

# Lecture 15 - Midterm Review

Bentley James Oakes

March 11, 2018

- First 2 TA tutorials next week
  - Mon 15:00-16:30 and Tue 11:00-12:30
  - In ENGTR 3120. No need to register.
- CSUS Helpdesk hosting a review session
  - Monday, March 12, 2018 from 18:30 to 21:00
  - ENGM C 11
  - RSVP: <https://goo.gl/forms/jbZ4v8GXXKcysjtm2>
  - <https://www.facebook.com/events/149525632383380/>

- Midterm on **March 13th, 18:00 to 21:00**
- If you miss the midterm, your final exam will be worth 65% of your final grade

<b>Room:</b>	<b>Start</b>	<b>End</b>
Leacock 132	Abanda	Lau
Leacock 219	Laxague	Mitsumasu
Leacock 26	Moccetti	Rodrigues, F.
Stewart Biology N2/2	Rodrigues, T.	Sultani
Stewart Biology S1/3	Sun	Xie
Stewart Biology S3/3	Xiong	Zou

No spare seats, so please go to the right room!

## Instructions:

### • DO NOT TURN THIS PAGE UNTIL INSTRUCTED

- This is a **closed book** examination; only a legal-sized (8.5" by 14") **crib sheet** is permitted. This crib sheet can be single or double-sided; it can be handwritten or typed. Non-electronic translation dictionaries are permitted, but instructors and invigilators reserve the right to inspect them at any time during the examination.
- Besides the above, only writing implements (pens, pencils, erasers, pencil sharpeners, etc.) are allowed. The possession of any other tools or devices is prohibited.
- Answer all questions on the provided answer sheets at the end of this exam.
- This examination has **10** pages including this cover page, and is printed on both sides of the paper. On page 10, you will find information about **useful classes and methods**.

## Scoring

The exam will be scored as follows:

1. Questions 1 to 8 are worth 1 point each
2. Questions 9 to 25 are worth 2 points each
3. Question 26 is worth 38 points
4. Total: 80 points

Bring ID (McGill or other government photo ID)

- Documentation of `String`, `print`, and `Math` methods are provided

`String` (package `java.lang`) Methods:

- `public boolean equals(Object anObject)`: Compares this `String` to an `Object`.
- `public int length()`: Calculates the length of this `String`.
- `public char charAt(int i)`: Gets the char at position `i` of the `String`. Note that counting starts from 0 so that to get the first character of the `String` you should input `i` equals 0.
- `public boolean equalsIgnoreCase(String anotherString)`: Compares, ignoring case considerations, this `String` to another `String`.

- Everything up to, and including two-dimensional arrays.
- Binary
- Types, expressions, scope
- Errors and exceptions
- Booleans, conditions, loops
- Methods
- String and char
- Arrays and references

Binary numbers are numbers in base 2, meaning that each digit is either 0 or 1. We normally count in base 10, so we have digits from 0-9.

What is 10110 (binary) in decimal notation (base 10)?

$$10110 = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$$

$$10110 = 16 + 0 + 4 + 2 + 0$$

$$10110 = 22$$

What is 29 in binary?

- Go through the powers of two and subtract if you can
- If you subtract, add 1 to the binary number, otherwise add 0

	16	8	4	2	1	Powers of two
29						Start
13	1					16 fits into 29, remainder 13
5		1				8 fits into 13, remainder 5
1			1			4 fits into 5, remainder 1
1				0		2 does not fit into 3
0					1	1 fits into 1, remainder 0

Answer is 11101



A variable is a named place in memory that is used to store a value.

Variables have:

- A name
- A type
  - What kind of value is stored here?
- A scope
  - The parts of the program where the variable exists

# Assignment Statements

Assignment statements are used to store a value inside a variable

- Assignment statements use a '='
- On the left must be a variable
- On the right must be an expression
- Example: `x = 5 + (3*2);`
- Example: `day = "Monday";`

Variables must be initialized (assigned for the first time), before they can be used. Otherwise, you will get a compile-time error!

Expressions represent a computation that Java will make  
Each variable/method call is replaced by its value, operators are applied,  
and the value of the expression is computed

Type casting, arithmetic, comparisons, equality...

If the operators have the same priority, they are applied from left to right.

