EECS2030 Advanced Object-Oriented Programming

Review Tutorials on OOP in Java - Building an Apple Refurbished Store App

Instructor: Jackie Wang

Created: Fall 2021

EECS2030-F21 Lectures Site

https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html#EECS2030_F21

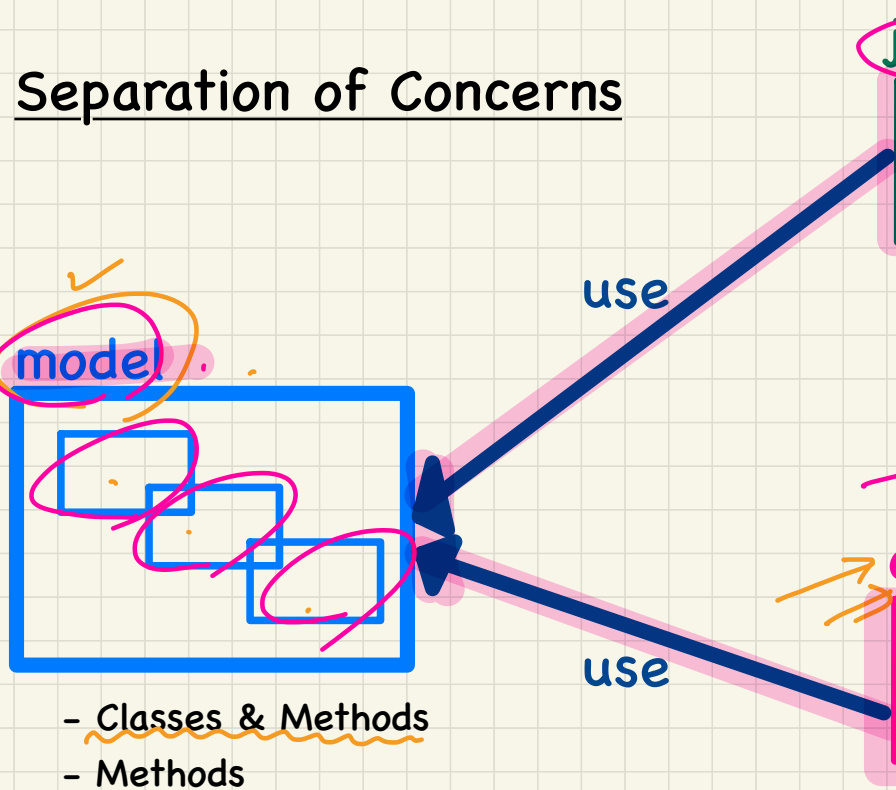
Resources

- EECS1022-W21 Lecture Recordings
- Review Slides on OOP (cross-referenced throughout the current tutorial series)
  + Weeks 7, 8, 9 of EECS1022-W21 lecture recordings
- Written Notes:
  + Inferring Classes/Methods from JUnit Tests
  + Declaring and Manipulating Reference-Typed, Multi-Valued Attributes
- Background Slides on Elementary Programming, Conditionals, Loops
  + Weeks 1 to 6 of EECS1022-W21 lecture recordings
- EECS1022-W21 Tutorials
  + Week 1: Eclipse Work Environment
  + Week 2c, 2d, 2e: Debugger → step over, into, out
  + Weeks 2, 3: Conditionals
  + Weeks 4, 5: Loops and Arrays
  + Weeks 6, 7, 8: OOP
  + Weeks 10, 11: Two-Dimensional Arrays
  + Week 12: Java API - ArrayList vs. Hashtable
- Github (educational account; private repositories)

Tips for Studying this Tutorial Series

- Type with me (watch out for spellings).
- Pay extra attention to **visualizations**, **tracing**, and **debugger**.
- Pause when you need to think.
- Replay if necessary.
- You will be required to submit the code.

