Stack

Dr. Matthias Hölzl

November 30, 2021

Goals

- Create a simple `Stack` data structure using TDD
- Proceed incrementally, baby steps
- This exercise is meant to demonstrate
  - how to develop features `incrementally` using TDD
  - how to write good tests that help you to grow your design
- Therefore: **Please do not look ahead in this document!**
  Only move to the next page after having finished the current exercise.
- (Also note that the stack implemented in this workshop has several pretty bad design flaws and should not be used for anything but demonstrating TDD.)

Stack

- Open the `Stack` project in your IDE/editor of choice
- Ensure that you can build the target `StackTest` and that the test fails when you execute it

- Solve simply!

Stack

- Implement a `Stack` data type for integers with the following signature:
- `void push(int element)`
- `int pop()`
- `bool is_empty()`
- If the stack is empty, `pop()` should throw an exception of type `std::out_of_range`

Extension (1)

- Extend the `Stack` data type with a method

  `std::size_t size()`

  that returns the number of elements on the stack

Extension (2)

- Extend the `Stack` data type with a member function

  `std::size_t count(int element)`

  that counts the occurrences of *element* on the stack

Extension (3)

- Extend the `Stack` data type with a member function

  `int pop_default(int default)`

  that acts like `pop()` when the stack is not empty and returns `default` when the stack is empty

Extension (4)

- Extend the `Stack` data type with member functions

  ```
  void set_default(int default)
  void clear_default()
  ```

  After `set_default(default)` has been called, `pop()` should act like `pop_default(default)`. When `clear_default()` is called, `pop()` should revert to its original behavior, i.e., throw an exception when the stack is empty

Extension (5)

- Our `Stack` class will now be used on an embedded system where dynamic memory allocation at runtime is not allowed. Therefore we have to replace its implementation with one that does not use `std::vector`.
  Modify your implementation so that it uses a `std::array<int, 16>` to store its elements. Make `push()` throw an exception of type `std::out_of_range`

  Hint: You will probably need a member variable to indicate how many elements are currently stored on the stack, e.g., you could use `std::size_t next_index` to indicate the index of the first unused element of the stack.