Algorithms and computational complexity

Preparation for the laboratory 3: Stacks. Simple algorithms that use stacks.

1. **Stack concept.**
   A stack is one of the types of data structures. The subsequent elements (data) are placed at the top of the stack. Only the element at the top of the stack can be removed. A good example of how the stack works is a stack of books on your desk. We can add another book by placing it on the top, after which the newly added book becomes the top of the stack. If we want to take a book from such a stack, we can only take a book that lies on top. If we want to reach for another book, we have to take the books off the top one by one.

2. **Stack operations.**
   push - adding an item to the stack
   pop - download the item from the stack (and return its value)
   isEmpty - check if the stack is empty
   Often, stack implementations provide additional operations, such as
   peek - check the value of the element at the top of the stack without removing it

3. IStack interface - a stack holding integers.

```
public interface IStack {
public void push (int i); // adds i on stack
 public int pop (); // removes the element, returns the value from the top
public int peek (); // returns the value from the top of the stack, but does not remove it
public boolean isEmpty (); // check if the stack is empty
public void print (); // print the contents of the stack on the screen
public void clear (); // remove all items from the stack
}
```

Tasks.
1. Implement the *IStack* interface basing on an array of integer. To do this, create a *TStack* class containing a private array for storing stack elements and a component counter.
2. Implement throwing exceptions in situations that lead to an inconsistent stack state (e.g., an attempt to add an element to a completely full stack, or attempt to remove an item from an empty stack). We throw an exception using the throw statement, e.g. *throw new Exception ("Stack full!");*
3. In the *TStack* class, implement two additional constructors: one that allows you to specify the size of the array (accepts an int parameter), the second constructor (so-called copy constructor) to take as a parameter another *TStack* object and copy its contents to the newly created object.
4. Create a *TStackDynamic* class that implements the *IStack* interface and also uses an array, but when adding data to a full stack, the internal array is doubled, and the elements are rewritten.