Transactional Stack

Background

A **stack** is a data structure which allows for ordered management of a collection of similar data types. It is defined as a LIFO (last-in first out) data structure so items which have been added last, are removed first from the collection. A stack consist of at least two methods:

- push: Adding an item on the top of the stack
- pop: Removing an item from the top of the stack and returning the value

A **transaction** is a closed set of changes which are either fully committed or fully rolled back to the affected system. Transaction are a closed scope, changes done in the same transaction scope are unaffected by changes which have been done outside the scope. A transaction consists of at least three methods:

- start: Starts the scope of the transaction
- commit: Applies the changes from the current transaction scope
- rollback: Discards the changes from the current transaction scope

Your Task

Your task is to implement an Integer Stack class which implements transactional behaviour and supports nested transactions. Nested transaction allow to open another transaction scope while a transaction scopes is already open. When commiting or rolling back only the latest transaction scope is applied or discarded.

For your implementation please use **PHP 7.2.**

Please use the Interfaces which have been provided with the Zip file. (Code is outlined below as well)

As result please return all files which are necessary to run your code as a zip file.

Good Luck!

StackInterface

```php
interface StackInterface
{
    /**
     * Pushes an integer value on top of the stack
     *
     * @param int $value
     */
    public function push(int $value): void;

    /**
     * Removes the top-most value from the stack and returns it.
     *
     * @return int
     */
    public function pop(): int;

    /**
     * Returns the top-most value from the stack.
     *
     * @return int
     */
    public function peek(): int;
}
```

TransactionableInterface

```
interface TransactionableInterface
{
    /**
     * Begins a transaction scope
     */
    public function start(): void;

    /**
     * Ends a transaction scope and applies all changes to the object
     * Returns true on success, false if no transaction scope is
available
     *
     * @return bool
     */
    public function commit(): bool;

    /**
     * Ends a transaction scope and discards all changes and rolls back
to
     * the state when the transaction has been started
     * Returns true on success, false if no transaction scope is
available
     *
     * @return bool
     */
    public function rollback(): bool;
}
```