Separation of Concerns

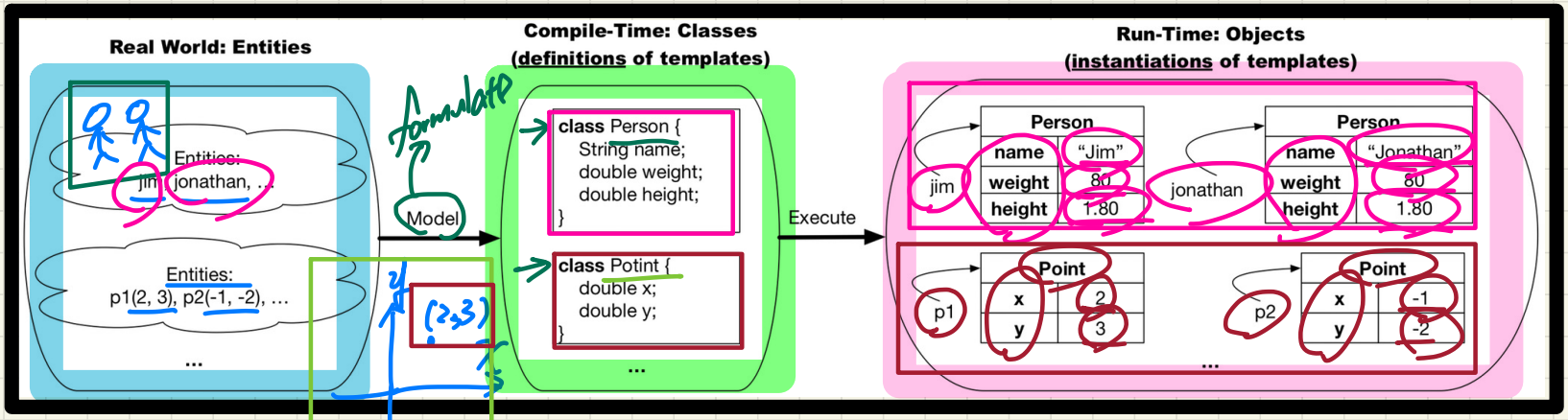


- Expected vs. Actual Values
- Methods
  * calling methods from model
  * assertions
  * containing no print statements

- Classes & Methods
- Methods
  * constructors
  * accessors: return statements
  * mutators: no return statements
  * containing no print statements

- main method (entry point of execution)
  * reading inputs from keyboard
  * calling methods from model
  * producing outputs to console (print)
  * containing no return statements

Observe-Model-Execute Process

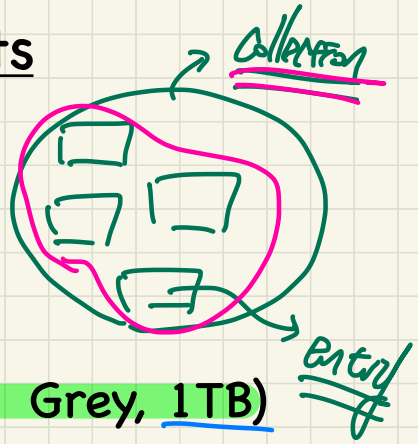


observe

model

based on

Problem: A Refurbished Shop of Apple Products



- https://www.apple.com/ca/shop/refurbished
- Product:
  - + unique serial number (e.g., F9FDN4NKQ1GC)
  - + model, finish, storage (e.g., iPad Pro 12.9, Space Grey, 1TB)
  - + cellular connectivity?, original price, discount value
- Entry: a pair of serial number and its associated product
- Refurbished Store: a collection of entries
  - + add/remove entries
  - + get the stored collection of entries/products
  - + get the associated product/model of a serial number
  - + get serial numbers of products satisfying some criteria

no two iPads have the same SN.

Visualization: Object Creations

false p == p2 (are they referring to same obj?)
true p != p2.

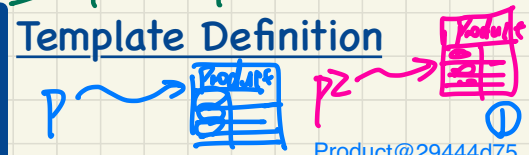
Computer Memory

```
public class Product {
    private String model;
    private String finish;
    private int storage;
    private boolean hasCellularConnectivity;
    private double originalPrice;
    private double discountValue;

    /* constructors */
    public Product() {
        // do nothing
    }

    // An overloaded version of the constructor.
    public Product(String model, double originalPrice) {
        this.model = model;
        this.originalPrice = originalPrice;
    }
}
```

attributes (structure of runtime objects)

object being created.

