('################ SecondsToTimeSpan ################');
  TestSecondsToTimeSpan(-1);
  TestSecondsToTimeSpan(0);
  TestSecondsToTimeSpan(1);
  TestSecondsToTimeSpan(3600);
  TestSecondsToTimeSpan(3601);
  TestSecondsToTimeSpan(3661);

  { Test get time difference }
  WriteLn('################ TimeDifference ################');
  TestTimeDifference(CreateTimeSpan(-1, -1, -1), CreateTimeSpan(-1, -1, -1));
  TestTimeDifference(CreateTimeSpan(0, -1, -1), CreateTimeSpan(0, -1, -1));
  TestTimeDifference(CreateTimeSpan(0, 0, -1), CreateTimeSpan(0, 0, -1));
  TestTimeDifference(CreateTimeSpan(0, 60, -1), CreateTimeSpan(0, 60, -1));
  TestTimeDifference(CreateTimeSpan(0, 0, 2), CreateTimeSpan(0, 0, 1));
  TestTimeDifference(CreateTimeSpan(0, 59, 1), CreateTimeSpan(0, 59, 1));
  TestTimeDifference(CreateTimeSpan(10, 10, 10), CreateTimeSpan(10, 10, 10));
  TestTimeDifference(CreateTimeSpan(9, 9, 9), CreateTimeSpan(10, 10, 10));


  { Test is shorter than }
  WriteLn('################ IsShorterThan ################');
  TestIsShorterThan(CreateTimeSpan(10, 10, -1), CreateTimeSpan(10, 10, 9));
  TestIsShorterThan(CreateTimeSpan(10, 10, 9), CreateTimeSpan(10, 10, -1));
  TestIsShorterThan(CreateTimeSpan(10, 10, 10), CreateTimeSpan(10, 10, 10));
  TestIsShorterThan(CreateTimeSpan(10, 10, 9), CreateTimeSpan(10, 10, 10));

  { Test convert time span to string }
  WriteLn('################ TImeSpanToString ################');
  WriteLn(TimeSpanToString(CreateTimeSpan(10, 10, 10)));
  WriteLn(TimeSpanToString(CreateTimeSpan(23, 59, 59)));
  WriteLn(TimeSpanToString(CreateTimeSpan(23, 60, -1)));
  WriteLn('');

END.
```

1.3 Tests

Hier sind die Tests der TimeSpanUnit beschrieben. Diese sollen sicherstellen, dass sich die TimeSpan Unit so verhält wie spezifiziert.

1.3.1 TimeSpanToSeconds

Dieser Test testen die Funktion TimeSpanToSeconds.

```
################## TimeSpanToSeconds ##################
time span:    -1:-1:-1
result (sec): 0
error:        Invalid time span: [-1]:[-1]:[-1]

time span:    1:60:60
result (sec): 0
error:        Invalid time span: 1:[60]:[60]

time span:    1:59:60
result (sec): 0
error:        Invalid time span: 1:59:[60]

time span:    1:1:1
result (sec): 3661
error:
```

Es ist zu sehen das sich diese Funktion wie erwartet verhält, einerseits bei einem Fehlerfall und andererseits bei korrekter Werteangabe. Bei einem Fehler wird der Sekundenwert auf 0 belassen.

1.3.2 SecondsToTimeSpan

Dieser Test testet die Funktion SecondsToTimeSpan.

```
################## SecondsToTimeSpan ##################
seconds:   -1
time span: 0:0:0
error:     Invalid Seconds: [-1]

seconds:   0
time span: 0:0:0
error:

seconds:   1
time span: 0:0:1
error:

seconds:   3600
time span: 1:0:0
error:

seconds:   3601
time span: 1:0:1
error:

seconds:   3661
time span: 1:1:1
error:
```

Auch diese Funktion funktioniert auch weiterhin trotz der Änderung am Übergabeparameter. Bei einem Fehler wir die Art des Fehlers auf der TimeSpan Instanz gesetzt. Bei einem Fehlerfall werden alle Werte der TImeSpan Instanz auf 0 belassen.

1.3.3 TimeDifference

Dieser Test testet die Funktion TimeDifference.

