PA 2

File list

common/BaseThread.java
common/Semaphore.java
BlockManager.java
BlockStack.java

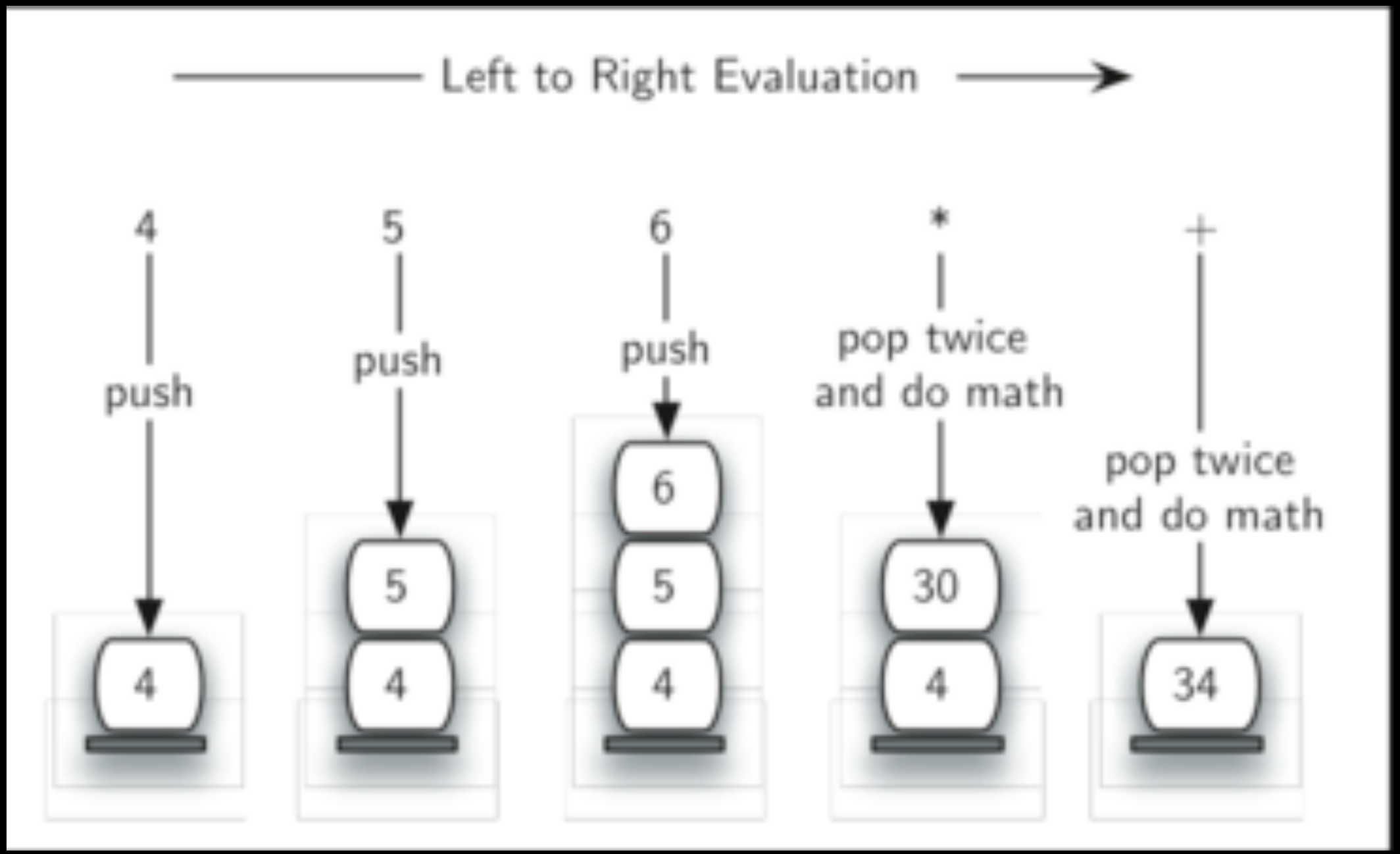
Semaphore.java

- Wait() (uppercase not `wait`) -> Acquire permit
- Signal() -> Release permit

BaseThread.java

- Monotonic increase ThreadID
- Phase 1 & Phase 2 prints out task
- TurnID (will explain)

Stack


Left to Right Evaluation
4
push
5
push
6
push
*
pop twice
and do math
+
pop twice
and do math
4
5
4
6
5
4
30
4
34

BlockStack.java

- iTop (top index of the internal array)
- iSize (capacity of stack)
- accessCounter (number of access to stack)
- Uses two Poison pills ($) ???

Compiling source code...

Missing methods

- getTop() -> retrieve iTop (TopIndex)
- getSize() -> retrieve iSize
- getAccessCounter() -> count time of accesses
- isEmpty()

BlockManager.java

AcquireBlock => Pops element to stack
ReleaseBlock => Pushes element to stack
ProberBlock => Inspects information stack

main()

- Creates BlockStack
- Creates 3 Acquirers
  -> phase1, pop stack, phase2
- Creates 3 Pushers
  -> phase1, push stack, phase2
- Creates 5 Probers
  -> phase1, print stack, phase2

Execution structure

1. Printout direction of phase 1
2. Do anything
3. Printout direction of phase 2

Assignment Tasks

- Other TAs may require differently
- Recommend to create package or project not file1, file2, file3

0. Missing methods

- getTop() -> retrieve iTop (TopIndex)
- getSize() -> retrieve iSize
- getAccessCounter() -> count time of accesses
- isEmpty()

1. Implements stack access counter

- Declare the counter
- Increase 1 when `push`, `pop`, `pick` & `getAt`

2. Correct the stack implementation

- Create Getter for iSize, iTop & acStack (Step 0)
- Add Push() method to push a
- Throw exceptions when
  1. Pop or peek an empty list (EmptyStackException)
  2. Push a full stack (OutOfCapacity)

3. Make things atomic

- Figure out the error of output
- Fix by using Mutex (binary semaphore)

4. Phase 1 must be completed before phase 2

- Easiest way is used Barrier class
- You can use semaphore

1. All 10 threads must start their PHASE II in order of their TID, i.e. 1, 2, 3, 4 ...

Semantic:

- How can we determine that PHASE2 gets started?
- PHASE2 of 2 only started when PHAS2 of 1 completed

TurnID

- Global state
- turnTestAndSet()
  1. If TurnID equals ThreadID: increases turnID, returns true
  2. Otherwise, returns false
- `synchronized` of turnTestAndSet is incorrect but no harm (threadIDs are unique)

Task5's Solutions

1. turnTestAndSet() and s2
2. CountUpLatch with `countUp` & `await(n)`
3. JDK Semaphore with `acquire(n)` & `release(n)`

CountUpLatch & JDK Semaphore won't change iTurn (It's OK)

CountUpLatch

```
public interface CountUpLatch {
    /**
     * Increase the counter value by one.
     * Notify to waiting instances
     */
    void countUp();

    /**
     * If the counter is equal or greater than value, just returns
     * Otherwise waits until satisfies the condition
     * @param value required value
     * @throws InterruptedException
     */
    void await(int value)throws InterruptedException;
}
```

Revisited

1. Asks your TA what he wants
2. You can use ANYTHING
3. Recommend to create package or project not file1, file2, file3