Template Definition

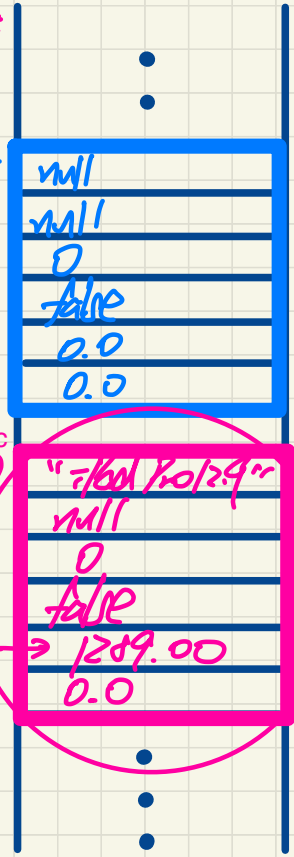


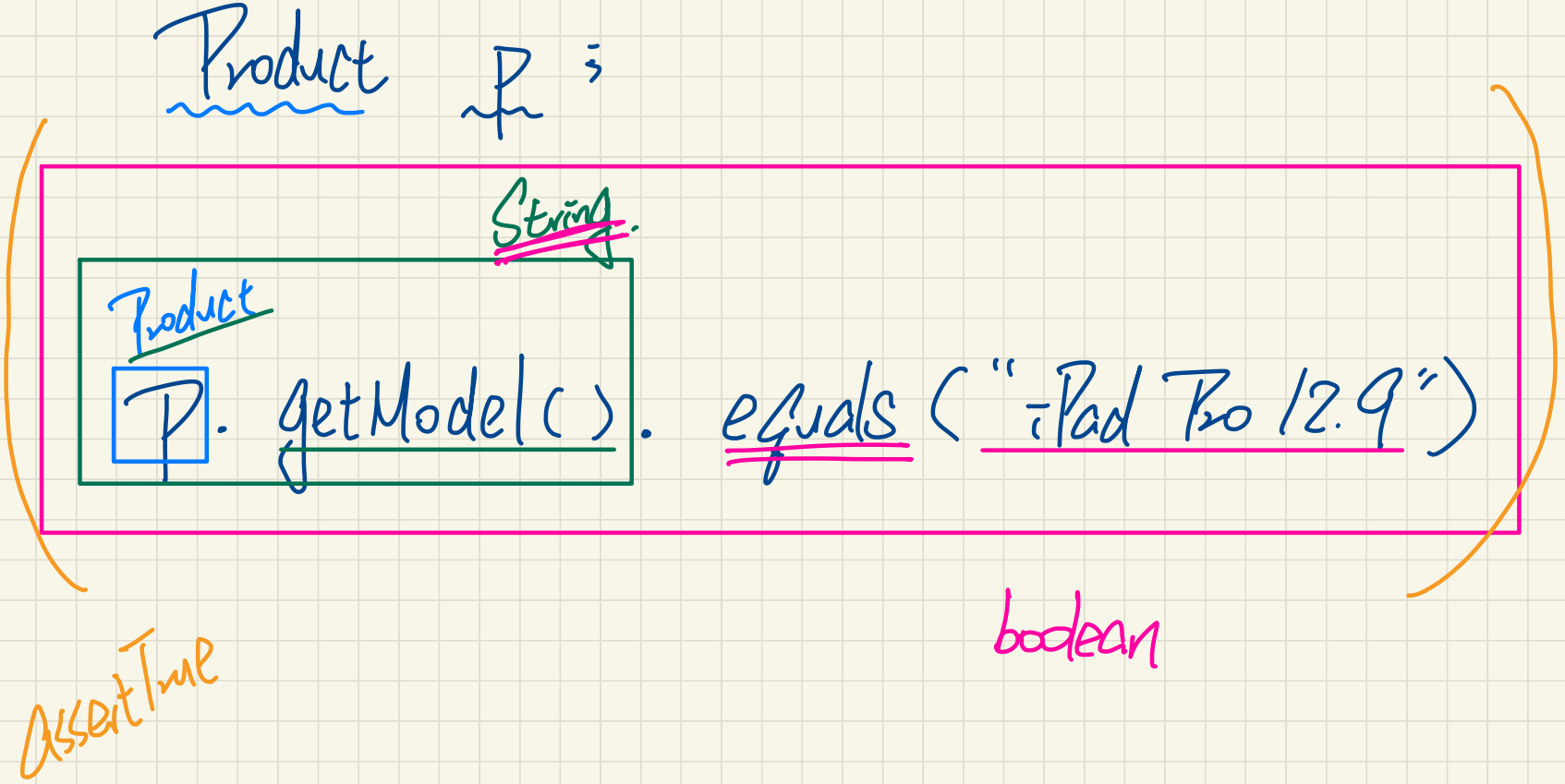
stores the address of same Product object

Product@2280cdac

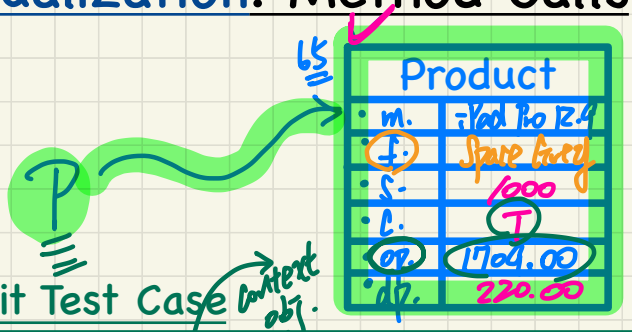
Instance Creation

```
public class ProductApp {

    public static void main(String[] args) {

        Product p = new Product();
        System.out.println(p);

        Product p2 = new Product("iPad Pro 12.9", 1289.00);
        System.out.println(p2);
    }
}
```




Product p;
String
Product
p. getModel(). equals("iPad Pro 12.9")
boolean
assertTrue