```
################# TimeDifference #################
before:          -1:-1:-1
after:           -1:-1:-1
result (h:m:s): 0:0:0
result (sec):    0
error:  Before: Invalid time span: [-1]:[-1]:[-1] ¦ After: Invalid time span: [-
1]:[-1]:[-1]

before:          0:-1:-1
after:           0:-1:-1
result (h:m:s): 0:0:0
result (sec):    0
error:  Before: Invalid time span: 0:[-1]:[-1] ¦ After: Invalid time span: 0:[-1
]:[-1]

before:          0:0:-1
after:           0:0:-1
result (h:m:s): 0:0:0
result (sec):    0
error:  Before: Invalid time span: 0:0:[-1] ¦ After: Invalid time span: 0:0:[-1]


before:          0:60:-1
after:           0:60:-1
result (h:m:s): 0:0:0
result (sec):    0
error:  Before: Invalid time span: 0:[60]:[-1] ¦ After: Invalid time span: 0:[60
]:[-1]

before:          0:0:2
after:           0:0:1
result (h:m:s): 0:0:0
result (sec):    0
error:  after overflows before time span

before:          0:59:1
after:           0:59:1
result (h:m:s): 0:0:0
result (sec):    0
error:

before:          10:10:10
after:           10:10:10
result (h:m:s): 0:0:0
result (sec):    0
error:

before:          9:9:9
after:           10:10:10
result (h:m:s): 1:1:1
result (sec):    3661
error:
```

Diese Funktion funktioniert wie erwartet. Bei einem Fehler wird die Art des Fehlers auf dem error Field der TimeSpan Instanz gesetzt. Alle Zeitwerte und der Sekundenwert werden auch hier bei 0 belassen.

1.3.4 IsShorterThan

Dieser Test testet die neu hinzugefügte Funktion IsShorterThan.

```
################# IsShorterThan #################
before: 10:10:-1
after:  10:10:9
result: FALSE

before: 10:10:9
after:  10:10:-1
result: FALSE

before: 10:10:10
after:  10:10:10
result: FALSE

before: 10:10:9
after:  10:10:10
result: TRUE
```

Diese Funktion verhält sich wie spezifiziert. Bei einem Fehlerfall wird FALSE als Resultat zurückgegeben, sowie wenn before kleiner gleich after TimeSpan Instanz ist, oder wenn eine der beiden TimeSpan Instanzen ungültige Werte enthält.

1.3.5 TImeSpanToString

Dieser Test testet die Funktion TimeSpanToString, welche eine Zeitrepräsentation einer TimeSpan Instanz in einen String übersetzt.

```
################## TImeSpanToString ##################
10:10:10
23:59:59
23:60:-1
```

Dieser Test zeigt, dass diese Funktion einen String aus den gesetzten Zeitwerten erstellt, ohne Rücksichtnahme auf etwaige Fehler in der Zeitdarstellung.

2 WorkManagementUnit

Folgend ist die Dokumentation der Aufgabe 2 der Übung 5 beschrieben, die die Implementierung einer Unit für ein Work Management verlangt. Hierbei sollen Arbeitseinträge für Personen und Task mit einer Zeitangabe persistent gehalten werden.

2.1 Lösungsansatz

Es soll eine Unit für ein Work Management implementiert werden. Als Gedächtnis soll ein Array des verwendeten Datentyps verwendet werden, welches maximal 1000 aber minimal 100 Einträge erfassen soll. Die Größe der speicherbaren Einträge soll vom Benutzer definiert werden können, wobei das Gedächtnis als Standard mit 100 initialisiert werden soll. Es soll eine Funktion implementiert werden, mit der das Gedächtnis zurückgesetzt werden kann. Hierbei sollen alle Elemente des Arrays mit Standardwerten initialisiert werden. Des Weiteren soll der letzte Index des zuletzt eingefügten Elements gehalten werden, damit das nächste Element angehängt werden kann. Es soll auch ein Boolean Flag verwendet werden, welches anzeigt ob Einträge im Gedächtnis vorhanden sind oder nicht. Für die Arbeitseinträge sollen über einen eigenen Datentyp spezifiziert werden, welcher den Namen der Person, die Bezeichnung des Task und einen TimeSpan (Arbeitsdauer) als Attribute definiert. Für etwaige Fehler die auftreten können wie:

1. Keine Einträge mehr erfassbar
2. Ungültige Zeitangabe (Min: 1 Minute, Max: 8 Stunden)
3. Ungültige Zeitsyntax

soll eine Enumeration spezifiziert werden, welche als Resultat bei den implementierten Prozeduren zurückgegeben wird, um den Aufrufer über aufgetretene Fehler zu informieren.
Die Prozeduren sollen entsprechend der Spezifikation implementiert werden, wobei bei den Iterationen über die Einträge nur solange iteriert werden soll bis zum letzten Eintrag, da es sich hier ja um einen statischen Speicher und keinen dynamischen Speicher handelt. Auch soll nur über die Einträge iteriert werden, wenn überhaupt Einträge vorhanden sind.
Des Weiteren soll eine Funktion implementiert werden, welche einen Eintrag gemäß den gegebenen Parameter initialisiert, sodass der Aufrufer nicht die Attribute des Datentyps WorkEntry setzen muss, sondern sich über eine Funktion die Instanz erstellen lassen kann, Wobei er gezwungen ist die benötigten Werte anzugeben.

Für jede weitere Dokumentation sei auf den Source verwiesen.

2.2 Source

Folgend ist der Source der WorkManagenemtUnit sowie seiner Tests angeführt.

2.2.1 WorkManagementUnit

Folgend ist der Source der WorkManagementUnit angeführt.

