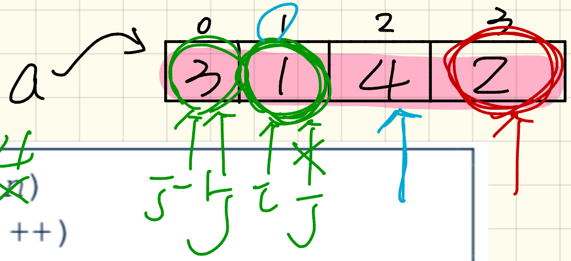
Wednesday Oct. 24

Lecture 13

Insertion Sort : Code



```
insertionSort(int[] a, int n)
  for (int i = 1; i < n; i ++)
    int current = a[i];
    int j = i;
    while (j > 0 && a[j - 1] > current)
      a[j] = a[j - 1];
      j --;
    a[j] = current;
```

4 ~~n~~

i = 1, 2, 3

j = 1, 0

F

a[1] > 4

3

a[1] = a[0]

→ a[0] = 1

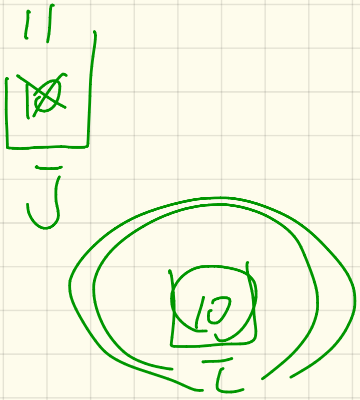
i	current	j at L8	a at L8	a after L8
1	a[1] 1		3 $\overset{3}{\cancel{1}}$ 4 2	1 3 4 2
2	a[2] 4			1 3 4 2 (i)
3	a[3] 2		1 $\overset{2}{\cancel{3}}$ $\overset{3}{\cancel{4}}$ $\overset{4}{\cancel{2}}$	

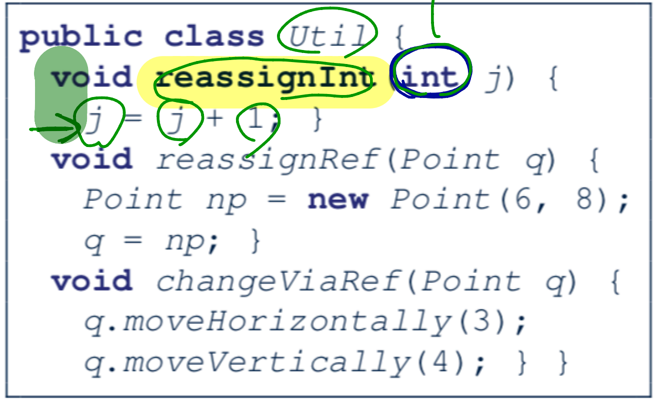
Call by Value : Primitive Type

$\boxed{\text{Scope of } j}$

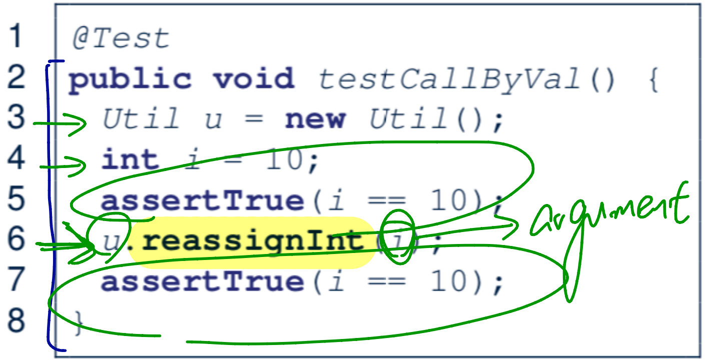
Implicitly:

