EECS2030 Advanced Object-Oriented Programming (Fall 2021)

Q&A - Review Tutorial Part 2

Thursday, September 23

Announcement

2pm EST

- Lab0P2 (due: Sep. 24)
- Lecture W3 (released: Sep. 20)
- Lab1 (released: Sep. 22) 8 days.
- Written Test (due: Sep. 30 - Oct. 1)

For part 26, at approx. 5:06 minute mark, why are we setting the index to -1 and not 0?

I know it is "expected to be re-assigned to a valid index if the 'sn' exists in the store" but I don't fully understand what that implies.

Thank you for your time.

`rs.getProduct("junk");`

```
public Product getProduct(String sn) {
    int index = -1; // expected to be re-assigned to a valid index if the 'sn' exists in the store
    // Problematic exit condition: i < this.entries.length
    for(int i = 0; i < this.noe; i ++) {
        Entry e = this.entries[i];
        if(e.getSerialNumber().equals(sn)) {
            index = i;
        }
    }
    if(index < 0) {
        return null;
    }
    else {
        return this.entries[index].getProduct();
    }
}
```

① valid index

② Say sn input does not exist ⇒ * will always evaluate to false

** never executed

rs → RefurbishedStore (noe: 3, entries)

entries: 0 1 2 3 4

e1 → Entry (sn: F9FDN..., p:) → Product (m.: iPad Pro 12.9, f.: Space Grey, s.: 1000, c.c.: true, o.p.: 1709.00, d.v.: 220.00)

* → Entry (sn: C9FZN..., p:) → Product (m.: iPad Air, f.: Gold, s.: 64, c.c.: false, o.p.: 649.00, d.v.: 100.00)

** → Entry (sn: 7YM4P..., p:) → Product (m.: iPad Pro 10.5, f.: null, s.: 0, c.c.: false, o.p.: 929.00, d.v.: 0.0)

index == 0 ⇒ if input sn # exists,

re-assign index to some valid index

⇒ 0 ↘ mistakenly return product at index 0

```
public Product getProduct(String sn) {
    [redacted] /* expected to be re-assigned
    // Problematic exit condition: i < this.entri
    for(int i = 0; i < this.noe; i ++) {
        Entry e = this.entries[i];
        if(e.getSerialNumber().equals(sn)) {
            [redacted]
        }
    }
    return null;
}
```

return e.getProduct();

```
if(index < 0) {
    return null;
}
else {
    return this.entries[index].getProduct();
}
```

1. In lab or PT, you can use either solution.

2. When your method has multiple returns, in general, it becomes harder to judge its correctness. Instead, use a single return at the end.

return (index >= 0) ? this.es[i].getP : null;

Single return version

```
Product result = null;
if(index >= 0) {
    result = this.entries[i].getProduct();
}
return result;
```

```
public Product getProduct(String sn) {
    int index = -1; /* expected to be re-assigned
    // Problematic exit condition: i < this.entri
    for(int i = 0; i < this.noe; i ++) {
        Entry e = this.entries[i];
        if(e.getSerialNumber().equals(sn)) {
            index = i;
        }
    }

    if(index < 0) {
        return null;
    }
    else {
        return this.entries[index].getProduct();
    }
}
```

boolean matchNotFound = true;

matchNotFound = false;

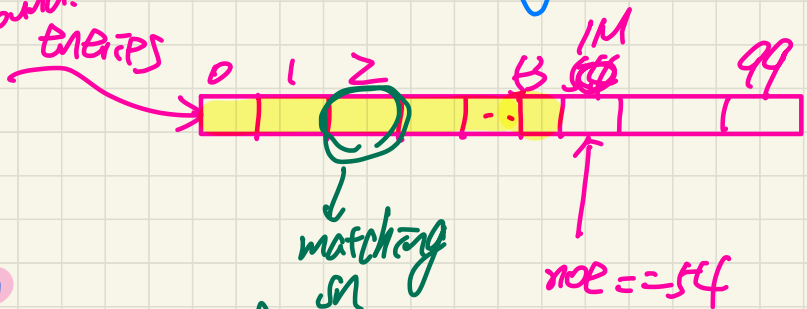
index == -1

① i < this.noe &&

② i < this.noe && matchNotFound.

Question:

In this version, the # of iterations corresponds to this.noe - 1 ~~× this.entries.length - 1~~



Challenge:

How can we exit the loop as soon as a matching sn is found?

⇒ What if this.noe is large (e.g. 1M), but the product with the matching sn exists in early part of array.