Assignments happen last

Note that `+=`, `-=`, etc count as assignment operators.

# Order of Operations

In Java, operations in a single statement happen in the following order from left to right:

- 1 Brackets
- 2 Multiplication, division, and mod (\*, /, %)
- 3 Addition and subtraction (+, -)
- 4 Relational (<, <=, etc)
- 5 Equality (==, !=)
- 6 Logical AND (&&)
- 7 Logical OR (||)

# String Concatenation

From Midterm Fall 2016:

```
public static void main(String[] args){  
    int x = 4;  
    int y = 5;  
    System.out.println(x + " + " + y + " is " + x + y);  
}
```

4 + 5 is 45

- Recall that + between numerical values is addition
- But + with a String on one or two sides becomes String concatenation

```
public static void main(String[] args){  
    int x = 4;  
    int y = 5;  
    System.out.println(x + y + " is " + x + y);  
}
```

9 is 45

- The + operator is evaluated left-to-right
- So it does addition first, then concatenation after that

What is the value (and type) of x?

1  $x = 5 + 2 * 3$

■ 11 as an int

2  $x = 3/2 + 8\%3$

■ 3 as an int

3  $x = -3/2 + 8\%3$

■ 1 as an int

4  $x = \text{"Hello"} + 2 * 2$

■ *Hello4 as a String*

5  $x = (4 == 2 + 2)$

■ *true as a boolean*

The scope of a variable is the part of the code where the variable exists (where it can be accessed).

A *variable* only exists inside of the method in which it is declared.

A variable exists from when it is declared to the next close of a block.



# Scope Example

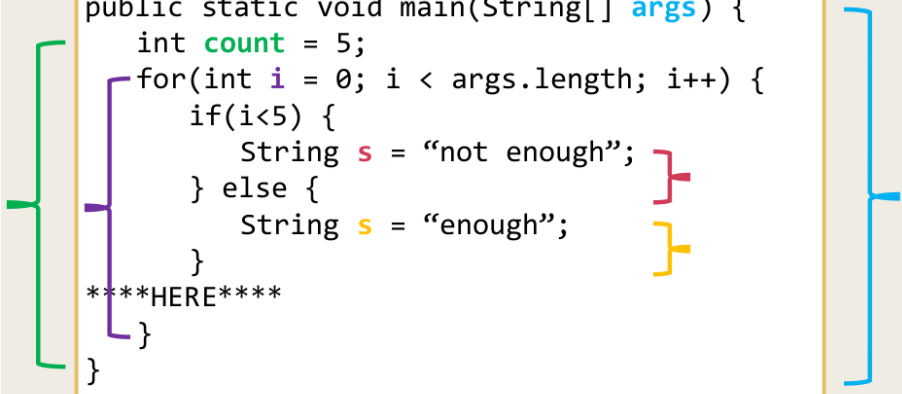
Does the variable x exist on the following lines?

```
public static void main(String[] args){  
  
    int y = 5;  
    //LINE 1  
    for (int i=0; i< 4; i++){  
        //LINE 2  
        int x = 4;  
        if (y > 12){  
            //LINE 3  
            x++;  
        }  
        //LINE 4  
    }  
    //LINE 5  
}
```

- Line 1: No
- Line 2: No
- Line 3: Yes
- Line 4: Yes
- Line 5: No

# Scope Example

What variables exist on the line marked **\*\*\*\*HERE\*\*\*\***?



```
public static void main(String[] args) {  
    int count = 5;  
    for(int i = 0; i < args.length; i++) {  
        if(i < 5) {  
            String s = "not enough";  
        } else {  
            String s = "enough";  
        }  
        ****HERE****  
    }  
}
```

The diagram illustrates variable scope using colored brackets:

  - A green bracket on the left spans the entire `main` method, indicating that `args` and `count` are in scope throughout.
  - A purple bracket on the left spans the `for` loop, indicating that `i` is in scope within the loop.
  - A red bracket on the right spans the `if` block, indicating that `s` is in scope within the `if` block.
  - A yellow bracket on the right spans the `else` block, indicating that `s` is in scope within the `else` block.

`args`, `count`, `i`

We can convert values between different types using type casting.

