peek and isEmpty

- Property we want: If there has been a push and no pop, then isEmpty returns false

- With peek as written, property can be violated – how?

```java
push(x)
boolean b = isEmpty()
```

```java
E ans = pop();
push(ans);
return ans;
```
**peek and isEmpty**

- Property we want: If there has been a `push` and no `pop`, then `isEmpty` returns `false`.

- With `peek` as written, property can be violated – how?

```java
E ans = pop();
push(ans);
return ans;
```

```java
push(x)
boolean b = isEmpty()
```
**peek and push**

- Property we want: Values are returned from `pop` in LIFO order
- With `peek` as written, property can be violated – how?

```plaintext
Thread 1 (peek)
E ans = pop();
push(ans);
return ans;

Thread 2
push(x)
push(y)
E e = pop()
```
peek and push

- Property we want: Values are returned from `pop` in LIFO order
- With `peek` as written, property can be violated – how?

```
Thread 1 (peek)
E ans = pop();
push(ans);
return ans;

Thread 2
push(x)
push(y)
E e = pop()
```
**peek and pop**

- Property we want: Values are returned from `pop` in LIFO order
- With `peek` as written, property can be violated – how?

```
Thread 1 (peek)
E ans = pop();
push(ans);
return ans;
```

```
Thread 2
push(x)
push(y)
E e = pop()
```
**peek and peek**

- Property we want: **peek** does not throw an exception if number of pushes exceeds number of pops

- With **peek** as written, property can be violated – how?

```java
Thread 1 (peek)
E ans = pop();
push(ans);
return ans;
```

```java
Thread 2
E ans = pop();
push(ans);
return ans;
```
**peek and peek**

- Property we want: `peek` doesn’t throw an exception if number of pushes exceeds number of pops

- With `peek` as written, property can be violated – how?

```java
Time

Thread 1 (peek)

E ans = pop();
push(ans);
return ans;

Thread 2

E ans = pop();
push(ans);
return ans;
```