


EECS2030 Advanced Object-Oriented Programming
(Fall 2021)

Q&A - Review Tutorial Part 1

Wednesday, September 15

assertTrue( _____) ;

-  names of classes & methods → exact
- follow the assertions shown in videos.
- put assertions on your own.

Announcement

- LabOP1 (due: Sep. 17)
- LabOP2 (due: Sep. 24)
- Lecture W3 (released: Sep. 20)
- Lab1 (released: Sep. 22)
- Written Test (due: Sep. 30 - Oct. 1)

Methods

1. Constructors (for creating new objects)

2. Accessors / Getters - inquiring info. on the context obj.

P. getBMI()
C.O. acc./get

3. Mutators / Setters. - modify att. values of the C.O.
P. setWeight(109);

```
public class TestProduct{
```

```
    @Test
```

```
    public void test_p_1() {
```

```
        ;
```

```
    }
```

```
    public void test_p_2() {
```

```
        ;
```

```
    }
```

```
}
```

the associated method should be executed as a test case.

When running a class as a JUnit Test class, only those methods with a @Test tag will be executed for their assertions.

I am a bit confused as to why creating an accessor called "toString()" works. Like how does the compiler know that now, every time we try to print 'p' it should print what is returned instead. If you could elaborate this a bit more it would be great, thanks!

default behaviour:
return address
of this.

Object ^{toString equals hashCode}

each class is a child class of

I've been thinking about this question and I'm wondering if perhaps it is similar to how constructors work for classes.

Object, inheriting all its methods.

I read that every object has a toString built in method so I'm wondering if I'm on the right track.

1. Every class has an implicit toString method defined (it's inherited from Object class).

v1: default behaviour of toString

```
class Person {  
    int age;  
}
```

```
Person jim = new Person();
```

```
System.out.println(jim);
```

as if: \downarrow `jim.toString()`

↳ toString is not redefined

in Person \Rightarrow address stored in jim is printed.

v2: redefined behaviour of toString

```
class Person {  
    int age;  
    redefined  
     $\rightarrow$  public String toString() {  
        return age + " years old";  
    }
```

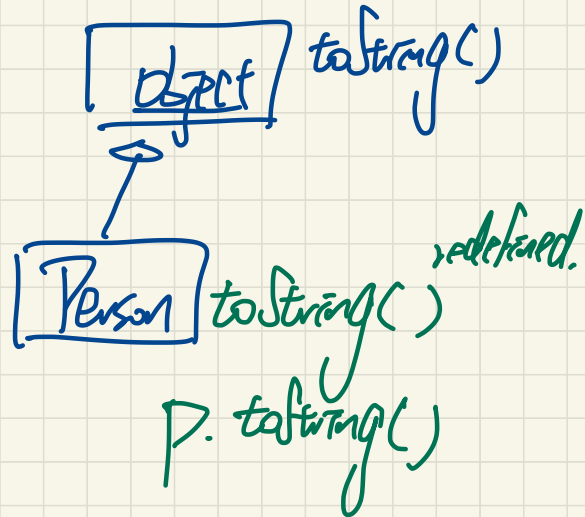
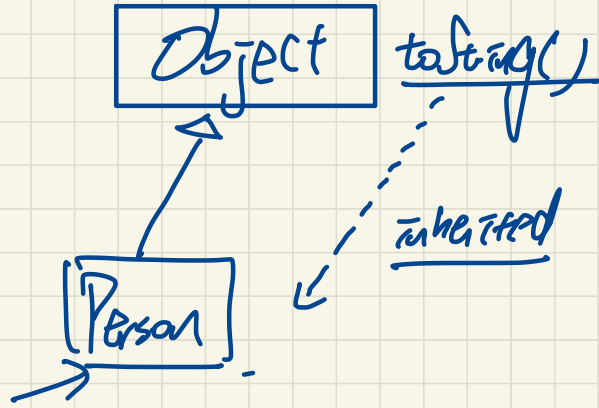
```
Person jim = new Person();
```

```
    jim.setAge(23);  
    System.out.println(jim);
```

↳ `jim.toString()`

↳ redefined version called.

```
class Person {  
:  
}
```



String toString() {

return ~~age~~ = 2;

variable assignment.