- `int x = (int) 5.4; //x = 5`
- `double z = (double) 98; //z = 98.0`
- `double w = (double) x; //w=5.0`

Type casting does not work to convert a `String` to a primitive data type and vice versa.

Use `Integer.parseInt` or `Double.parseDouble`

```
int n = Integer.parseInt("53"); //n=53
```

# Type Conversion

- Type casting is not always necessary
  - If there's no possible loss of information, Java automatically makes the conversion
  - If loss of information is possible and you don't use type casting, then there is a compile-time error!
- `int x = 5.4; // compile-time error`
- `double w = 5; // automatic conversion (int to double)`
- `char a = 99.7; // compile-time error!`
- `int n = Integer.parseInt(53); // compile-time error!`
- `double m = Integer.parseInt("53"); // automatic conversion (int to double)`
- `int n = Double.parseDouble("53"); // compile-time error!`
- `int n = Integer.parseInt("cat"); // run-time error!`

# The Components of Iteration

- Creation - We create an **iteration variable** and assign it a starting value
- Condition/test - We define the test for how many times the loop should run
- Modification - We change the iteration variable

# The For Loop

- We iterate so much in programming, we have a special loop for it

```
//the three parts of a for loop:  
//initialization - int k=0  
//condition/test - k < 5  
//iteration      - k++  
for (int k=0 ; k<5 ; k++)  
{  
    System.out.println("K: " + k);  
}
```

- It's the three components of **iteration**
- We just put them on one line
- And have two semi-colons in there

## From Winter 2017

21. How many times does the letter *A* print when the following code executes?

```
for(int i=0; i<5; i++){  
    System.out.println("A");  
    for(int j=0; j<10; j+=2){  
        System.out.println("A");  
        for(int k=10; k>0; k--){  
            System.out.println("A");  
        }  
    }  
    System.out.println("A");  
}
```

From Winter 2017

21. How many times does the letter A print when the following code executes?

```
for(int i=0; i<5; i++){ 5 times
    System.out.println("A"); 5
    for(int j=0; j<10; j+=2){ 5 times
        System.out.println("A"); 5·5
        for(int k=10; k>0; k--){ 10 times
            System.out.println("A"); 5·5·10
        }
    }
    System.out.println("A"); 5
}
```

$5 + 5 + 25 + 250 = 285$



From Fall 2017

18. What will the following code print?

```
public static void main(String[] args) {  
    int count = 0;  
    for(int i=0; i<4; i++){  
        count++;  
        for(int j=0; j<i; j+=2){  
            count++;  
            for(int k=j; k<i; k++){  
                count++;  
            }  
        }  
    }  
    System.out.println(count);  
}
```

From Fall 2017

18. What will the following code print?

```
public static void main(String[] args) {  
    int count = 0;  
    for(int i=0; i<4; i++){ 0 1 2 3  
        count++;  
        for(int j=0; j<i; j+=2){ 0 0 0 2  
            count++;  
            for(int k=j; k<i; k++){ 0 0 1 0 1 2 2  
                count++;  
            }  
        }  
    }  
    } 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
    System.out.println(count);  
}
```

15

# Comparing Characters

If you want to compare chars use characters and not their ASCII value.

For instance, the following condition checks whether the char-type variable letter is a digit between 0 and 5:

```
letter >= '0' && letter <= '5'
```

- Methods can **return a value** (maximum 1) or they can be **void** (return no value)
- Methods can take **no input** or any number of **inputs**.
- If a method is void, a **return statement** is not necessary, and you cannot return a value
- If a method is not void, then there must be at least one return statement. The type of the value returned must match the type declared in the method header.
- There can be multiple return statements in one method

- From the **method header** we can tell if the method is void or what type of value it will return. We can also tell how many inputs the method takes and their type.
  - `public static void firstMethod(int x, double y)`
  - `public static int otherMethod(String s)`
- Classes without main: you can compile it, BUT you cannot run it

## From Midterm Fall 2016:

14. Consider the following method header:

```
public static void method(double x, int y)
```

Which of the following are *valid* ways to use this method? Write the *letters* corresponding to the correct answer(s) on your answer sheet. Note that there are five options total. Two are on the next page.

