

EECS1022 Programming for Mobile Computing
(Winter 2021)

Q&A - Lectures W5

Monday, February 22

Tue 9:30 am EST.

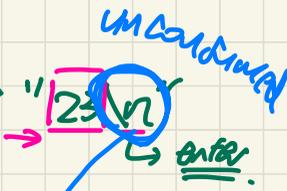
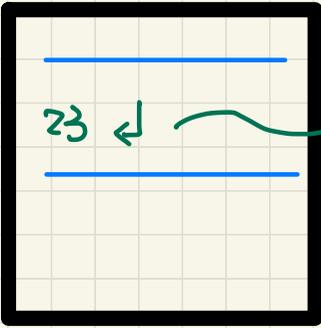
Agenda

- Lectures W5 Q&A (GoogleDoc)
- Lab5 (inferring classes from tests; Counter example)
- Programming Test 2 Practice (Eclipse)
- + Lab4 Problems
- + Practice Test Problems
- + coding bat (<https://codingbat.com/java>)
 - * Array-2 withoutTen
 - * Array-2 tenRun

Can you please give a brief explanation as to why we have to write "consume" the new line character after getting an integer input?

```
public class ReadingIntegerString {  
  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
  
        → System.out.println("Enter an integer:");  
        → int i = input.nextInt();  
  
        → System.out.println("Enter your name:");  
        String name = input.nextLine();  
  
        System.out.println(name + ", you entered: " + i);  
  
        input.close();  
  
    }  
}
```

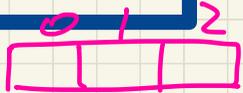
Console



Member m1 = ~~new Member()~~; GoldenMember

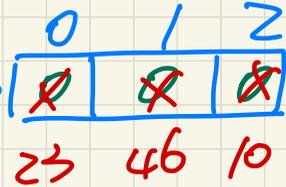
Is there a reason as to why we have to redeclare the type when we do discrete assignments for arrays? ↑

int[] a = new int[3]; → not declaration! never possible.



boolean[] a = new boolean[4]; X

↓
declared already.



a[0] = 23;
a[1] = 46;
a[2] = 10;

int[] b = { 1, 2, 3 };



→ a = b; * reassignment *

~~import java.util.Arrays; Arrays.sort(-X)~~

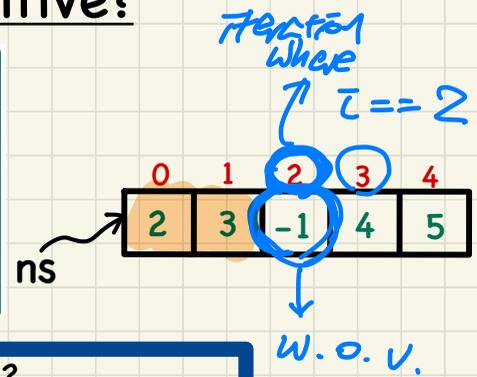
Computational Problem: Are All Numbers Positive?

```

1 int[] ns = {2, 3, -1, 4, 5};
2 boolean soFarOnlyPosNums = true;
3 int i = 0;
4 while (i < ns.length) {
5   soFarOnlyPosNums = soFarOnlyPosNums && (ns[i] > 0);
6   i = i + 1;
7 }

```

Version 1



shouldn't the soFarOnlyPosNums var be false starting when i=2, not i=3?
 (Perhaps I'm reading the timing of the var assignments/reassignments incorrectly and the soFarOnlyPosNums column gives the var's value only as the iteration begins?)