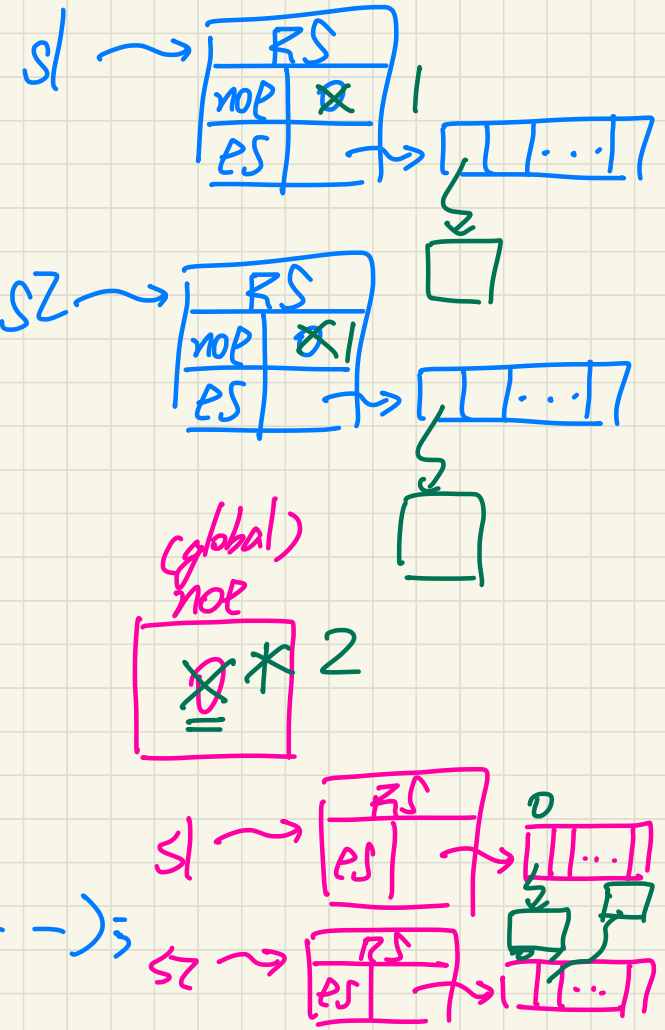
without having multiple return.

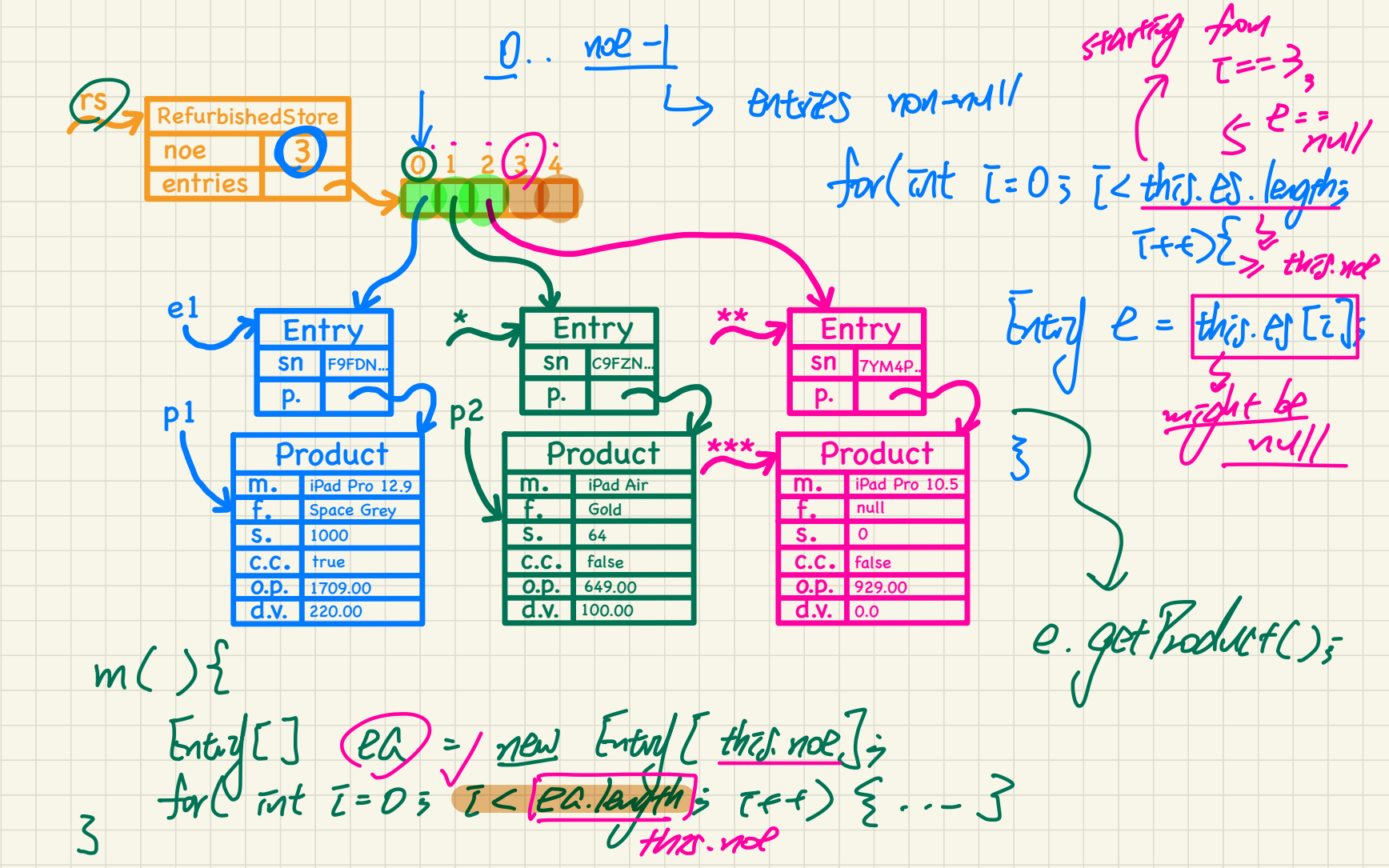
```
class RS {
    static int noe = 0;
    Entry[] es;

    void addEntry(Entry e) {
        this.es[this.noe] = e;
        this.noe ++;
    }
}
```