```
{
  Unit for handling work entries which are hold by a entry holder.
  Call Reset to reset or reinitialize the memeory which are representing the
  work entry holder.
}
UNIT WorkManagementUnit;

{ ######################### Interface part ######################### }
INTERFACE

{ Uses the time span unit }
USES TImeSpanUnit;

TYPE
  { Spezifies the supported range }
  SupportedRange = 100..1000;

  { The error codes which vcan be handled by the caller }
  ErrorCode = (NONE, OVERFLOW, TO_SHORT, TO_LONG, INVALID_SPAN);

  { Compound representing a work entry }
  WorkEntry = RECORD
    name: STRING;
    task: STRING;
    spendTime: TimeSpan;
  END;

{
  Initializes the holder for the workflow entries with the given size.

  @param
      size: the size of the work entry holder (100 - 1000)
}
PROCEDURE Reset(size: SupportedRange);

{
  Creates a work entry for thegiven data.

  @param
      name: the name of the employee
  @param
      task: the name of the task the employee worked on
  @param
      spendTime: the TimeSpan instance which represents the time the employee hs worked
      on the defined task
  @return
      the create workEntry instance
}
FUNCTION CreateWorkEntry(name, task: STRING; spendTime: TimeSpan): WorkEntry;

{
  Adds a WorkEntry to the backed entry holder.

  @param
      entry: the entry to be added
  @param
      error: the error parameter which will be set <> NONE if an error occurs
  @returns
      the occured error or the given error from the calles which should be NONE
}
PROCEDURE AddWorkEntry(entry: WorkEntry; VAR error: ErrorCode);

{
  Gets the total work time for the given employee.
  @param
      name: the name of the employee
  @param
```

```
      spendtime: the TimeSpan instance given by the caller which will get set with the total
work time,
      should be initialized by the caller with 0:0:0
  @return
      the TimeSpan representing the total work time, or all values set to 0 when the employee
would not be found,
      or no entry exists
}
PROCEDURE GetTotalWorkTimeForPerson(name: STRING; VAR spendTime: TimeSpan);

{
  Gets the average time worked on a task.

  @param
      task: the task to get the average work time for
  @param
      average: the TimeSpan instance where the result will be saved, shall be initialized by
the
      caller with 0:0:0
  @return
      the TimeSpan instance which contains the average time worked on the task, or all field
set to 0,
      if the taks ccould not be found, because there can no task be present in the entry
holder which has
      a time spend less than 1 minute.
}
PROCEDURE GetAverageWorkTimeForTask(task: STRING; VAR average: TimeSpan);

{
  Prints the persons which has been working ont the task, along with their time spend.

  @param
      task: the task to be printed.
}
PROCEDURE PrintPersonForTask(task: STRING);

{
  Prints the work summary for the given person.

  @param
      name: the namee of the persosn to seachr for the work summaray
}
PROCEDURE PrintWorkSummaryForPerson(name: STRING);

{ ######################## Implementation part ######################## }
IMPLEMENTATION

TYPE
  { The specified range of the backed array which holds the entries }
  ArrayRange = 1..1000;
VAR
  { the memory of this unit which holds the entries }
  workEntryArray: ARRAY[ArrayRange] OF WorkEntry;
  { the last index where the last added entry resides }
  lastIdx: ArrayRange;
  { the from the user maximum set memory size}
  maxIdx: ArrayRange;
  { Flag which indicates that the memeory has no entries set }
  empty: BOOLEAN;

{ Reset memory }
PROCEDURE Reset(size: SupportedRange);
VAR
  i: ArrayRange;
BEGIN
  maxIdx := size;
  lastIdx := Low(ArrayRange);
  empty := true;