$j = i$





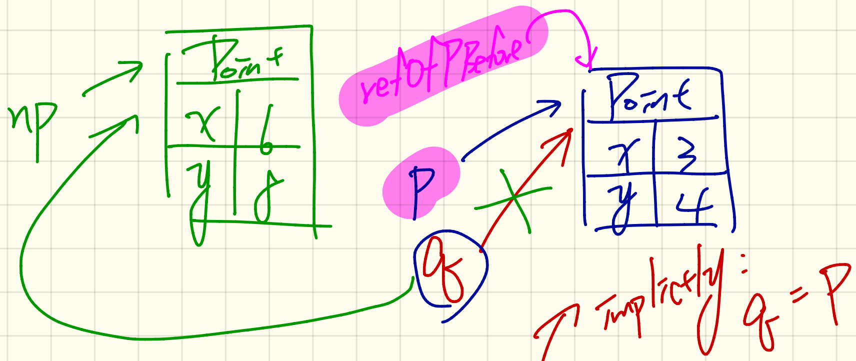
```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```



```
@Test
public void testCallByVal() {
  Util u = new Util();
  int i = 10;
  assertTrue(i == 10);
  u.reassignInt(i);
  assertTrue(i == 10);
}
```

argument

Call by Value : Reference Type (1)



```
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

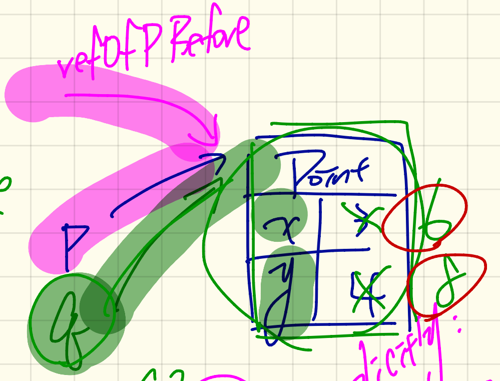
```
1  @Test
2  public void testCallByRef_1() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.reassignRef(p);
7    assertTrue(p==refOfPBefore);
8    assertTrue(p.x==3 && p.y==4);
9  }
```

argument

Call by Value: Reference Type (2)

1. p and q are aliases of the same object.

2. To modify that object, you can use p or q as C.O.

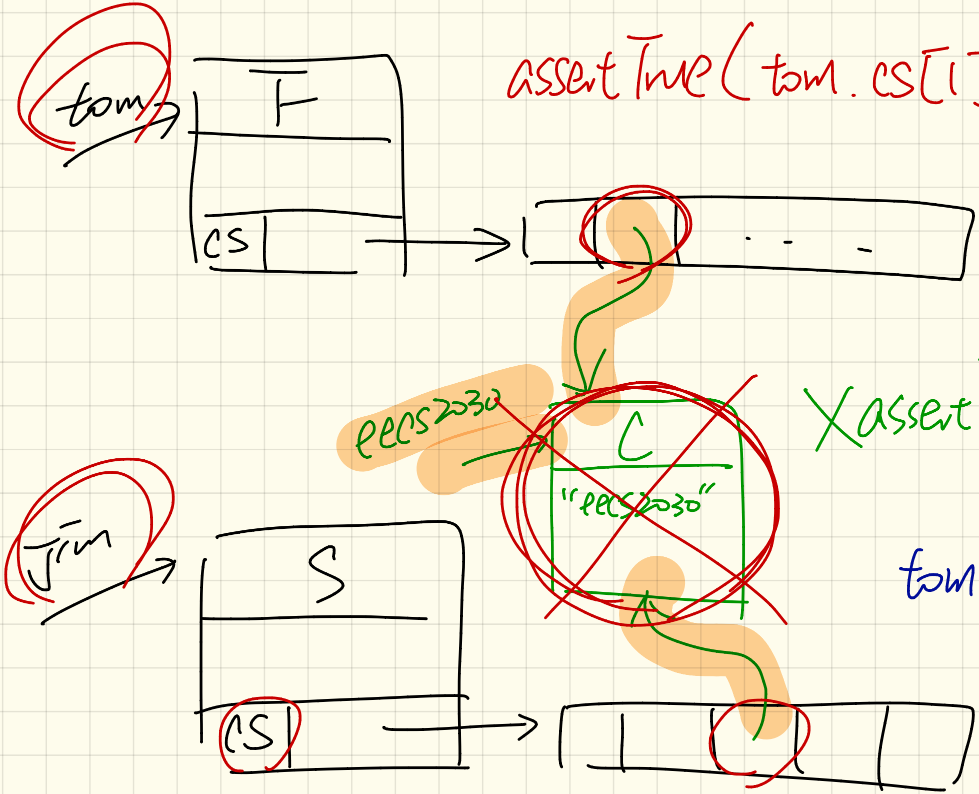


```
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