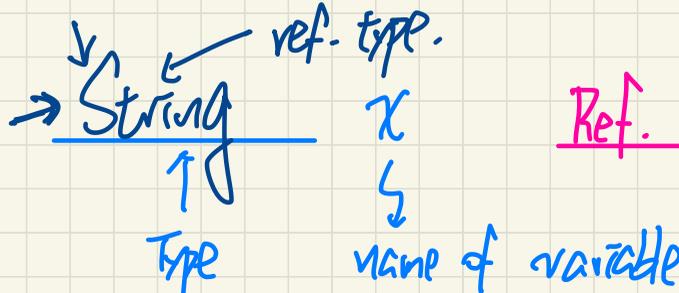
<i>i</i>	<i>soFarOnlyPosNums</i>	<i>i < ns.length</i>	stay?	<i>ns[i]</i>	<i>ns[i] > 0</i>
0	true	true	YES	2	true
1	true	true	YES	3	true
2	true	true	YES	-1	false
3	false	true	YES	4	true
4	false	true	YES	5	true
5	false	false	No	-	-

You brought up how the reason why we have to capitalize "String" when declaring the type is because String is a class.

Could you maybe explain a little bit more about this?

I was curious as to why all other types are lowercase except for String.

Declaration of Variable



1. Primitive

Prim. Type var

int (l)

at RT, stops int value only.

Ref. Type var

Member (m)

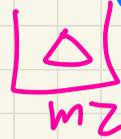
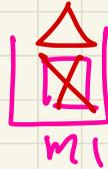
23
l

int, boolean, char, double

2. Reference (non-primitive & there's a corresponding class).

Ref. Type Var

① Object Creation.



Member $m = \text{new Member}(\text{"jini"});$

Member $m \Rightarrow$



- not store an entire object.
- stores the address of some object of type Member.

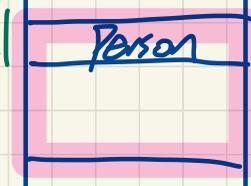
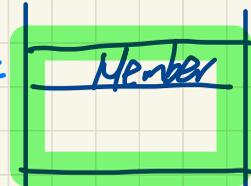
② Member $m1 = \text{new Member}();$

Member $m2 = \text{new$

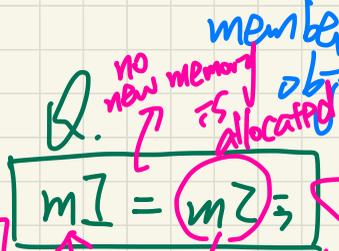
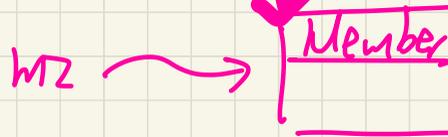
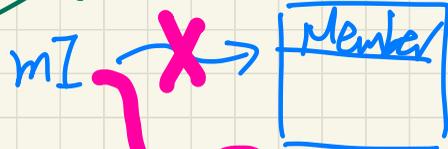
Member();

Member $m = 0x1101 \Rightarrow X$

0x1101



aliasing



Q. no new memory is allocated.

member object address

direct 0x1231 address assignment of object into m_1 .

Copy address stored in $m_2 \rightarrow$ address of object not allowed.

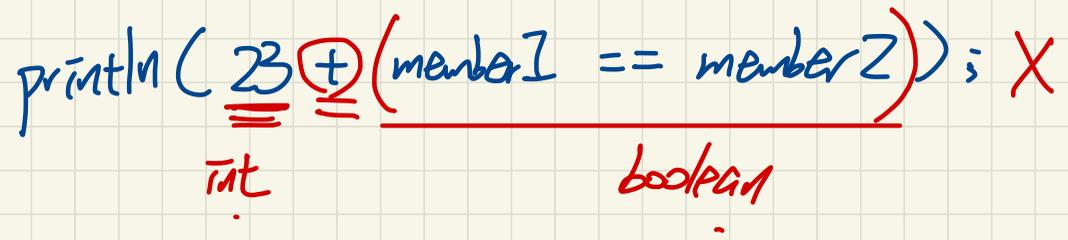
```

7 public static void main(String[] args) {
8     System.out.println("Before creating two members...");
9     Member member1 = new Member(); // default constructor
10    System.out.println("After creating member 1...");
11    Member member2 = new Member();
12    System.out.println("After creating member 2...");
13    System.out.println("Member1 and Member2 are the same object: " + member1 == member2);
14    System.out.println("Member1 and Member2 are distinct object: " + member1 != member2);
15 }