  { Sets the array elements with default isntances }
  FOR i:= Low(workEntryArray) TO High(workEntryArray) DO BEGIN
    workEntryArray[i] := CreateWorkEntry('', '', TimeSpanUnit.CreateTimeSpan(0, 0, 0));
  END;
END;

{ Creates a WorkEntry instance }
```

```
FUNCTION CreateWorkEntry(name, task: STRING; spendTime: TimeSpan): WorkEntry;
BEGIN
  CreateWorkEntry.name := name;
  CreateWorkEntry.task := task;
  CreateWorkEntry.spendTime := spendTime;
END;

{ Adds the work entry to the memory}
PROCEDURE AddWorkEntry(entry: WorkEntry; VAR error: ErrorCode);
BEGIN
  { Converts the time span to seonds, and will contain any error }
  entry.spendTime := TimeSpanToSeconds(entry.spendTime);
  { Invalid time span detected }
  IF (entry.spendTime.error <> '') THEN BEGIN
    error := INVALID_SPAN
  END
  { To less time span }
  ELSE IF (entry.spendTime.timeInSeconds < 60) THEN BEGIN
    error := TO_SHORT
  END
  { To much time }
  ELSE IF (entry.spendTime.timeInSeconds > (8 * 60 * 60)) THEN BEGIN
    error := TO_LONG
  END
  { Handle no space left }
  ELSE IF (lastIdx = maxIdx) THEN BEGIN
      error := OVERFLOW;
  END
  { valid to add entry }
  ELSE BEGIN
    IF empty THEN BEGIN
      empty := false
    END
    { Only increase after first elements has been added }
    ELSE BEGIN
      Inc(lastIdx)
    END;
    workEntryArray[lastIdx] := entry;
  END;
END;

{ Gets total work time }
PROCEDURE GetTotalWorkTimeForPerson(name: STRING; VAR spendTime: TimeSpan);
VAR
  i: ArrayRange;
  sec: LONGINT;
  entry: WorkEntry;
BEGIN
  sec := 0;
  { Only get total time when entries are present }
  IF NOT empty THEN BEGIN
    FOR i := Low(workEntryArray) TO lastIdx DO BEGIN
      entry := workEntryArray[i];
      { Build sum for the given name }
      IF (entry.name = name) THEN BEGIN
        sec := sec + TimeSpanUnit.TimeSpanToSeconds(entry.spendTime).timeInSeconds;
      END;
    END;
  END;

  spendTime := TimeSpanUnit.SecondsToTimeSpan(sec);
END;

{ Gets average of spend time }
PROCEDURE GetAverageWorkTimeForTask(task: STRING; VAR average: TimeSpan);
VAR
  i: ArrayRange;
  name: STRING;
  sec: LONGINT;
  count: INTEGER;
  entry: WorkEntry;
BEGIN
  sec := 0;
  count := 0;
  average.timeInSeconds := 0;
  name := '';
```

```
  { Only get average time when entries are present }
  IF NOT empty THEN BEGIN
    FOR i := Low(workEntryArray) TO lastIdx DO BEGIN
      entry := workEntryArray[i];
      { Search for this task }
      IF (entry.task = task) THEN BEGIN
        sec := sec + TimeSpanUnit.TimeSpanToSeconds(entry.spendTime).timeInSeconds;
        Inc(count);
      END;
    END;
  END;

  { Calculate the average if the task on working persons have been found }
  IF (sec <> 0) THEN BEGIN
    sec := (sec DIV count);
  END;

  { Create time span out of seconds }
  average := TimeSpanUnit.SecondsToTimeSpan(sec);
END;

{ Print persons for task to table }
PROCEDURE PrintPersonForTask(task: STRING);
VAR
  i: ArrayRange;
  found: BOOLEAN;
  count: INTEGER;
  entry: WorkEntry;
  name: STRING;
BEGIN
  found := false;
  count := 0;
  name := '';

  { Only print result if there are entries available }
  WriteLn('################### Persons working on task ###################');
  WriteLn('Task    : ', task);
  WriteLn('Persons : ');
  IF NOT empty THEN BEGIN
    FOR i := Low(workEntryArray) TO lastIdx DO BEGIN
      entry := workEntryArray[i];
      { Search for this task }
      IF (entry.task = task) THEN BEGIN
        IF NOT found THEN BEGIN
          found := true;
        END;
        IF (entry.name <> name) THEN BEGIN
          name := entry.name;
          Inc(count);
          WriteLn('          ', count, ': ', name);
        END;
      END;
    END;
  END;
  { Either no entries or task not found }
  IF NOT FOUND THEN BEGIN
    WriteLn('          No work entries found for the task !!!');
  END;
  WriteLn('################### Persons working on task ###################');
END;