Visualization: Method Calls

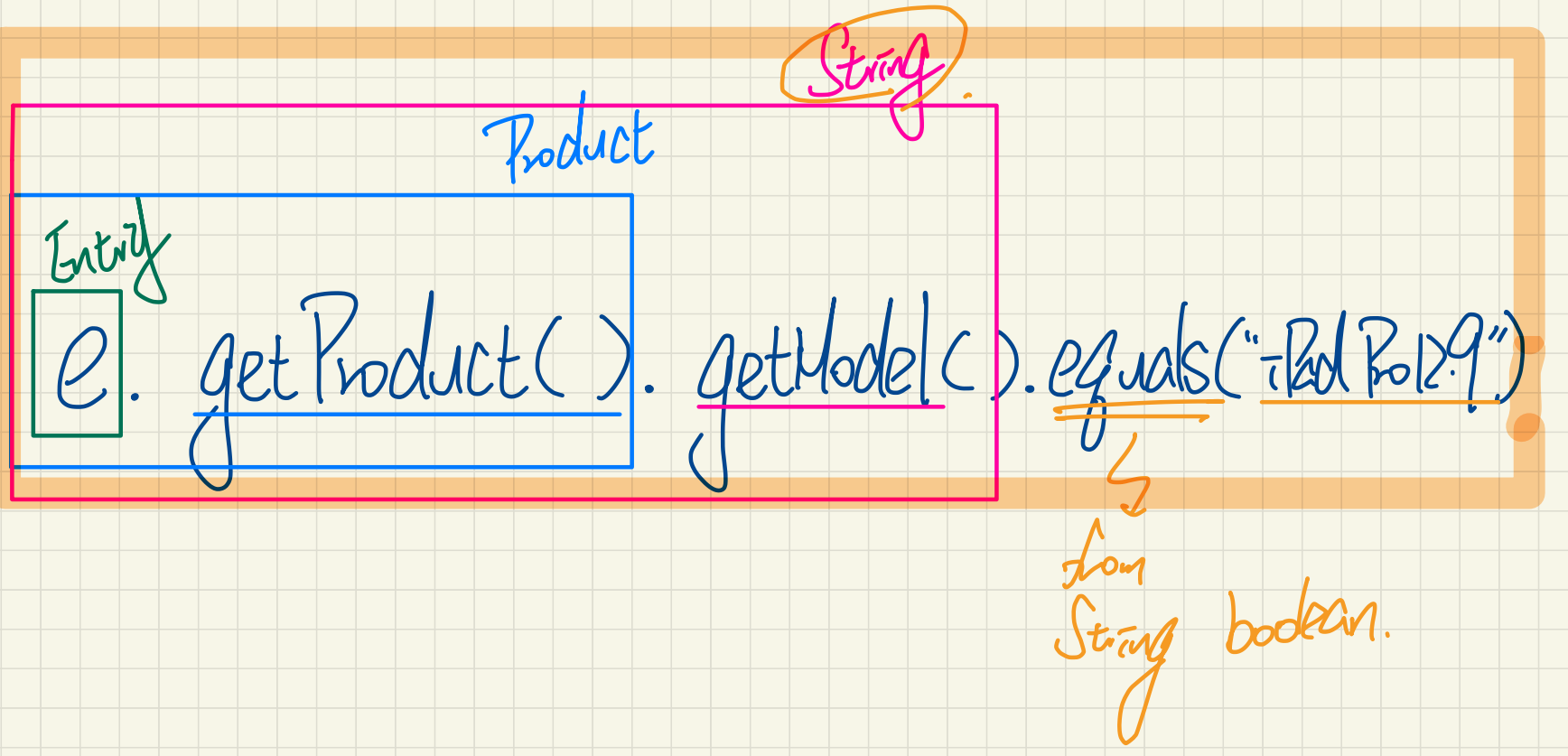


JUnit Test Case

```
@Test
public void test_product_3() {
    Product p = new Product(new String("iPad Pro 12.9"), 1709.00);
    assertNotNull(p.getModel());
    assertEquals("iPad Pro 12.9", p.getModel());
    assertTrue(p.getModel().equals("iPad Pro 12.9"));
    p.setFinish("Space Grey");
    assertFalse(p.getFinish() == null);
    assertTrue(p.getFinish() != null);
    assertEquals("Space Grey", p.getFinish());
    assertTrue(p.getFinish().equals("Space Grey"));
    p.setStorage(1000);
    assertTrue(p.getStorage() == 1000);
    assertEquals(1000, p.getStorage());
    p.setHasCellularConnectivity(true);
    assertFalse(!p.hasCellularConnectivity());
    assertFalse(p.hasCellularConnectivity() == false);
    assertTrue(p.hasCellularConnectivity() == true);
    assertTrue(p.hasCellularConnectivity() != false);
    assertTrue(!p.hasCellularConnectivity() == false));
    assertTrue(!p.hasCellularConnectivity()));
    assertEquals(1709.00, p.getOriginalPrice(), 0.1);
    p.setDiscountValue(220.00);
    assertEquals(220.00, p.getDiscountValue(), 0.1);
    assertEquals(1489.00, p.getPrice(), 0.1);
    assertEquals("iPad Pro 12.9 Space Grey 1000GB (cellular connectivity: true): $(1709.00 - 220.00)", p.toString());
}
```

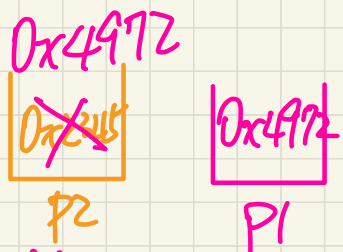
Template Definition

```
public class Product {
    private String model;
    private String finish;
    private int storage;
    private boolean hasCellularConnectivity;
    private double originalPrice;
    private double discountValue;

    public Product(String model, double originalPrice) {
        this.model = model;
        this.originalPrice = originalPrice;
    }

    public void setFinish(String finish) { this.finish = finish; }
    public void setStorage(int storage) { this.storage = storage; }
    public void setHasCellularConnectivity(boolean hasCellularConnectivity) {
        this.hasCellularConnectivity = hasCellularConnectivity;
    }
    public void setOriginalPrice(double originalPrice) { this.originalPrice = originalPrice; }
    public void setDiscountValue(double discountValue) { this.discountValue = discountValue; }

    public String getModel() { return this.model; }
    public String getFinish() { return finish; }
    public int getStorage() { return storage; }
    public boolean hasCellularConnectivity() { return hasCellularConnectivity; }
    public double getOriginalPrice() { return originalPrice; }
    public double getDiscountValue() { return discountValue; }
    public double getPrice() {
        double price = 0.0;
        price = this.originalPrice - this.discountValue;
        return price;
    }
    public String toString() {
        String s = "";
        s += model + " " + finish + " " + storage + "GB "
           + "(cellular connectivity: " + hasCellularConnectivity + "): $("
           + String.format("%.2f", originalPrice) + " - "
           + String.format("%.2f", discountValue) + ")";
        return s;
    }
}
```