```
@Test
public void testCallByRef_2() {
  Util u = new Util();
  Point p = new Point(3, 4);
  Point refOfPBefore = p;
  u.changeViaRef(p);
  assertTrue(p==refOfPBefore);
  assertTrue(p.x==6 && p.y==8);
}
```

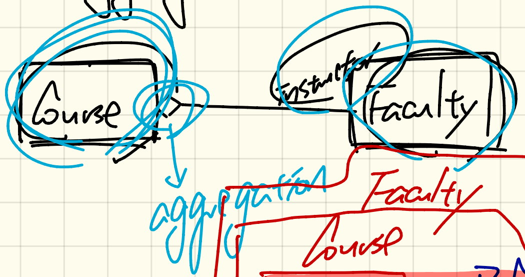
assertTrue(tom.cs[1] == jim.cs[2])



X assertEquals(,)
.equals

tom.cs[1].setName("eecs2040")

Aggregation (1)



```
class Course {
  String title;
  Faculty prof;
  Course(String title) {
    this.title = title;
  }
  void setProf(Faculty prof) {
    this.prof = prof;
  }
  Faculty getProf() {
    return this.prof;
  }
}
```

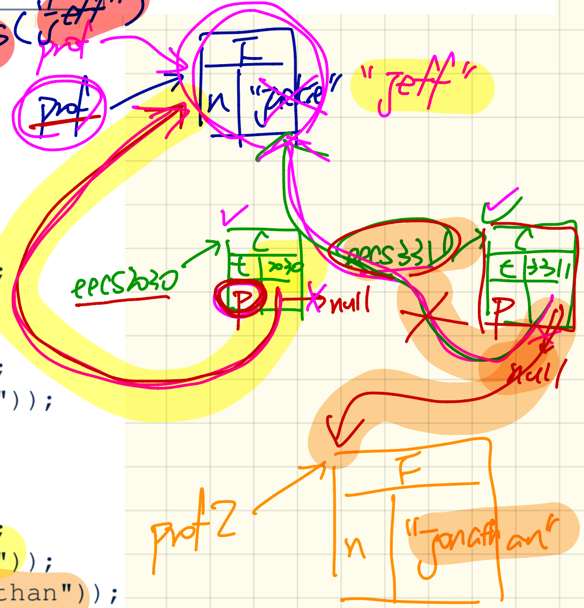
eecs2030
eecs3311

```
class Faculty {
  String name;
  Faculty(String name) {
    this.name = name;
  }
  void setName(String name) {
    this.name = name;
  }
  String getName() {
    return this.name;
  }
}
```

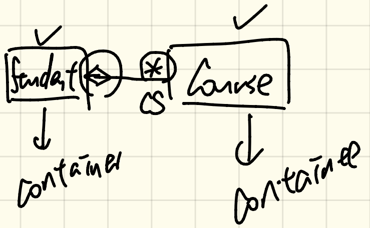
Faculty
Course
eecs2030.getProf().getName().equals("Jeff")
String