{ Print work summary for a person }
PROCEDURE PrintWorkSummaryForPerson(name: STRING);
VAR
  i: ArrayRange;
  found: BOOLEAN;
  entry: WorkEntry;
BEGIN
  found := false;

  { Only print result if entries are avialable }
  WriteLn('################### Work summary for person ###################');
  WriteLn('Person : ', name);
  IF NOT empty THEN BEGIN
    FOR i := Low(workEntryArray) TO lastIdx DO BEGIN
```

```
      entry := workEntryArray[i];
      { Search for this person }
      IF (entry.name = name) THEN BEGIN
        WriteLn('         Task: ', entry.task:15, ' | Spend time: ',
TimeSpanUnit.TimeSpanToString(entry.spendTime));
        found := true;
      END;
    END;
  END;
  { Either no entries or person not found }
  IF NOT found THEN BEGIN
    WriteLn('         No work entries found for the person !!!':10);
  END;
  WriteLn('################### Work summary for person ###################');
END;

BEGIN
  Reset(100);
END.
```

2.2.2 WorkManagementUnitTest

Folgend ist der Source der Tests der WorkManagemenUnit angeführt.

```
PROGRAM WorkManagementUnitTest;

USES WorkManagementUnit, TimeSpanUnit;

{
  Adds a entry and prints the added entry to the console
}
PROCEDURE AddEntry(name, task: STRING; span: TimeSpan);
VAR
  entry: WorkEntry;
  error: ErrorCode;
BEGIN
  error := NONE;
  entry := WorkManagementUnit.CreateWorkEntry(name, task, span);
  WorkManagementUnit.AddWorkEntry(entry, error);
  WriteLn('Error: ', error:10, 'Name: ', entry.name, ' | Task: ', entry.task, ' | Time: ',
TimeSpanUnit.TimeSpanToString(entry.spendtime));
END;

{
  Tests GetTotalWorkTimeForPerson
}
PROCEDURE TestGetTotalWorkTimeForPerson;
VAR
  span: TimeSpan;
BEGIN
  WriteLn('---------------------- TestGetTotalWorkTimeForPerson ----------------------');
  WorkManagementUnit.Reset(100);
  WriteLn('No entries are present: ');
  WorkManagementUnit.GetTotalWorkTimeForPerson('Thomas', span);
  WriteLn('Total work time for person: ', TimeSpanUnit.TimeSpanToString(span));
  WriteLn('');

  WorkManagementUnit.Reset(100);
  AddEntry('Thomas' , 'Impl', TimeSpanUnit.CreateTimeSpan(0, 1, 0));
  AddEntry('Thomas' , 'Spec', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Thomas' , 'Test', TimeSpanUnit.CreateTimeSpan(1, 0, 0));
  AddEntry('Hannes' , 'Impl', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Hannes' , 'Spec', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Hannes' , 'Test', TimeSpanUnit.CreateTimeSpan(2, 2, 0));
  WorkManagementUnit.GetTotalWorkTimeForPerson('Thomas', span);
  WriteLn('');
  WriteLn('Total work time for Thomas: ', TimeSpanUnit.TimeSpanToString(span));
  WorkManagementUnit.GetTotalWorkTimeForPerson('Hannes', span);
  WriteLn('Total work time for Hannes: ', TimeSpanUnit.TimeSpanToString(span));
  WriteLn('---------------------- TestGetTotalWorkTimeForPerson ----------------------');
  WriteLn('');
END;

{
  Tests the AddWorkEntry
}
```

```
PROCEDURE TestAddWorkEntry;
VAR
  i: INTEGER;
  entry: WorkEntry;
  error: ErrorCode;
BEGIN
  WriteLn('---------------------- TestAddWorkEntry ----------------------');
  WorkManagementUnit.Reset(100);
  AddEntry('Thomas', 'Impl', TimeSpanUnit.CreateTimeSpan(8, 0, 1));
  AddEntry('Thomas', 'Impl',  TimeSpanUnit.CreateTimeSpan(0, 0, 59));

  FOR i := 1 TO 101 DO BEGIN
    Write(i:3, '. ');
    AddEntry('Hannes' , 'Specification', TimeSpanUnit.CreateTimeSpan(0, 0, 59));
  END;
  WriteLn('---------------------- TestAddWorkEntry ----------------------');
  WriteLn('');
END;

