Professional Issues II
Unit 2: code commenting
Preparation for Lab 2

Markus Roggenbach

February 2020

Commenting the stack example

Variable declarations – uncommented

```
static int stackSize = 5;

static int topOfStack = -1;
static int[] stack = new int[stackSize];
static boolean errorFree = true;
```

Variable declarations – commented

```
static int stackSize = 5;
  /* maximal number of elements in the stack */

  /* data type stack is represented by
   - an array
   - a pointer to the top element
   - a flag concerning the history of the stack
  */

static int topOfStack = -1;
   /* pointer to the top element of the stack,
      -1 indicates the empty stack
   */
```

```
static int[] stack = new int[stackSize];
  /* array to store the stack elements,
     array elements with index > topOfStack
     don't belong to the stack
  */

static boolean errorFree = true;
  /* flag that indicates if the stack has
     always been used safely, i.e., without
       - calling top or pop on empty empty stack or
       - calling push on a full stack
     'true' indicates safe use
  */
```

The first three methods – uncommented

```
public static boolean isEmpty () {
    return topOfStack == -1;
}

public static boolean isFull () {
    return topOfStack == stackSize - 1;
}

public static void empty () {
    errorFree = true;
    topOfStack = -1;
}
```

The first three methods – commented

```
public static boolean isEmpty () {
    return topOfStack == -1;
}

public static boolean isFull () {
    return topOfStack == stackSize - 1;
}

public static void empty () {
    errorFree = true;
    topOfStack = -1;
}
```

Note

- no rule applies
- all these three methods are so simple that – any comment you might want to write – would be prevented by the rule "do not write trivial comments".

top uncommented

```
public static int top () {
    errorFree = ! (isEmpty ()) & errorFree;
    if (errorFree) {
        return stack[topOfStack];
    } else {
        return 0;
    }
}
```

top commented, first iteration

The rule on branching structures applies and asks for three comments:

```
public static int top () {
    errorFree = ! (isEmpty ()) & errorFree;
    /* conditionally return the top element */
    if (errorFree) {
        return stack[topOfStack]; /* return an element
                                     if there is one
                                     and it is safe */
    } else {
        return 0; /* indicate an error by returning 0 */
    }
}
```

top commented, 2nd and final iteration

The rule on avoiding trivial comments applies; deletes the first comment, simplifies the third comment:

```
public static int top () {
    errorFree = ! (isEmpty ()) & errorFree;
    if (errorFree) {
        return stack[topOfStack]; /* return an element
                                     if there is one
                                     and it is safe */
    } else {
        return 0; /* indicate an error */
    }
}
```

push commmented

```
public static void push (int value) {
    errorFree = ! (isFull ()) & errorFree;
    if (errorFree) { /* put the value on the stack
                        if it is safe to do so     */
        topOfStack = topOfStack + 1;
        stack[topOfStack] = value;
    }
}
```

pop commented

```
public static void pop () {
    errorFree = ! (isEmpty ()) & errorFree;
    if (errorFree) { /* remove an element of the stack
                        if it is safe to do so;
                        for efficiency the 'removed'
                        element remains in the array   */
        topOfStack = topOfStack - 1;
    }
}
```

Stacks & Queues

Stack

follows the LIFO principle:

- last in
- first out

Example:

```
push(17);
push(18);
top; \\ -- returns 18
pop;
top; \\ -- returns 17
```


Queue

follows the FIFO principle:

- first in
- first out

Example:

```
enqueue(17);
enqueue(18);
dequeue; \\ -- returns 17
dequeue; \\ -- returns 18
```

Used in: process queues of the operating system.

An implementation of Queues with arrays

- empty():

  [_,_,_,_,_] front=4, back=4, length=0

- enqueue(17)

  [17,_,_,_,_] front=4, back=0, length=1

- enqueue(18)

  [17,18,_,_,_] front=4, back=1, length=2

- dequeue() – returns 17

  [_,18,_,_,_] front=0, back=1, length=1

- dequeue() – returns 18

  [_,_,_,_,_] front=1, back=1, length=0