```

Str.
Concat - higher precedence
Str X Member
ST
boolean

Lines 13 and 14 made me wonder what precedence the concatenation operator has compared to relational operators (i.e. <, >, !=). Does it have the same precedence as addition since the symbol is overloaded?



```
public class Member {  
    private int id;  
    private char type;  
    private double balance;
```

```
    private String name;
```

```
    private double weight;  
    private double height;
```

```
→ public Member(int id, char type, double balance) {  
    this.id = id;  
    this.type = type;  
    this.balance = balance;  
}
```

shadowing — "clash of names" between method and

parameters
class-level variables.

```
public Member(int id_input) {  
    id = id_input;  
}
```

How can you have a **parameter** for the class's Constructor and have it declared the same variable name as an **Attribute var** in the same class? Wouldn't the **scope** of the class's Constructor's method be inside the scope of the class's Attributes? Wouldn't the global scope of the attribute 'id' overlap (and clash) with the local scope of the parameter 'id'?

(while researching this question, I think the answer has to do with **'variable shadowing'** and the keyword **'this'**?)

Visualization: Calling Methods

```
public class Member {
    private int id;
    private char type;
    private double balance;

    private String name;

    private double weight;
    private double height;

    public Member(double weight, double height) {
        this.weight = weight;
        this.height = height;
    }

    public double getWeight() {
        return this.weight;
    }

    public double getHeight() {
        return this.height;
    }

    public String getBMIReport() {
        String result = "";

        double heightInMeters = this.height / 100;
        double bmi = this.weight / (heightInMeters * heightInMeters);

        String interpretation = "";
        if (bmi < 18.5) {
            interpretation = "Underweight";
        }
        else if (bmi < 25.0) {
            interpretation = "Normal";
        }
        else if (bmi < 30.0) {
            interpretation = "Overweight";
        }
        else {
            interpretation = "Obese";
        }

        result = interpretation + " (" + String.format("%.1f", bmi) + ")";

        return result;
    }

    public void changeWeightBy(double units) {
        this.weight += units;
    }
}
```

weight height public String getBMIReport() (double weight) ?
weight
this.weight

weight height public String getBMIReport()
not necessary
(: no parameters
dash with
attributes
weight and height)

Between the components of the class template (Attributes, Constructor, Accessor and Mutator), I realized you can get away with just using the keyword 'this' in the Constructor method(s) - every other component doesn't require the keyword 'this'.
It's not necessary for Access and Mutator methods? Then how do you know which object's attributes you are using?
Is it because when you invoked the method, you called it with the object name in dot notation i.e.alan.getWeight?

```
@Test
public void testMember_04() {
    Member alan = new Member(85, 175);
    Member mark = new Member(101, 181);
    // Initial measures
    assertEquals(85, alan.getWeight(), 0.1);
    assertEquals(101, mark.getWeight(), 0.1);
    assertEquals("Overweight (27.8)", alan.getBMIReport());
    assertEquals("Obese (30.8)", mark.getBMIReport());
    // Change measures
    alan.changeWeightBy(-13);
    mark.changeWeightBy(-13);
    assertEquals("Normal (23.5)", alan.getBMIReport());
    assertEquals("Overweight (26.9)", mark.getBMIReport());
}
```

Member	
id	
type	
balance	
name	
weight	
height	

Visualization: Calling Methods

When declaring a variable of a type certain class, and assigning it the address of an object of that class, are you assigning this variable the starting address of that object, or is this a level of detail we don't need to bother with for this course?

```
public class Member {
    private int id;
    private char type;
    private double balance;

    private String name;

    private double weight;
    private double height;

    public Member(double weight, double height) {
        this.weight = weight;
        this.height = height;
    }

    public double getWeight() {
        return this.weight;
    }

    public double getHeight() {
        return this.height;
    }

    public String getBMIReport() {
        String result = "";

        double heightInMeters = this.height / 100;
        double bmi = this.weight / (heightInMeters * heightInMeters);

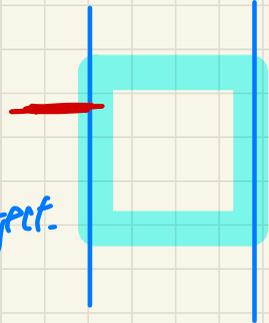
        String interpretation = "";
        if(bmi < 18.5) {
            interpretation = "Underweight";
        }
        else if (bmi < 25.0) {
            interpretation = "Normal";
        }
        else if (bmi < 30.0) {
            interpretation = "Overweight";
        }
        else {
            interpretation = "Obese";
        }

        result = interpretation + " (" + String.format("%.1f", bmi) + ")";

        return result;
    }

    public void changeWeightBy(double units) {
        this.weight += units;
    }
}
```