{
  Tests the GetAverageWorkTimeForTask
}
PROCEDURE TestGetAverageWorkTimeForTask;
VAR
  span: TImeSpan;
BEGIN
  WriteLn('---------------------- TestGetAverageWorkTimeForTask ----------------------');
  WorkManagementUnit.Reset(100);
  WriteLn('No entries are present: ');
  WorkManagementUnit.GetAverageWorkTimeForTask('Doku', span);
  WriteLn('AverageTime for Doku: ', TimeSpanUnit.TimeSpanToString(span));
  WriteLn('');

  WorkManagementUnit.Reset(100);
  AddEntry('Thomas' , 'Impl', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Thomas' , 'Impl', TimeSpanUnit.CreateTimeSpan(0, 1, 0));
  AddEntry('Hannes' , 'Spec', TimeSpanUnit.CreateTimeSpan(1, 0, 0));
  AddEntry('Hannes' , 'Spec', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  WriteLn('');
  span := TimeSpanUnit.CreateTimeSpan(0, 0, 0);
  WorkManagementUnit.GetAverageWorkTimeForTask('Impl', span);
  WriteLn('Average time for Impl: ', TimeSpanUnit.TimeSpanToString(span));
  span := TimeSpanUnit.createTimeSpan(0, 0, 0);
  WorkManagementUnit.GetAverageWorkTimeForTask('Spec', span);
  WriteLn('Average time for Spec: ', TimeSpanUnit.TimeSpanToString(span));
  WriteLn('---------------------- TestGetAverageWorkTimeForTask ----------------------');
  WriteLn('');
END;

PROCEDURE TestPrintPersonForTask;
BEGIN
  WorkManagementUnit.Reset(100);
  WriteLn('No entries are present: ');
  WorkManagementUnit.Reset(100);
  WorkManagementUnit.PrintPersonForTask('Implementation');
  WriteLn('');
  WriteLn('---------------------- TestPrintPersonForTask ----------------------');
  AddEntry('Maria' , 'Implementation', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Thomas' , 'Implementation', TimeSpanUnit.CreateTimeSpan(0, 1, 0));
  AddEntry('Maria' , 'Specification', TimeSpanUnit.CreateTimeSpan(1, 0, 0));
  AddEntry('Thomas' , 'Specification', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Hannes' , 'Implementation', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Hannes' , 'Implementation', TimeSpanUnit.CreateTimeSpan(0, 1, 0));
  AddEntry('Hannes' , 'Specification', TimeSpanUnit.CreateTimeSpan(1, 1, 1));
  AddEntry('Hannes' , 'Specification', TimeSpanUnit.CreateTimeSpan(0, 1, 1));
  WriteLn('');
  WorkManagementUnit.PrintPersonForTask('Implementation');
  WriteLn('---------------------- TestPrintPersonForTask ----------------------');
  WriteLn('');
END;

{
  Test the procedure PrintWorkSummaryForPerson.
}
PROCEDURE TestPrintWorkSummaryForPerson;
BEGIN
```

```
  WorkManagementUnit.Reset(100);
  WriteLn('---------------------- TestPrintWorkSummaryForPerson ----------------------');
  WriteLn('No entries are present');
  WorkManagementUnit.PrintWorkSummaryForPerson('Thomas');
  WriteLn('');
  AddEntry('Thomas' , 'Implementation', TimeSpanUnit.CreateTimeSpan(0, 2, 2));
  AddEntry('Thomas' , 'Implementation', TimeSpanUnit.CreateTimeSpan(0, 1, 0));
  AddEntry('Thomas' , 'Specification', TimeSpanUnit.CreateTimeSpan(1, 1, 1));
  AddEntry('Thomas' , 'Test', TimeSpanUnit.CreateTimeSpan(2, 2, 2));
  AddEntry('Hannes' , 'Implementation', TimeSpanUnit.CreateTimeSpan(0, 2, 0));
  AddEntry('Hannes' , 'Test', TimeSpanUnit.CreateTimeSpan(0, 1, 2));
  AddEntry('Hannes' , 'Specification', TimeSpanUnit.CreateTimeSpan(1, 1, 1));
  AddEntry('Hannes' , 'Specification', TimeSpanUnit.CreateTimeSpan(0, 1, 1));

  WorkManagementUnit.PrintWorkSummaryForPerson('Thomas');
  WriteLn('');
  WorkManagementUnit.PrintWorkSummaryForPerson('Hannes');
  WriteLn('---------------------- TestPrintWorkSummaryForPerson ----------------------');
  WriteLn('');
END;