String
Product
Entry
e. getProduct( ). getModel( ). equals("iPad Pro12.9")
from
String boolean.

Value Copying

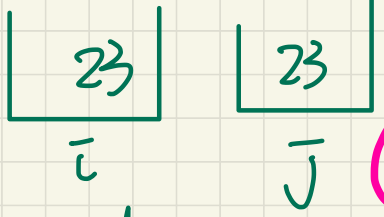
Aliasing: address of an object stored in multiple variables.



Case 1: Primitive Variables

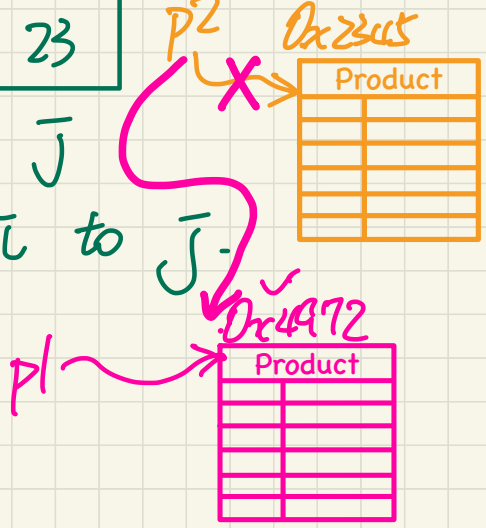
✓ `int i = 23;`

✓ `int j = i;`



Copy value stored in `i` to `j`.

Case 2: Reference Variable

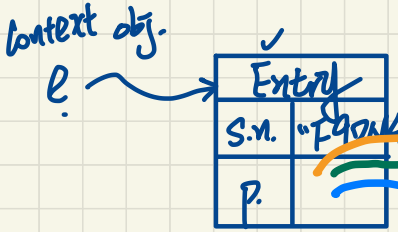


✓ `Product p1 = new Product();`

`Product p2 = new Product();`

`p2 = p1;` → copy address stored in `p1` to `p2`

Visualization: Aliasing



Template Definition

```
public class Entry {
    private String serialNumber;
    private Product product;

    public Entry(String serialNumber, Product product) {
        this.serialNumber = serialNumber;
        this.product = product;
    }

    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    public void setProduct(String model, double originalPrice) {
        //this.product = new Product(model, originalPrice);
        Product p = new Product(model, originalPrice);
        this.product = p;
    }

    public String toString() {
        return "[" + this.serialNumber + "]" + " " + this.product.toString();
    }
}
```

JUnit Test Case

```
@Test
public void test_entry_2() {
    Product p = new Product("iPad Pro 12.9", 1709.00);
    p.setFinish("Space Grey");
    p.setStorage(1000);
    p.setHasCellularConnectivity(true);
    p.setDiscountValue(220.00);

    Entry e = new Entry("F9DN4NKQ1GC", p);
    assertEquals("F9DN4NKQ1GC", e.getSerialNumber());
    assertTrue(e.getProduct() == p);
    assertSame(e.getProduct(), p);
    assertEquals("[F9DN4NKQ1GC] iPad Pro 12.9 Space Grey 1000GB (cellular connectivity: true): $(1709.00 - 220.00)", e.toString());

    Product p2 = new Product("iPad Air", 649.00);
    p2.setFinish("Gold");
    p2.setStorage(64);
    p2.setHasCellularConnectivity(false);
    p2.setDiscountValue(100.00);

    e.setProduct(p2);

    assertEquals("F9DN4NKQ1GC", e.getSerialNumber());
    assertFalse(e.getProduct() == p);
    assertNotSame(e.getProduct(), p);
    assertTrue(e.getProduct() == p2);
    assertSame(e.getProduct(), p2);
    assertEquals("[F9DN4NKQ1GC] iPad Air Gold 64GB (cellular connectivity: false): $(649.00 - 100.00)", e.toString());

    e.setProduct("iPad Air", 649.00);

    assertEquals("F9DN4NKQ1GC", e.getSerialNumber());
    assertFalse(e.getProduct() == p);
    assertNotSame(e.getProduct(), p);
    assertFalse(e.getProduct() == p2);
    assertNotSame(e.getProduct(), p2);
    assertEquals("[F9DN4NKQ1GC] iPad Air null 0GB (cellular connectivity: false): $(649.00 - 0.00)", e.toString());
}
```

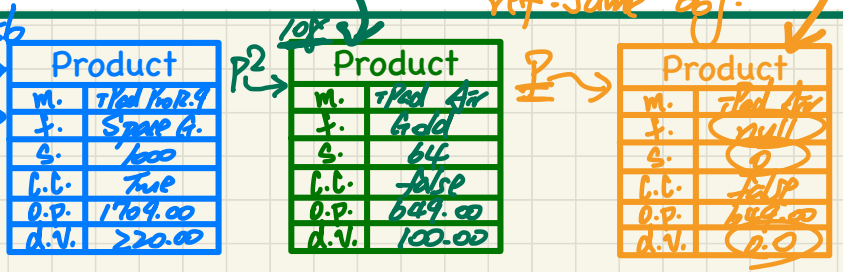
aliasing — both p and e.product reference same obj.