- (a) `System.out.println(method(4,4));`
- (b) `String s = "apple"`  
`method(Math.pow(2,3), s.length());`
- (c) `double x=1.2;`  
`int y=7;`  
`method(double x, int y);`
- (d) `method(2.5, Math.sqrt(4));`
- (e) `method(4, 4);`

B and E

## From Midterm Winter 2017:

17. Which of the following are valid ways to call the method whose header is

```
public static double thingOne(int x, double y)
```

Write the letters corresponding to *all* correct answers on your answer sheet.

A `int a = 5;`  
`a = thingOne(a, a);`

B `System.out.println(thingOne(1, 4.5));`

C `double b = 5.0;`  
`b = thingOne(b, b);`

D `double b = thingOne(Math.sqrt(16.0), 5.5);`

E `double x = 1;`  
`int y = 2;`  
`double z = thingOne(x, y);`

➤ B

What is wrong with the other options?

- A. `thingOne` returns a double which can't be stored inside an `int` without explicit typecasting.
- C. The first input should be an `int`. The conversion from `double` to `int` is not automatic.
- D. `Math.sqrt()` returns a `double`, and the first input is an `int`.
- E. Once again, `x` is a `double` not an `int`.

# Multiple Return Values

```
public static String getSymbol(int x){  
    if(x == 1) {  
        return "Cherries";  
    } else if (x == 2) {  
        return "Oranges";  
    } else if (x == 3) {  
        return "Plums";  
    } else if (x == 4) {  
        return "Bells";  
    } else if (x == 5) {  
        return "Melons";  
    } else if (x == 6) {  
        return "Bars";  
    } else {  
        return "ERROR";  
    }  
}
```



# Return Values

```
public static void main(String[] args){  
  
    int x = first();  
    double y = second();  
    System.out.println("x: " + x);  
    System.out.println("y: " + y);  
}  
  
public static int first(){  
    int x = 6;  
    int y = 12;  
    return y;  
}  
public static double second(){  
    return (double) first();  
}
```

x: 12

y: 12.0

```
public static void main(String[] args){
```

```
    int x = 4;
```

```
    int y = 15;
```

```
    y = swap(x, y);
```

```
    System.out.println("3-x: " + x);
```

```
    System.out.println("3-y: " + y);
```

```
}
```

1-x: 4

1-y: 15

2-x: 15

```
public static int swap(int x, int y){
```

```
    System.out.println("1-x: " + x);
```

```
    System.out.println("1-y: " + y);
```

```
    int temp = x;
```

```
    x = y;
```

```
    y = temp;
```

```
    System.out.println("2-x: " + x);
```

```
    System.out.println("2-y: " + y);
```

```
    return y;
```

```
}
```

2-y: 4

3-x: 4

3-y: 4

## Conditions

Series 1:

```
if(condition1) {  
    \\Do something  
}  
if(condition2){  
    \\Do something  
}  
else {  
    \\Do something else  
}
```

Series 2:

```
if(condition1) {  
    \\Do something  
}  
else if(condition2){  
    \\Do something else  
}  
else {  
    \\Do another something else  
}
```

What is the difference between the above two condition series?

- ▶ In Series 1, one or two blocks will execute. In Series 2, exactly one block will execute.

What prints?

```
String s = ""; String t = "";  
for(int i=0; i < 10; i++){  
    s = s + i;  
    t = i + t;  
}  
System.out.println(s + t);
```

01234567899876543210

# Exercise

What prints?

```
int j = 10;
while(j>0){

    System.out.print("cats ");
    j-=2;

    if(j<8){
        j=-1;
        System.out.println("dogs");
    }
}
```

cats cats dogs

- Compile-time errors
  - Something wrong with the *syntax* of the program or type checking
  - Examples: Missing `;`, storing 5.4 in an int
- Run-time errors
  - Something wrong during the execution of a program
  - Occur during running, not compilation
  - Examples: Dividing by zero, accessing array out of bounds
- Style errors
  - Something that makes your program hard to read and maintain.
  - Examples: Bad variable names, missing indentation
- Logic errors
  - The program does the wrong thing
  - Examples: Returning the wrong value

# Error Question

From Midterm Fall 2016:

Which of the following are examples of *compile-time* errors?