BEGIN
  { Test AddWorkEntry }
  TestAddWorkEntry;
  { TestGetTotalWorkTimeForPerson }
  TestGetTotalWorkTimeForPerson;
  { Tests GetAverageWorkTimeForTask }
  TestGetAverageWorkTimeForTask;
  { Test for PrintWorkSummaryForPerson }
  TestPrintWorkSummaryForPerson;
  { Test PrintPersonForTimeTask }
  TestPrintPersonForTask;
END.
```

2.3 Tests

Folgend sind die Tests für die WorkManagementUnit angeführt.

2.3.1 AddWorkEntry

Die folgenden Tests testen die Prozedur AddWorkEntry.

Ungültige Zeitangaben:

Dieser Test testet das Verhalten wenn versucht wird einen Eintrag zu speichern, der ungültige Zeitangaben definiert.

```
Error: TO_LONG   Name: Thomas | Task: Impl | Time: 8:0:1
Error: TO_SHORT  Name: Thomas | Task: Impl | Time: 0:0:59
```

Zeitangaben unter einer Minute und über acht Stunden werden nicht gespeichert, und es wird eine entsprechende Enumeration als Fehlerindikator zurückgegeben.

Speicher voll:

Dieser Test testet das Verhalten wenn der zur Verfügung gestellte Speicher, der WorkManagementUnit voll ist und keine Einträge mehr aufgenommen werden können. Aus Platzgründen werden nur die Letzten Einträge angezeigt. (Siehe Source für Funktionalität)

```
 80. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 81. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 82. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 83. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 84. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 85. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 86. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 87. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 88. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 89. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 90. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 91. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 92. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 93. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 94. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 95. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 96. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 97. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 98. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
 99. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
100. Error: NONE      Name: Hannes | Task: Specification | Time: 7:59:20
101. Error: OVERFLOW  Name: Hannes | Task: Specification | Time: 7:59:20
```

Bei der Standardinitialisierung mit hundert Einträgen ist zu sehen, dass ab den 101 Eintrag ein Error zurückgegeben wird, da der zur Verfügung gestellte Speicher bereits aufgebraucht ist. Es werden also keine Einträge mehr aufgenommen.

2.3.2 GetTotalWorkTimeForPerson

Dies Tests testen die Prozedur GetTotalWorkTimeForPerson.

Keine Einträge:

Dieser Test testet das Verhalten der Prozedur wenn keine Einträge vorhanden sind.

```
No entries are present:
Total work time for person: 0:0:0
```

In diesem Fall wird eine TimeSpan Instanz zurückgegeben mit allen Werten auf 0 gesetzt. Null ist nicht möglich das es sich hier um keinen Pointer handelt.

Einträge vorhanden:

Dieser Test testet das Verhalten der Prozedur wenn es bereits Einträge für verschiedene Task und Personen gibt.

```
Error: NONE       Name: Thomas  | Task: Impl  | Time: 0:1:0
Error: NONE       Name: Thomas  | Task: Spec  | Time: 0:2:0
Error: NONE       Name: Thomas  | Task: Test  | Time: 1:0:0
Error: NONE       Name: Hannes  | Task: Impl  | Time: 0:2:0
Error: NONE       Name: Hannes  | Task: Spec  | Time: 0:2:0
Error: NONE       Name: Hannes  | Task: Test  | Time: 2:2:0

Total work time for Thomas: 1:3:0
Total work time for Hannes: 2:6:0
```

Hierbei ist zu erkennen, dass nur die Summen der jeweilig gewählten Personen ausgegeben werden.

2.3.3 GetAverageWorkTimeForTask

Die folgenden Tests testen die Prozedur GetAverageWorkTimeForTask.

Keine Einträge vorhanden:

Dieser Test testet das Verhalten der Prozedur wenn keine Einträge vorhanden sind.

```
No entries are present:
AverageTime for Doku: 0:0:0
```

Wenn keine Einträge vorhanden sind so wird das Ergebnis 0:0:0 ausgegeben werden, da es keine Einträge für einen Task geben kann, der kleiner als eine Minute oder größer als 8 Stunden sein kann. Null als Rückgabewert ist hierbei nicht möglich, da es sich hier nicht um einen Pointer handelt.

Mit Einträgen:

Dieser Test testet das Verhalten der Prozedur wenn es Einträge für zwei verschiedene Tasks gibt.

