Past Year PE1 Question: ArrayStack

Adapted from PE1 of 21/22 Semester 2

Instructions to Past-Year PE1 Question:

1. Accept the repo on GitHub Classroom here
2. Log into the PE nodes and run `~cs2030s/get py1` to get the skeleton for all available past year PE1 questions.
3. The skeleton for this question can be found under `2122-s2-q2`. You should see the following files:
   - The files `Test1.java`, `Test2.java`, and `CS2030STest.java` for testing your solution.
   - The skeleton files for this question: `ArrayStack.java`, and `Stack.java`
   - You may add new classes/interfaces as needed by the design.

Background

Recall the Stack, a First-In-Last-Out (FILO) data structure. You can pop an item off the top of the stack, and push an item on to the stack. In this question, we will implement a generic stack using an array.

In this question, you are not permitted to use `java.util.Stack` or `java.util.ArrayList`.

Create a new generic interface `Stack` and an `ArrayStack`

We first need to create a `Stack<T>` interface. It is a generic interface, with three abstract methods: - A `pop` method which returns an object of type `T` and has no arguments - A `push` method which returns nothing and has a single argument of type `T` - A `getStackSize` method returns an `int` and has no arguments

Next, create a class `ArrayStack<T>` which implements `Stack<T>` using an array. The order of the items in the array dictates the order of items in the stack. This class has a constructor that takes in a single `int` which represents the maximum depth of the stack.

The `push` method should put an item on top of the stack. If there is no more space in the stack, the `push` method should disregard the item being pushed onto the stack. The `pop` method should remove an item from the top of the stack and return it. If there are no items on the stack, the `pop` method should return `null`. The `getStackSize` method should return how many items are in the stack. Finally, the `toString` method should show the contents of the stack.

If you find yourself in a situation where the compilers generate an unchecked type warning, but you are sure that your code is type-safe, you can use `@SuppressWarnings("unchecked")` (responsibly) to suppress the warning.

Study the sample calls below to understand what is expected for the constructor, `toString` and other methods of `ArrayStack`. Implement your class so that it outputs in the same way.

```
jshell> Stack<Integer> st = new ArrayStack<>(3);
st ==> Stack:
jshell> st.push(1);
jshell> st;
st ==> Stack: 1
jshell> st.push(1);
jshell> st;
st ==> Stack: 1 1
jshell> st.push(2);
jshell> st;
st ==> Stack: 1 1 2
jshell> st.getStackSize();
$.. ==> 3
jshell> st.push(3);
jshell> st;
st ==> Stack: 1 1 2
jshell> st.pop();
$.. ==> 2
jshell> st;
st ==> Stack: 1 1
jshell> st.getStackSize();
$.. ==> 2
jshell> st.pop();
$.. ==> 1
jshell> st
st ==> Stack: 1
jshell> st.getStackSize();
$.. ==> 1
jshell> st.pop();
$.. ==> 1
jshell> st
st ==> Stack:
jshell> st.pop();
$.. ==> null
jshell> st
st ==> Stack:
jshell> st.pop();
```

```
$.. ==> null
jshell> st
st ==> Stack:
jshell> st.push(2);
jshell> st;
st ==> Stack: 2
jshell> Stack<String> st2 = new ArrayStack<>(10);
st2 ==> Stack:
jshell> st2.push("Hello");
jshell> st2;
st2 ==> Stack: Hello
jshell> st2.push("World");
jshell> st2;
st2 ==> Stack: Hello World
jshell> st2.pop();
$.. ==> "World"
jshell> st2.pop();
$.. ==> "Hello"
```

You can test your code by running the `Test1.java` provided. Make sure your code follows the CS2030S Java style.

```
$ javac -Xlint:rawtypes -Xlint:unchecked Test1.java
$ java Test1
$ java -jar ~cs2030s/bin/checkstyle.jar -c ~cs2030s/bin/cs2030_checks.xml
*.java
```

Creating a factory method `of` and a `pushAll` method

We will now implement a factory method `of`, this method will take in an array of items and an `int` which represents the maximum depth of the stack, and return an `ArrayStack` with the items pushed onto the stack in the order that they are present in the array. If the array length is greater than the size of the stack, only include the first `n` items of the array, where `n` is the stack size. For compatibility with `Test1.java`, you should not make your original constructor private.

We will also create a `pushAll` method that has a single argument which is an `ArrayStack`. `pushAll` repeatedly pops one item from the given `ArrayStack` and pushes it onto the target `ArrayStack`, until the given `ArrayStack` is empty. Note that if the target `ArrayStack` is full, the pushed items will be lost.

In addition, we will create a `popAll` method that has a single argument which is an `ArrayStack`. `popAll` repeatedly pops one item from the target `ArrayStack` and pushes it onto the given `ArrayStack`, until the target `ArrayStack` is empty. Note that if the given `ArrayStack` is full, the pushed items will be lost.