↳ r.v. from e.product.toString()

aliasing — both p2 and e.product ref. same obj.

↳ r.v. from e.product.toString()

aliasing — both p and e.product ref. same obj.



Visualization: Adding Entries

aliasing: e1, rs.es[0], rs.getEntries()[0]

JUnit Test Case

```
RefurbishedStore rs = new RefurbishedStore();

Product p1 = new Product("iPad Pro 12.9", 1709.00);
p1.setFinish("Space Grey");
p1.setStorage(1000);
p1.setHasCellularConnectivity(true);
p1.setDiscountValue(220.00);
Entry e1 = new Entry("F9FDN4NKQ1GC", p1);
rs.addEntry(e1); // Add entry 1.

Product p2 = new Product("iPad Air", 649.00);
p2.setFinish("Gold");
p2.setStorage(64);
p2.setHasCellularConnectivity(false);
p2.setDiscountValue(100.00);
rs.addEntry("C9FZN4J8QC82", p2) // Add entry 2.

rs.addEntry("7YM4PFZ779UB", "iPad Pro 10.5", 929.00);

assertTrue(rs.getNumberOfEntries() == 3);
assertTrue(rs.getPrivateEntriesArray().length == 5);
assertTrue(rs.getPrivateEntriesArray()[0] == e1);
assertTrue(rs.getPrivateEntriesArray()[1].getSerialNumber().equals("C9FZN4J8QC82"));
assertTrue(rs.getPrivateEntriesArray()[1].getProduct() == p2);
assertTrue(rs.getPrivateEntriesArray()[2].getSerialNumber().equals("7YM4PFZ779UB"));
assertTrue(rs.getPrivateEntriesArray()[2].getProduct().getModel().equals("iPad Pro 10.5"));
assertEquals(929.00, rs.getPrivateEntriesArray()[2].getProduct().getOriginalPrice(), 0.1);
assertTrue(rs.getPrivateEntriesArray()[3] == null);
assertTrue(rs.getPrivateEntriesArray()[4] == null);

assertTrue(rs.getEntries().length == 3);
assertTrue(rs.getEntries()[0] == e1);
assertTrue(rs.getEntries()[1].getSerialNumber().equals("C9FZN4J8QC82"));
assertTrue(rs.getEntries()[1].getProduct() == p2);
assertTrue(rs.getEntries()[2].getSerialNumber().equals("7YM4PFZ779UB"));
assertTrue(rs.getEntries()[2].getProduct().getModel().equals("iPad Pro 10.5"));
assertEquals(929.00, rs.getEntries()[2].getProduct().getOriginalPrice(), 0.1);
```

Template Definition

```
public class RefurbishedStore {
    private Entry[] entries;
    private int noe;
    private final int MAX_CAPACITY = 5;

    public RefurbishedStore() {
        this.entries = new Entry[MAX_CAPACITY];
        this.noe = 0;
    }

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

    public void addEntry(String sn, Product p) {
        this.addEntry(new Entry(sn, p));
    }

    public void addEntry(String sn, String model, double originalPrice) {
        this.addEntry(new Entry(sn, new Product(model, originalPrice)));
    }

    public int getNumberOfEntries() {
        return this.noe;
    }

    public Entry[] getPrivateEntriesArray() {
        return this.entries;
    }

    public Entry[] getEntries() {
        Entry[] es = new Entry[this.noe];
        for(int i = 0; i < this.noe; i ++) {
            es[i] = this.entries[i];
        }
        return es;
    }
}
```

