Wednesday Sep. 26

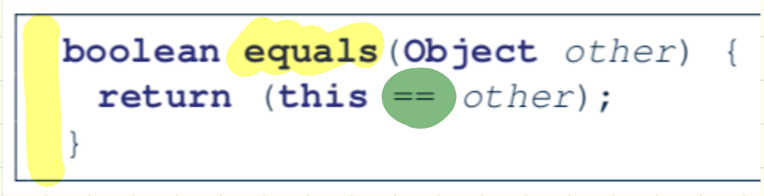
Lecture 7

- Today :
  1. More equals method examples
     (will be covered in Lab Test I)
  2. Comparable and compareTo
     (will not be covered in Lab Test I)

- int hashCode()
  ~ integer accessor (≈ getBMI()) based on attribute values and a formula
  ~ complete story next Monday!

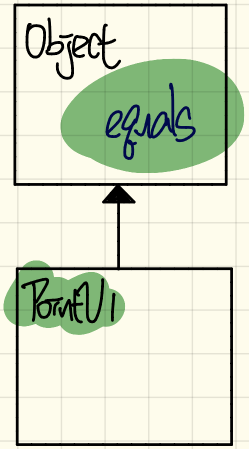
equals method in Object class

Case 1: equals not overridden



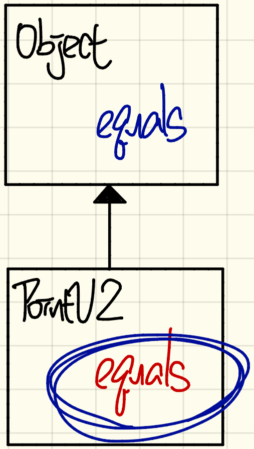
```
boolean equals(Object other) {
  return (this == other);
}
```

```
PointV1 p1 = ...
PointV1 p2 = ...
p1.equals(p2);
```



Case 2: equals overridden

```
PointV2 p3 = ...
PointV2 p4 = ...
p3.equals(p4);
```



(Case 1)

```
boolean equals(Object other) {
  return (this == other);
}
```

```
class PointV1 {
  double x; double y;
  PointV1(double x, double y) { this.x = x; this.y = y; }
}
```

(Case 2)

```
class PointV2 {
  double x; double y;
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false; }
    Point other = (PointV2) obj;
    return this.x == other.x && this.y == other.y; } }
```

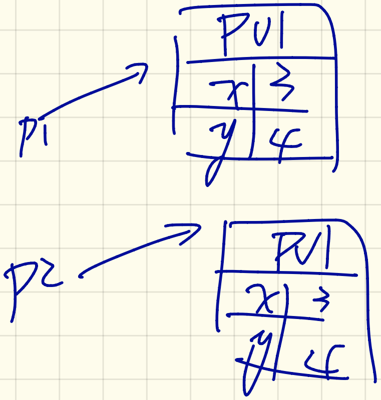
assertSame vs. assertEquals (1)

equals from Object class

```
@Test
public void testEqualityOfPointV1() {
    PointV1 p1 = new PointV1(3, 4);
    PointV1 p2 = new PointV1(3, 4);
    assertFalse(p1 == p2);
    assertFalse(p2 == p1);
    assertSame(p1, p2); // fail
    assertSame(p2, p1); // fail
    // default version of equals
    // from Object is called.
    assertFalse(p1.equals(p2));
    assertFalse(p2.equals(p1));

    // Compare contents of p1 and p2 explicitly
    // this is what a overridden equals would do
    assertTrue(p1.x == p2.x && p2.y == p2.y);
}
```

→ assertTrue(p1 != p2)

```
boolean equals(Object other) {
  return (this == other);
}
```



```
class PointV1 {
double x; double y;
PointV1(double x, double y) { this.x = x; this.y = y; }
}
```

assertSame vs. assertEquals (2)

```java
@Test
public void testEqualityOfPointV2() {
    PointV2 p3 = new PointV2(3, 4);
    PointV2 p4 = new PointV2(3, 4);
    assertFalse(p3 == p4);
    assertFalse(p4 == p3);
    assertSame(p3, p4); // fail
    assertSame(p4, p4); // fail
    // overridden version of equals
    // from PointV2 is called
    assertTrue(p3.equals(p4));
    assertTrue(p4.equals(p3));
    assertEquals(p3, p4);
    assertEquals(p4, p3);
}
```

