# Problem Set 3
# Course **Safe and Secure Software**
# (Winter Term 2016)

Bauhaus-Universität Weimar, Chair of Media Security
Prof. Dr. Stefan Lucks, Eik List
URL: `http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching/`

**Due Date:** 21 Nov 2016, 1:30 PM, via email to `eik.list(at)uni-weimar.de`.

**Goals:** Testing Ada code with testgen/AUnit and Ada2012 Pre- and Post-Conditions.

### Task 1 – Introduction (No Credits)

Read Chapters 9 - 13 of John English.

### Task 2 – Testing (No Credits)

If not already done, read up on the testing frameworks **`testgen`** and **`AUnit`**:

a) **testgen:** A small, easy-to-learn tool that allows to quickly write a battery of tests called *test driver* - for any Ada packages. Note that **`testgen`** depends on **`tg`** (by André Spiegel) which can also be found here. **`testgen`** can be found here.

b) **AUnit:** The standard testing framework for Ada; read and understand the implementation and the design of test cases, test suites, and test harnesses.

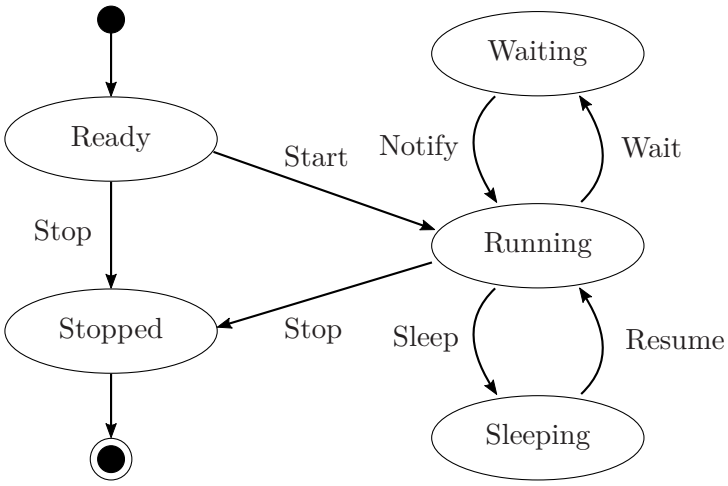
### Task 3 – Mini Project 1 – Testing Vectors (4 Credits)

Use **`testgen`** *and* **`AUnit`** to write a test suite for the Vectors package from the second problem set. Test at least all functions from the specification except for the output functions.

### Task 4 – Mini Project 2 – Elections (4 Credits)

Implement the **`Elections`** package. Add Ada 2012 annotations for pre- and post-conditions and write a test suite using *either* **`testgen`** *or* **`AUnit`**. *Note:* If the conditions get lengthy, you can add some helper function(s).

```
1  package Elections is
2      type Party is (None, A, B, C, D);
3      type Votes_Array is array(Party) of Natural;
4
5      Zero_Votes_Distribution: constant Votes_Array := (others => 0);
6      Votes_Distribution: Votes_Array := Zero_Votes_Distribution;
7      Num_Votes_Made: Natural := 0;
8      Num_Total_Voters: Natural := 0;
9
10     procedure Initialize(Num_Voters: Natural);
11     -- Resets the number of made votes and votes for all parties to 0, and
12     -- sets the number of total Voters to the given.
13     procedure Vote_For(Vote: Party);
14     function All_Voters_Voted return Boolean;
15     function Find_Winner return Party;
16     -- Returns Party with most votes assigned.
17     -- Returns None if multiple parties share the highest votes.
18 end Elections;
```




**Task 5 – Mini Project 3 – State Machine (4 Credits)**
Implement the following state machine that represents the states and transitions of a **`Thread`**. The figure represents the valid transitions between states. Write a test suite using *either* **`testgen`** *or* **`AUnit`**. Enrich the specification with sufficient Ada2012 pre- and post-conditions.

```
package Thread is
    type State is (None, Ready, Running, Stopped, Sleeping, Waiting);
    type Action is (Notify, Resume, Sleep, Start, Stop, Wait);

    procedure Initialize(S: out State);
    -- Sets S to Ready.
    procedure Do_Action(S: in out State; A: Action);
    -- Updates the state S according to the given input state S, and the
    -- given action A. Sets S to None if the transition is not defined.
end Thread;
```

**Task 6 – Mini Project 4 – Inheritance and White-Box Testing (6 Credits)**
The package **`Bank_Accounts`** below was slightly modified compared to the previous problem set. Derive another bank-account type in a package **`Bank_Accounts.Overdrawable`** which allows the account to be overdrawn up to a defined limit. Define a third package **`Bank_Accounts.Fees`** which charges a fee for every withdrawal and transfer. Implement test cases for each account type using *either* **`testgen`** *or* **`AUnit`**.

```
package Bank_Accounts is
    subtype Cents_Type is Integer;
    Default_Balance: constant Cents_Type := 0;

    type Account_Type is tagged limited private;

    Overspent_Exception: exception;
    Invalid_Amount_Exception: exception;

    function Get_Balance(Account: Account_Type) return Cents_Type;
    -- Returns the current Balance from Account.
    procedure Deposit(Account: in out Account_Type; Amount: Cents_Type);
    -- Deposits Amount at the given Account.
    procedure Withdraw(Account: in out Account_Type; Amount: Cents_Type);
    -- Withdraws Amount from the given Account.
```

```
16      procedure Transfer(From: in out Account_Type;
17                         To: in out Account_Type;
18                         Amount: in Cents_Type);
19      -- Transfers Amount from Account From to Account To.
20  private
21    type Account_Type is tagged limited record
22      Balance: Cents_Type;
23    end record;
24  end Bank_Accounts;
```