global, single copy shared by all RS objects

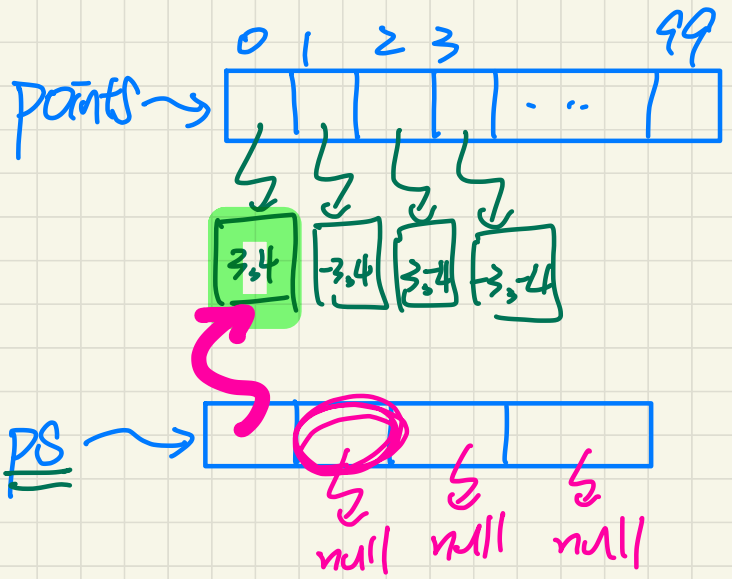
```
RS s1 = new RS();
RS s2 = new RS();
s1.addEntry(...);   s2.addEntry(...);
```




0 .. noe - 1
entries non-null
rs
RefurbishedStore
noe
3
entries
0 1 2 3 4
e1
p1
p2
*
**
***
Entry
sn
F9FDN...
p.
Product
m.
iPad Pro 12.9
f.
Space Grey
s.
1000
c.c.
true
o.p.
1709.00
d.v.
220.00
Entry
sn
C9FZN...
p.
Product
m.
iPad Air
f.
Gold
s.
64
c.c.
false
o.p.
649.00
d.v.
100.00
Entry
sn
7YM4P...
p.
Product
m.
iPad Pro 10.5
f.
null
s.
0
c.c.
false
o.p.
929.00
d.v.
0.0
starting from i == 3;
<= e == null
for(int i=0; i< this.es.length; i++){
>= this.noe
Entry e = this.es[i];
might be null
e.getProduct();
m(){
Entry[] ea = new Entry[this.noe];
for(int i=0; i < ea.length; i++) {...}
this.noe
}

```
public Point[] getPointsInQuadrantI() {
  Point[] ps = new Point[this.nop];
  int count = 0; /* number of points in Quadrant I */
  for(int i = 0; i < this.nop; i ++) {
    Point p = points[i];
    if(p.getX() > 0 && p.getY() > 0) {
      ps[count] = p;
      count ++;
    }
  }
  Point[] q1Points = new Point[count];
  /* ps contains null if count < nop */
  for(int i = 0; i < count; i ++) {
    q1Points[i] = ps[i]
  }
  return q1Points;
}
```

PS



① assertTrue(pc.getPIQI().length==1); ×

fail

② assertTrue(pc.getPIQI()[1].getX()==3); ×

null

NullPointerException

① pc.addPoint(new Point(3,4))

↳ anynomous object

② Point p1 = new Point(3,4);

pc.addPoint(p1);

→ do we need to invoke any methods on p1 ?

p1.moveUp(2);

YES → then ① would not allow you to do that

No → then ②