```
@Test
public void testAggregation1() {
  Course eecs2030 = new Course("Advanced OOP");
  Course eecs3311 = new Course("Software Design");
  Faculty prof = new Faculty("Jackie");
  eecs2030.setProf(prof);
  eecs3311.setProf(prof);
  assertTrue(eecs2030.getProf() == eecs3311.getProf());
  /* aliasing */
  prof.setName("Jeff");
  assertTrue(eecs2030.getProf() == eecs3311.getProf());
  assertTrue(eecs2030.getProf().getName().equals("Jeff"));

  Faculty prof2 = new Faculty("Jonathan");
  eecs3311.setProf(prof2);
  assertTrue(eecs2030.getProf() != eecs3311.getProf());
  assertTrue(eecs2030.getProf().getName().equals("Jeff"));
  assertTrue(eecs3311.getProf().getName().equals("Jonathan"));
}
```



Aggregation (2)



```java
class Student {
  String id; ArrayList<Course> cs; /* courses */
  Student(String id) { this.id = id; cs = new ArrayList<>(); }
  void addCourse(Course c) { cs.add(c); }
  ArrayList<Course> getCS() { return cs; }
}
```

```java
class Course { String title; ~~}~~ prof }
```

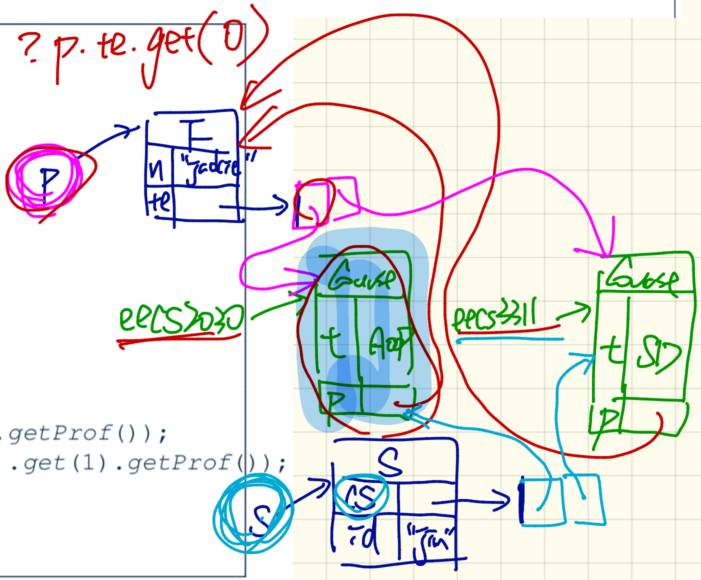
```java
class Faculty {
  String name; ArrayList<Course> te; /* teaching */
  Faculty(String name) { this.name = name; te = new ArrayList<>(); }
  void addTeaching(Course c) { te.add(c); }
  ArrayList<Course> getTE() { return te; }
}
```

```java
@Test
public void testAggregation2() {
  Faculty p = new Faculty("Jackie");
  Student s = new Student("Jim");
  Course eecs2030 = new Course("Advanced OOP");
  Course eecs3311 = new Course("Software Design");
  eecs2030.setProf(p);
  eecs3311.setProf(p);
  p.addTeaching(eecs2030);
  p.addTeaching(eecs3311);
  s.addCourse(eecs2030);
  s.addCourse(eecs3311);

  assertTrue(eecs2030.getProf() == s.getCS().get(0).getProf());
  assertTrue(s.getCS().get(0).getProf() == s.getCS().get(1).getProf());
  assertTrue(eecs3311 == s.getCS().get(1));
  assertTrue(s.getCS().get(1) == p.getTE().get(1));
}
```

eecs2030 == ? p.te.get(0)

==
eecs2030
s.cs.get(0)
p.te.get(0)



p
F
n
"Jackie"
te
eecs2030
Course
t
AOP
p
eecs3311
Course
t
SD
p
s
S
cs
id
"Jim"