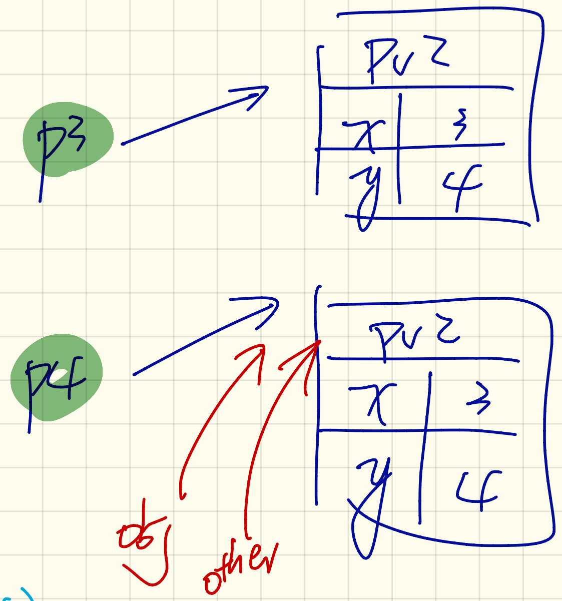
F

this

obj

True

p3.equals(p4)



```java
class PointV2 {
  double x; double y;
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false; }
    PointV2 other = (PointV2) obj;
    return this.x == other.x && this.y == other.y; } }
```

p4

p3 p4

p4

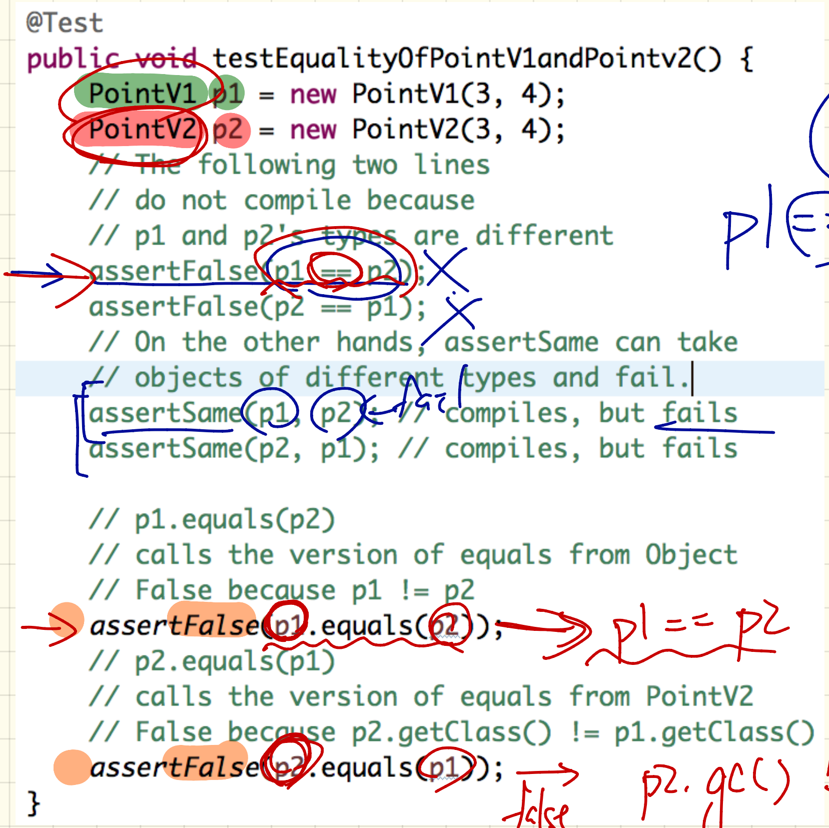
p3 p4

V2

p3 p4 p3 p4

obj.x

assertSame vs. assertEquals (3)