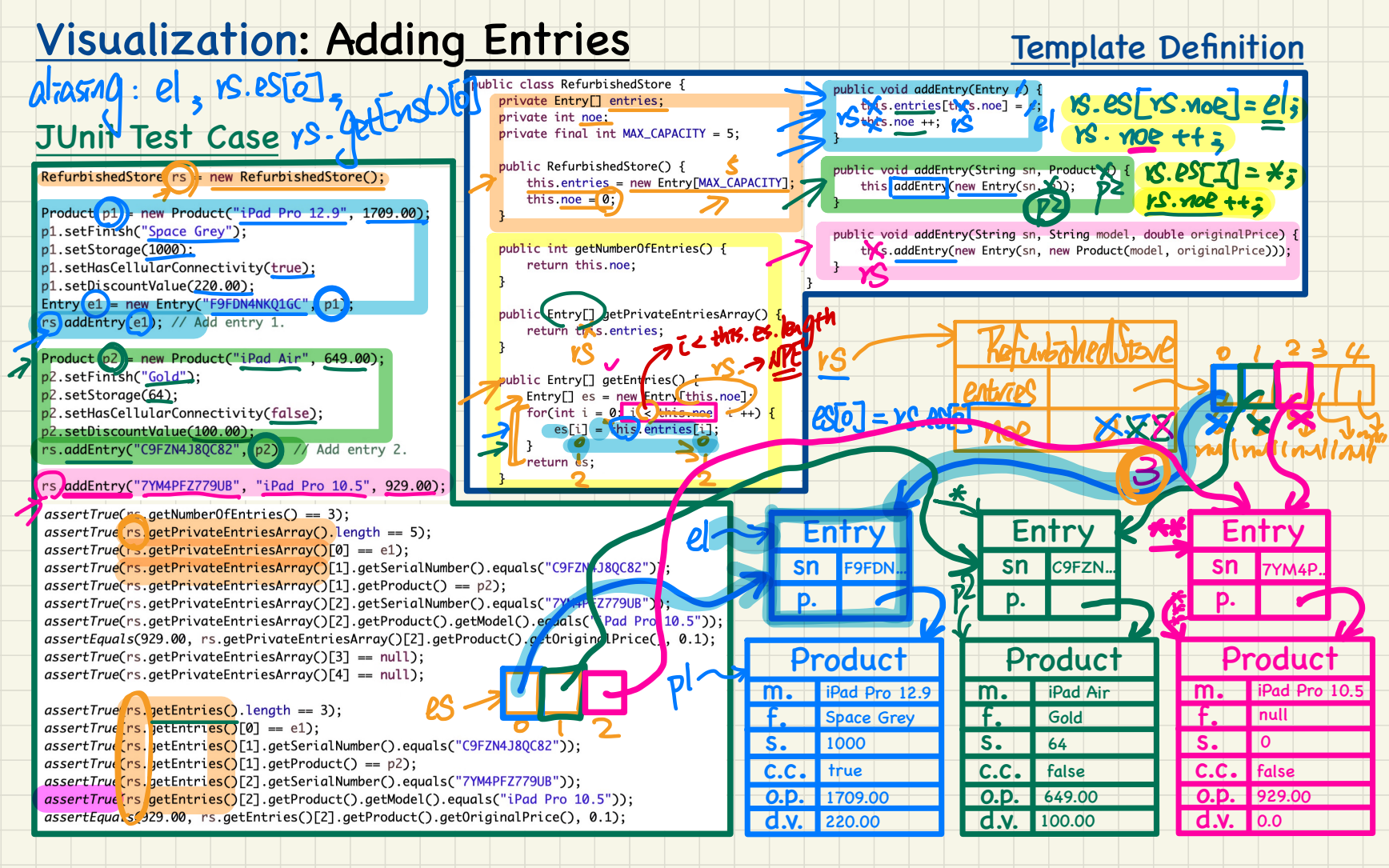
rs.es[rs.noe] = e1; rs.noe ++;

rs.es[i] = *; rs.noe ++;