assign the address of newly-created Member object.



```
@Test
public void testMember_04() {
    Member alan = new Member(85, 175);
    Member mark = new Member(101, 181);
    // Initial measures
    assertEquals(85, alan.getWeight(), 0.1);
    assertEquals(101, mark.getWeight(), 0.1);
    assertEquals("Overweight (27.8)", alan.getBMIReport());
    assertEquals("Obese (30.8)", mark.getBMIReport());
    // Change measures
    alan.changeWeightBy(-13);
    mark.changeWeightBy(-13);
    assertEquals("Normal (23.5)", alan.getBMIReport());
    assertEquals("Overweight (26.9)", mark.getBMIReport());
}
```

Member	
id	
type	
balance	
name	
weight	
height	

Lab5: Test Driven Development (TDD)

junit_tests

Lab 3 PT3

```
public class Tester {  
    @Test  
    public void test() {  
        ... /* create and manipulate objects */  
    }  
}
```

USES

model

```
public class ... {  
    /* attributes */  
    ...  
  
    /* constructors */  
    ...  
  
    /* accessors */  
    ...  
  
    /* mutators */  
    ...  
}
```

```
public class ... {  
    /* attributes */  
    ...  
  
    /* constructors */  
    ...  
  
    /* accessors */  
    ...  
  
    /* mutators */  
    ...  
}
```

Wb tutorials

build class → write tests

notes

Array-2 > withoutTen

[prev](#) | [next](#) | [chance](#)

<https://codingbat.com/prob/p196976>

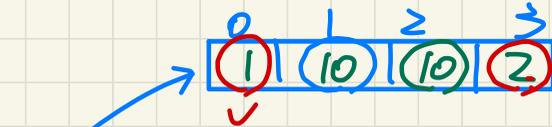
Return a version of the given array where all the 10's have been removed. The remaining elements should shift left towards the start of the array as needed, and the empty spaces at the end of the array should be 0. So {1, 10, 10, 2} yields {1, 2, 0, 0}. You may modify and return the given array or make a new array.

withoutTen([1, 10, 10, 2]) → [1, 2, 0, 0]

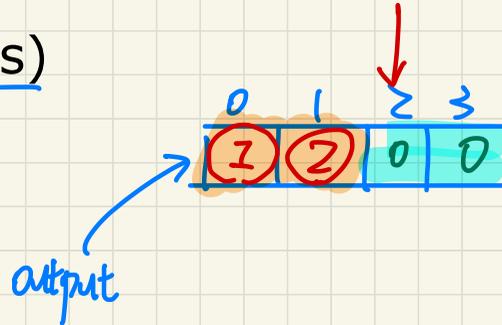
withoutTen([10, 2, 10]) → [2, 0, 0]

withoutTen([1, 99, 10]) → [1, 99, 0]

`int[]` withoutTen(`int[]` nums)



count & ✗ (2)



Array-2 > tenRun

[prev](#) | [next](#) | [chance](#)

<https://codingbat.com/prob/p199484>

For each multiple of 10 in the given array, change all the values following it to be that multiple of 10, until encountering another multiple of 10. So {2, 10, 3, 4, 20, 5} yields {2, 10, 10, 10, 20, 20}.

```
tenRun([2, 10, 3, 4, 20, 5]) → [2, 10, 10, 10, 20, 20]
```

```
tenRun([10, 1, 20, 2]) → [10, 10, 20, 20]
```

```
tenRun([10, 1, 9, 20]) → [10, 10, 10, 20]
```

```
int[] tenRun(int[] nums)
```