Study the sample calls below to understand what is expected for the new methods of

`ArrayStack`. Implement your class so that it outputs in the same way.

```
jshell> ArrayStack.of(new Integer[] {1, 2, 3}, 10);
$.. ==> Stack: 1 2 3
jshell> ArrayStack.of(new Object[] {1, "foo", "bar"}, 10);
$.. ==> Stack: 1 foo bar
jshell> ArrayStack<Integer> as0 = ArrayStack.of(new Integer[] {1, 2, 3,
4}, 2);
as0$ ==> Stack: 1 2
jshell> ArrayStack<Integer> as1 = ArrayStack.of(new Integer[] {4, 5, 6},
10);
as1 ==> Stack: 4 5 6
jshell> ArrayStack<Integer> as2 = ArrayStack.of(new Integer[] {1, 2, 3},
10);
as2 ==> Stack: 1 2 3
jshell> as2.pushAll(as1);
jshell> as2;
as2 ==> Stack: 1 2 3 6 5 4
jshell> as1;
as1 ==> Stack:
jshell> as1 = ArrayStack.of(new Integer[] {4, 5, 6}, 10);
as1 ==> Stack: 4 5 6
jshell> ArrayStack<Integer> as3 = ArrayStack.of(new Integer[] {1, 2, 3},
5);
as3 ==> Stack: 1 2 3
jshell> as3.pushAll(as1);
jshell> as3;
as3 ==> Stack: 1 2 3 6 5
jshell> ArrayStack<Number> asn = new ArrayStack<>(10);
asn ==> Stack:
jshell> asn.pushAll(as2);
jshell> asn
asn ==> Stack: 4 5 6 3 2 1
jshell> ArrayStack<String> as4 = ArrayStack.of(new String[] {"d", "e",
"f"}, 10);
as4 ==> Stack: d e f
jshell> ArrayStack<String> as5 = ArrayStack.of(new String[] {"a", "b",
"c"}, 10);
as5 ==> Stack: a b c
jshell> as4.popAll(as5);
jshell> as5;
as5 ==> Stack: a b c f e d
jshell> as4 = ArrayStack.of(new String[] {"d", "e", "f"}, 10);
as4 ==> Stack: d e f
jshell> ArrayStack<String> as6 = ArrayStack.of(new String[] {"a", "b",
"c"}, 5);
as6 ==> Stack: a b c
jshell> as4.popAll(as6);
jshell> as6;
as6 ==> Stack: a b c f e
jshell> ArrayStack<Integer> as7 = ArrayStack.of(new Integer[] {7, 8, 9},
5);
as7 ==> Stack: 7 8 9
jshell> as7.popAll(asn);
jshell> asn;
asn ==> Stack: 4 5 6 3 2 1 9 8 7
```

You can test your code by running the `Test2.java` provided. Make sure your code follows the CS2030S Java style.

```
$ javac -Xlint:rawtypes -Xlint:unchecked Test2.java
$ java Test2
$ java -jar ~cs2030s/bin/checkstyle.jar -c ~cs2030s/bin/cs2030_checks.xml
*.java
```

Stack.java

```
interface Stack<T> {
    T pop();

    void push(T item);

    int getStackSize();
}
```

ArrayStack.java

```java
public class ArrayStack<T> implements Stack<T> {
    private int maxStackSize;
    private T[] stack;
    private int currentIndex = 0;

    public ArrayStack(int maxStackSize) {
        this.maxStackSize = maxStackSize;
        // I know what I am doing alright?
        @SuppressWarnings("unchecked")
            T[] temp = (T[]) new Object[maxStackSize];
        this.stack = temp;
    }

    @Override
    public T pop() {
        if (currentIndex == 0) {
            return null;
        }
        currentIndex--;
        T item = this.stack[currentIndex];
        return item;
    }

    @Override
    public void push(T item) {
        if (currentIndex != maxStackSize) {
            this.stack[currentIndex] = item;
            currentIndex++;
        }
    }

    @Override
    public String toString() {
        String s = "Stack:";
        for (int i = 0; i < currentIndex; i++) {
            s += " " + this.stack[i];
        }
        return s;
    }

    @Override
    public int getStackSize() {
        return this.currentIndex;
    }


    public static <S> ArrayStack<S> of(S[] a, int maxStackSize) {
        ArrayStack<S> tempStack = new ArrayStack<>(maxStackSize);

        int copyLength = Math.min(a.length, maxStackSize);
        for (int i = 0; i < copyLength; i++) {
            tempStack.push(a[i]);
        }
        return tempStack;
    }

    public void pushAll(ArrayStack<? extends T> stack) {
```

```
58          int copyLength = Math.min(
59                  stack.getStackSize(),
60                  maxStackSize - this.getStackSize());
61          for (int i = 0; i < copyLength; i++) {
62              this.push(stack.pop());
63          }
64      }
65
66      public void popAll(ArrayStack<? super T> stack) {
67          int copyLength = Math.min(
68                  stack.getStackSize(),
69                  maxStackSize - this.getStackSize() + 1);
70
71          for (int i = 0; i < copyLength; i++) {
72              stack.push(this.pop());
73          }
74      }
75  }
```