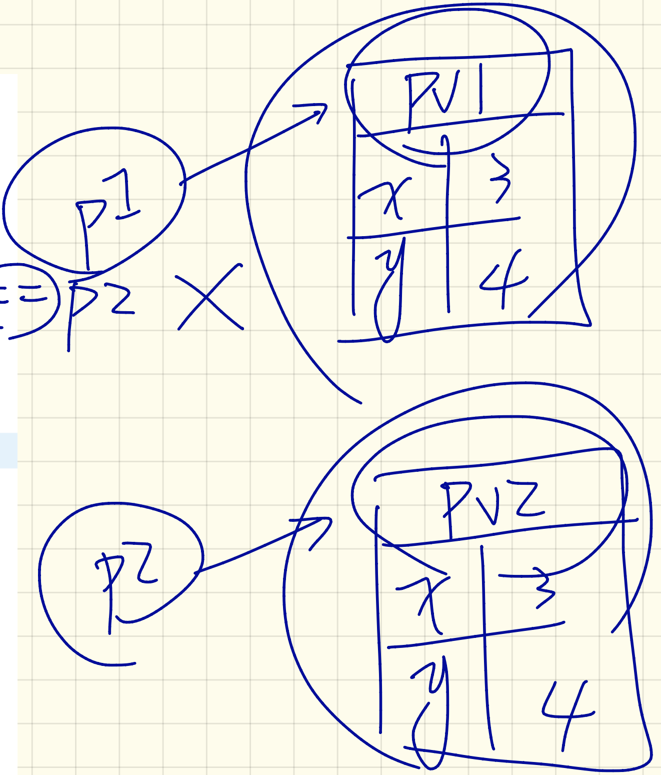
```java
@Test
public void testEqualityOfPointV1andPointv2() {
    PointV1 p1 = new PointV1(3, 4);
    PointV2 p2 = new PointV2(3, 4);
    // The following two lines
    // do not compile because
    // p1 and p2's types are different
    assertFalse(p1 == p2);
    assertFalse(p2 == p1);
    // On the other hands, assertSame can take
    // objects of different types and fail.
    assertSame(p1, p2); // compiles, but fails
    assertSame(p2, p1); // compiles, but fails

    // p1.equals(p2)
    // calls the version of equals from Object
    // False because p1 != p2
    assertFalse(p1.equals(p2));
    // p2.equals(p1)
    // calls the version of equals from PointV2
    // False because p2.getClass() != p1.getClass()
    assertFalse(p2.equals(p1));
}
```

$p1 == p2$ ✗

fail

$\rightarrow p1 == p2$

$\rightarrow$ false $p2.gc() \,!=\, p1.gc()$



Overriding & Reusing equals method

```java
class Person {
    String firstName;
    String lastName;
    double weight;
    double height;

    public Person(String firstName, String lastName, double weight, double height) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.weight = weight;
        this.height = height;
    }

    public boolean equals (Object obj) {
        if(this == obj) { return true; }
        if(obj == null || this.getClass() != obj.getClass()) {
            return false; }
        Person other = (Person) obj;
        return
            this.weight == other.weight
            && this.height == other.height
            && this.firstName.equals (other.firstName)
            && this.lastName.equals (other.lastName);
    }
```

content objects

redefined version in String

```java
class PersonCollector {
    Person[] persons;
    int nop; /* number of persons */

    public PersonCollector() {
        persons = new Person[10];
    }

    public void addPerson(Person p) {
        persons[nop] = p;
        nop ++;
    }

    public boolean equals (Object obj) {
        if(this == obj) { return true; }
        if(obj == null || this.getClass() != obj.getClass()) {
            return false; }
        PersonCollector other = (PersonCollector) obj;
        boolean equal = false;
        if(this.nop == other.nop) {
            equal = true;
            for(int i = 0; equal && i < this.nop; i ++) {
                equal = this.persons[i].equals(other.persons[i]);
            }
        }
        return equal;
    }
}
```