}

expecting some value
String

Hello professor. I'm a little bit confused about the functionality of `'StringBuilder'` in the `toString` method.

What benefit does it have and what can we do with it that we can't do with just creating the `String` and `concatenating` it (like the first method we typed for `toString()` in the tutorial). Thank you!

Using `StringBuilder` is `computationally more efficient` than using concatenation. However, as far as this course is concerned, opting for either solution is equally acceptable (as you won't have to deal with large-sized strings).

Implementing an Accessor/Getter Returning String

space is wasted in intermediate computation.

1. Use concatenation op. (+).

```
String s = "Hello" + "World" + "There";
```

use this return if the string is large-sized.

2. StringBuilder class @ "Hello"

② Perform: "Hello" + "World"

③ "H W" + "There" → "Hello World"

3. String.format (→ "H..W..T") intermediate result.

↳ from C.

is complex.

↳ use it when the pattern of return value

Does p actually store the address of the product object 'p' or does it point to the address of the object in memory?

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
    }

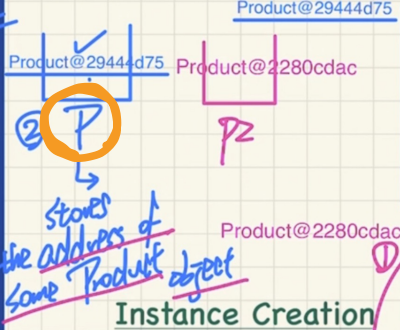
    // An overloaded version of the constructor.
    public Product(String model, double originalPrice) {
        this.model = model;
        this.originalPrice = originalPrice;
    }
}
```

Attributes (Structure of runtime objects)

iPad Pro 12.9 1289.00

object being created.

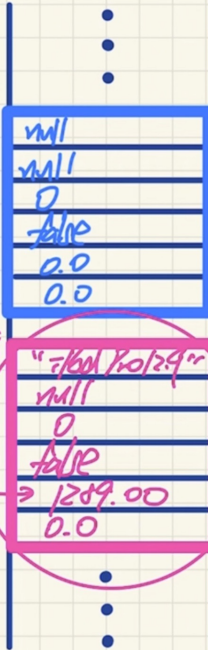
Template Definition



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);
    }
}
```



Ref. Abstraction

v1 = v2

- same type
- ref type

Hello Prof. Jackie. Why do we use p for testing the overloaded constructor?

Shouldn't we use p2 or p3, as defined on ProductApp?

By using p on all test cases, is that not going to interfere with the p defined for the default constructor?

The variable p declared within each test method has its scope (and thus is only visible) within that method, so there is no interference. That is, variable p in test_product_1 is distinct from variable p in test_product_2.

```

class Product {
    String model;
    public Product() { --- }
    public setModel(Product p) {
        this.model ≠ p.model;
    }
}

```

```

class TestProduct {

```

```

    @Test
    public void test_1() {
        Product p = new Product();
        Product p2 = new Product();

```

```

        ...
        p.setModel(p2);
        ...
    }

```

① After executing each method, all refs ↑

local vars are flushed out (reset).

↳ will replace 'p' in setModel ⇒ no interference!

```

    @Test
    public void test_2() {
        Product p = new Product();
        Product p2 = new Product();
        ...
        p.setModel(p2);
    }

```

}

```
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;
}
```

P →

Product	
m.	null
f.	null
s.	0
c.c.	false
o.p.	0.0
d.v.	0.0

p2 →

Product		
m.	null	1289.00
f.	null	
s.	0	
c.c.	false	
o.p.	0.0	1289.00
d.v.	0.0	

```
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);
    }
}
```