String methods

length()

length()

- `foo.length()`
- returns the length of `foo`
- What is the length of the smallest possible string?

foo.substring(left, right)

foo.substring(left, right)

- returns a new String that is made by slicing `foo`
- new String made of the characters between index `left` and `right`
- What would make `substring` crash?
- What are things that we should know about `left` and `right` when using substring?

substring(start)

substring(start)

- `foo.substring(start)`
- returns a new String that is made by slicing foo
- new String made of characters beginning at index `start` and going to the end of `foo`

indexOf(bar)

indexOf(bar)

- `foo.indexOf(bar)`
- returns the beginning index of the first occurrence of `bar` within `foo`
- What if `bar` doesn't show up at all?

equals(bar)

equals(bar)

- `foo.equals(bar)`
- returns `true` if `foo` contains the exact same characters in the exact same order as `bar`
- unless you know what you're doing, USE `.equals()` ***NOT*** `==` for Strings

compareTo(bar)

compareTo(bar)

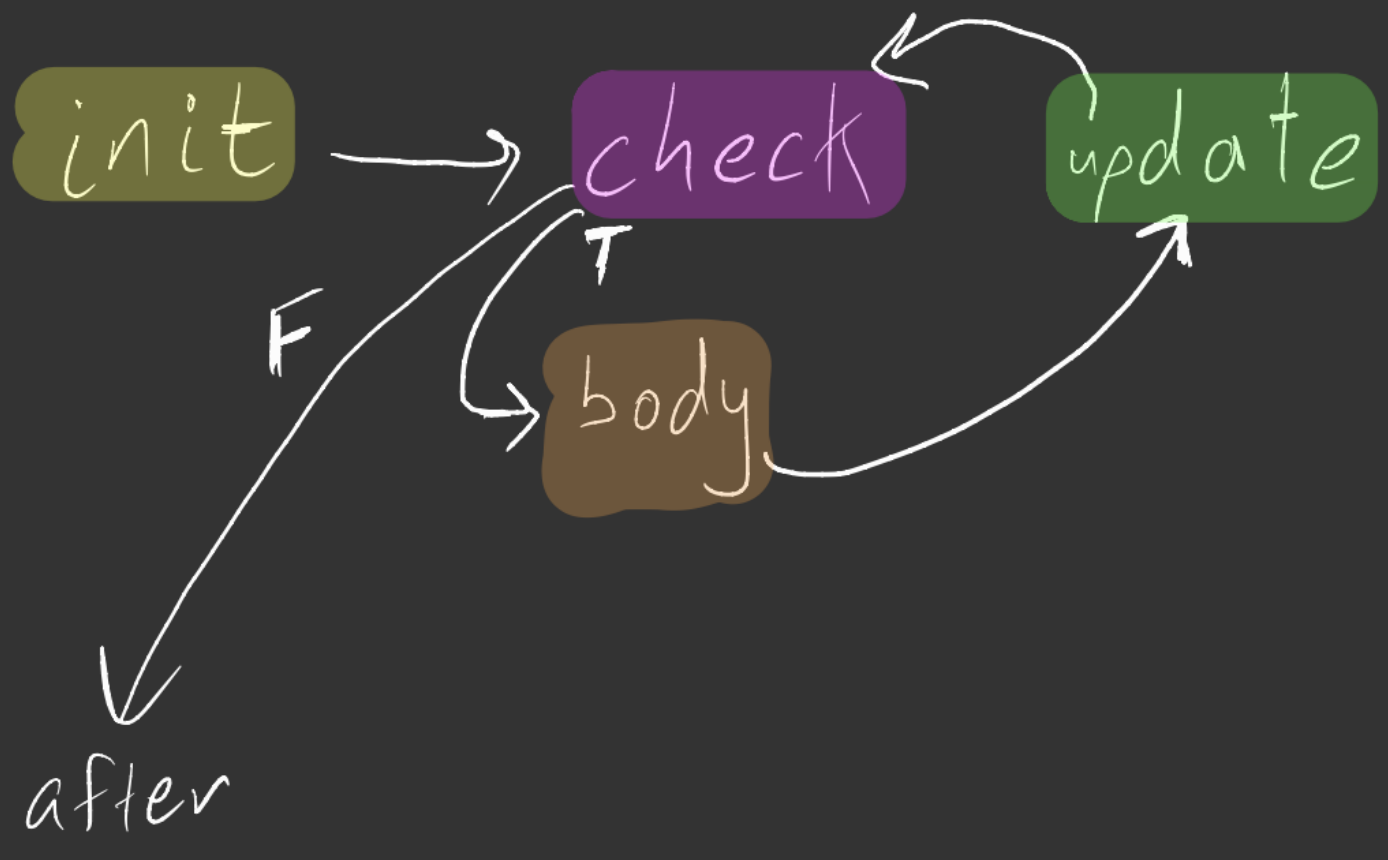
- `foo.compareTo(bar)`
- returns `-1` if "`foo < bar`", i.e., if `foo` comes "before" `bar` in a special ordering called lexicographic ordering
- returns `1` if "`foo > bar`", i.e., if `foo` comes "after" `bar` in a special ordering called lexicographic ordering
- returns `0` if `foo.equals(bar)`

for loops

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

- header:
    - initialization clause: happens 1st, happens once
    - check clause: evaluated, if true, body runs
    - update clause: happens after each time body runs
- init, check, body, update, check, body, update...

```
for (int i = 0; i < 5; i++) {
    int num = i*i + i;
    System.out.println(num);
}
```



init

check

body

update

---

check

body

update

---

check

body

update

etc.

for loops

- "safer" than while loops
- easier to read than while loops (once you're used to it)
- if you can easily write it as a for loop, you probably should