PC1 PC2

0

Person

Testing Person and PersonCollector

```java
@Test
public void testPersonCollector() {
  Person p1 = new Person("A", "a", 180, 1.8); Person p2 = new Person("A", "a", 180, 1.8);
  Person p3 = new Person("B", "b", 200, 2.1); Person p4 = p3;
  assertFalse(p1 == p2); assertTrue(p1.equals(p2));
  assertTrue(p3 == p4); assertTrue(p3.equals(p4));

  PersonCollector pc1 = new PersonCollector(); PersonCollector pc2 = new PersonCollector()
  assertFalse(pc1 == pc2); assertTrue(pc1.equals(pc2));

  pc1.addPerson(p1);
  assertFalse(pc1.equals(pc2));

  pc2.addPerson(p2);
  assertFalse(pc1.persons[0] == pc2.persons[0]);
  assertTrue(pc1.persons[0].equals(pc2.persons[0]));
  assertTrue(pc1.equals(pc2));

  pc1.addPerson(p3); pc2.addPerson(p4);
  assertTrue(pc1.persons[1] == pc2.persons[1]);
  assertTrue(pc1.persons[1].equals(pc2.persons[1]));
  assertTrue(pc1.equals(pc2));

  pc1.addPerson(new Person("A", "a", 175, 1.75));
  pc2.addPerson(new Person("A", "a", 165, 1.55));
  assertFalse(pc1.persons[2] == pc2.persons[2]);
  assertFalse(pc1.persons[2].equals(pc2.persons[2]));
  assertFalse(pc1.equals(pc2));
}
```

T

F

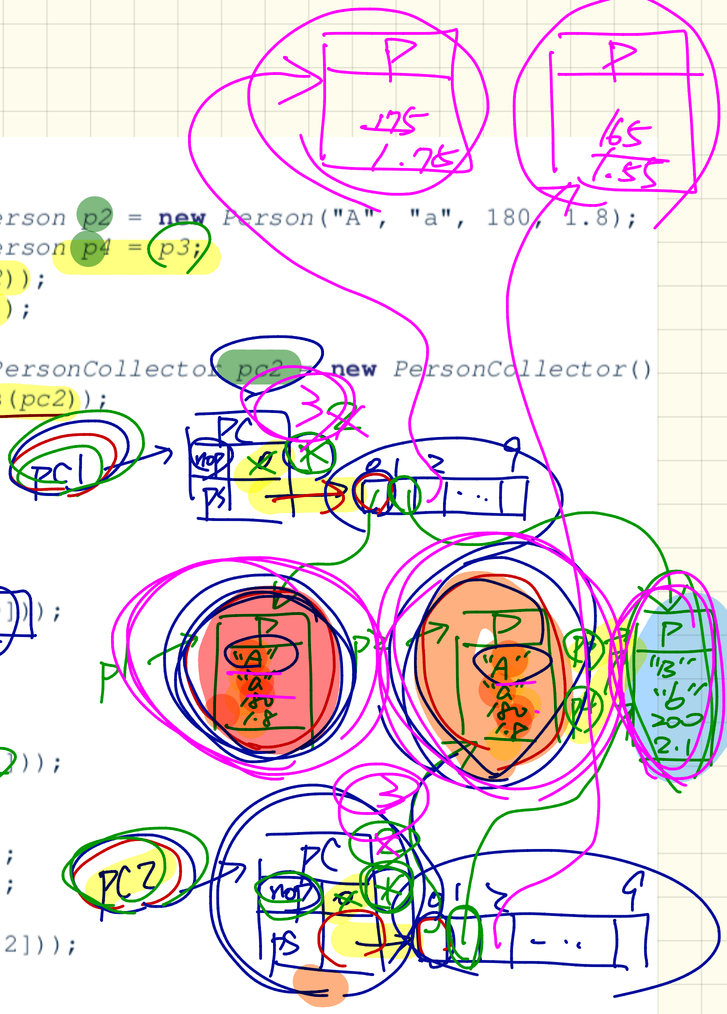
T

T

T

F

F



Employees:

name	id	salary
alan	2	4500.34
mark	3	3450.67
tom	1	3450.67

Sorting based on id's:

tom alan mark

Sorting based on salaries and id's.

emp larger comes first

smaller if id smaller smaller comes first

alan tom mark