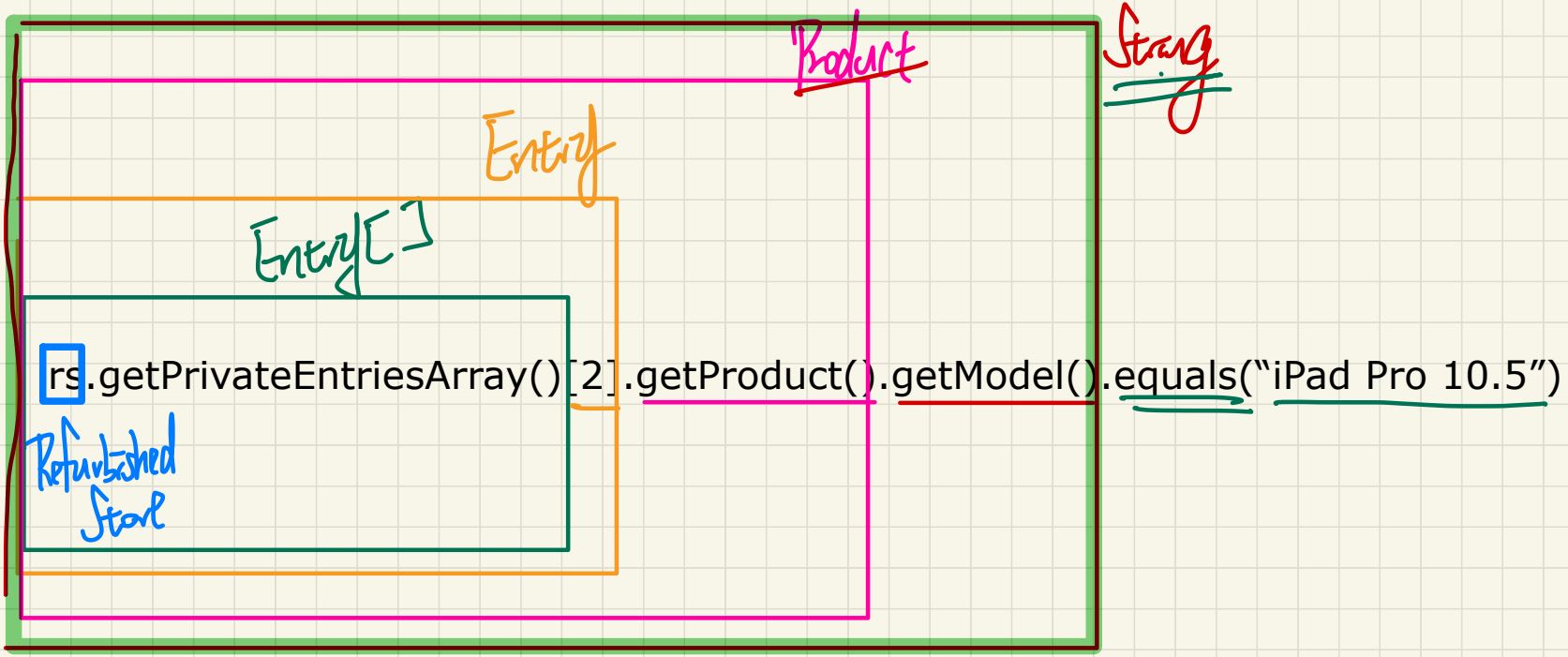
i < this.es.length

rs -> NPE

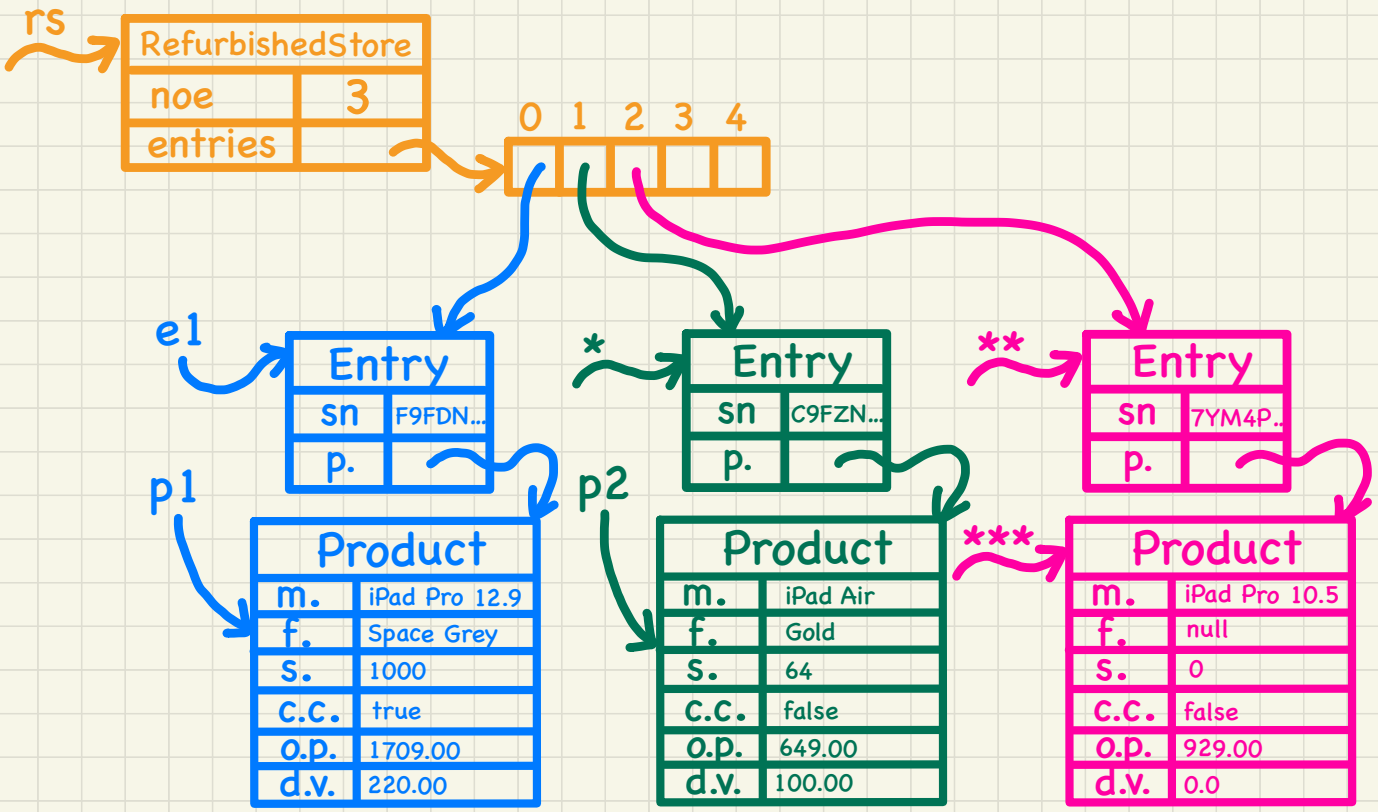
es[0] = rs.es[0]



Analyzing Dot Notation



Visualization of a Refurbished Store with 3 Entries



RefurbishedStore Class: getSpaceGreyOrPro()

What is the max # of products satisfying the search criteria?