```
Error: NONE       Name: Thomas  | Task: Impl  | Time: 0:2:0
Error: NONE       Name: Thomas  | Task: Impl  | Time: 0:1:0
Error: NONE       Name: Hannes  | Task: Spec  | Time: 1:0:0
Error: NONE       Name: Hannes  | Task: Spec  | Time: 0:2:0

Average time for Impl: 0:1:30
Average time for Spec: 0:31:0
```

Es ist zu sehen, das die beiden Ergebnisse unabhängig voneinander ausgegeben werden, wobei immer nur der gewählte Task berücksichtigt wird und durch die Anzahl der Einträge dividiert wird, um die durchschnittliche Arbeitszeit zu ermitteln. Es ist anzumerken, dass hierbei eine ganzzahlige Division durchgeführt wird, wobei eine gewisse Ungenauigkeit in Kauf genommen wird.

2.3.4 PrintPersonsForTask

Diese Test testen das Verhalten der Prozedur PrintPersonsForTask.

Keine Einträge:

Dieser Test testet das Verhalten der Prozedur wenn keine Einträge vorhanden sind.

```
No entries are present:
#################### Persons working on task ####################
Task    : Implementation
Persons :
        No work entries found for the task !!!
#################### Persons working on task ####################
```

Wenn keine Einträge für den gegebenen Task vorhanden sind wird eine entsprechende Meldung ausgegeben, die anzeigt das es keine Einträge für diesen Task gibt.

Einträge vorhanden:

Dieser Test testet, das Verhalten der Prozedur wenn Einträge vorhanden sind.

```
---------------------- TestPrintPersonForTask -----------------------
Error: NONE       Name: Maria | Task: Implementation | Time: 0:2:0
Error: NONE       Name: Thomas | Task: Implementation | Time: 0:1:0
Error: NONE       Name: Maria | Task: Specification | Time: 1:0:0
Error: NONE       Name: Thomas | Task: Specification | Time: 0:2:0
Error: NONE       Name: Hannes | Task: Implementation | Time: 0:2:0
Error: NONE       Name: Hannes | Task: Implementation | Time: 0:1:0
Error: NONE       Name: Hannes | Task: Specification | Time: 1:1:1
Error: NONE       Name: Hannes | Task: Specification | Time: 0:1:1

##################### Persons working on task ####################
Task    : Implementation
Persons :
          1: Maria
          2: Thomas
          3: Hannes
##################### Persons_working on_task ####################
```

Es ist zu sehen, dass alle Personen, die an den Task arbeiten ausgegeben werden.

2.3.5 PrintWorkSummaryForPerson

Diese Tests testen die Prozedur PrintWorkSummaryForPerson.

Keine Einträge:

Dieser Tests testet das Verhalten der Prozedur wenn noch keine Einträge im System vorhanden sind.

```
No entries are present
##################### Work summary for person ####################
Person : Thomas
         No work entries found for the person !!!
##################### Work summary for person ####################
```

Wenn keine Einträge vorhanden sind, dann wird eine entsprechende Meldung ausgegeben, die anzeigt, dass keine Einträge für diese Person vorhanden sind.

Einträge vorhanden:

Dieser Test testet das Verhalten der Prozedur wenn Einträge für zwei verschiedene Personen im System vorhanden sind.

```
##################### Work summary for person ####################

Error: NONE       Name: Thomas | Task: Implementation | Time: 0:2:2
Error: NONE       Name: Thomas | Task: Implementation | Time: 0:1:0
Error: NONE       Name: Thomas | Task: Specification | Time: 1:1:1
Error: NONE       Name: Thomas | Task: Test | Time: 2:2:2
Error: NONE       Name: Hannes | Task: Implementation | Time: 0:2:0
Error: NONE       Name: Hannes | Task: Test | Time: 0:1:2
Error: NONE       Name: Hannes | Task: Specification | Time: 1:1:1
Error: NONE       Name: Hannes | Task: Specification | Time: 0:1:1
##################### Work summary for person ####################
Person : Thomas
         Task:  Implementation | Spend time: 0:2:2
         Task:  Implementation | Spend time: 0:1:0
         Task:   Specification | Spend time: 1:1:1
         Task:            Test | Spend time: 2:2:2
##################### Work summary for person ####################

##################### Work summary for person ####################
Person : Hannes
         Task:  Implementation | Spend time: 0:2:0
         Task:            Test | Spend time: 0:1:2
         Task:   Specification | Spend time: 1:1:1
         Task:   Specification | Spend time: 0:1:1
##################### Work summary for person ####################
```

Es ist zu sehen, dass jeweils nur die Einträge der gewählten Person angezeigt werden. Es werden alle gespeicherten Einträge der jeweiligen Person angeführt.