- 1 Accessing an array at index -1
- 2 Dividing an integer by 0
- 3 Passing an int value as input to a method that expects a double input argument
- 4 Accessing a variable outside of its scope
- 5 Omitting a semi-colon at the end of a statement

- 1 Run-time error (ArrayIndexOutOfBoundsException)
- 2 Run-time error (ArithmeticException)
- 3 No error (automatic conversion)
- 4 Compile-time error
- 5 Compile-time error

## From Midterm Winter 2017:

15. Which of the following statements (or group of statements) will cause compile-time errors? Write the letters corresponding to *all* correct answers on your answer sheet.

A `String s = Integer.parseInt("twenty");`

B `int result = 35/0;`

```
C public static void main (String[] args) {  
    String day = "Tuesday";  
    today();  
}  
public static void today() {  
    System.out.println("Today is " + day);  
}
```

```
D int[][] a = new int[3][];  
   System.out.println(a[0]);
```

```
E int[] a = {1,2,3};  
   int x = a[-1];
```

A and C



```
for (int i=0; i < 5; i+=2){  
    for (int j=0; j <= i; j++){  
        System.out.println("Bubblegum");  
    }  
}
```

How many times is the condition in the *inner-loop* evaluated?

How many times does the text *Bubblegum* get printed?

1 12 times

- Outer loop iterates 3 times:  $i = 0, 2, 4$
- Inner loop iterates 1 times when  $i = 0$ , 3 times when  $i = 2$ , and 5 times when  $i = 4$
- Condition is evaluated at end of each iteration

2  $1 + 3 + 5 = 9$  printings of *Bubblegum*

An array references a list of values

The value of an array variable is the memory address of that list

- The values in the array are called elements
- The list of elements is indexed, the order matters
- The length of the array is established when the array is created and it cannot be changed
- All the elements of an array must be of the same type

To create an array, we first have to declare a variable with an array type:

- `String[] days;`
- `int[] grades;`
- `double[] heights;`
- `String[] args;`

- Can assign values to all elements in the array as soon as we declare the variable using curly brackets: `String[] pets = {"Cats", "Dogs", "Ferrets"};`
- Need to know the length and values of the array when creating it.

- Can use the new operator to create an array of a certain size.
- Can then changes entries one at a time, later in the program.
- These examples both create an array of String with length 3.

```
String[] pets = new String[3];  
String[] pets;  
pets = new String[3];
```

Java assigns default values to each position in the array when it is created using the `new` keyword

- For integer/double arrays: 0
- For String arrays: null
- For boolean arrays: false.

- Declaring an array
  - `int[] a;`
- Initializing that array
  - `a = new int[8];`
- Declaring and initializing an array
  - `int[] b = {1,2,3};`
  - `int[] c = new int [6];`

26. What are the contents of the array `a` after the following lines of code execute?

```
int[] a = {1,2,3};  
method(a[0],a[1],a[2]);
```

```
//Below is the entire body of the method called on the last line above.  
public static void method(int a, int b, int c){  
    int x = a;  
    a = b;  
    b = c;  
    c = x;  
}
```



26. What are the contents of the array `a` after the following lines of code execute?

```
int[] a = {1,2,3};  
method(a[0],a[1],a[2]);
```

```
//Below is the entire body of the method called on the last line above.  
public static void method(int a, int b, int c){  
    int x = a;  
    a = b;  
    b = c;  
    c = x;  
}
```

■ [1, 2, 3]

Rough work

`a[0]`, `a[1]`, `a[2]` are integers. No matter what method does, it cannot make changes to the elements of the array referenced by `a` since it does not take a reference as input. Therefore, the content of the array `a` does not change.

# Exercise

```
public static void main(String[] args) {  
    int[] arr1 = {2, 3};  
    int[] arr2 = {1, 4};  
    foo(arr1, arr2);  
    System.out.print(arr1[0] + " " + arr1[1] + " " + arr2[0] + " " +  
        arr2[1]);  
}  
  
public static void foo(int[] a, int[] b) {  
    a[0] = b[1];  
    a[1] = b[0];  
    b[0] = a[1];  
    b[1] = a[0];  
}
```

# Exercise

```
public static void main(String[] args) {
    int[] arr1 = {2, 3};
    int[] arr2 = {1, 4};
    foo(arr1, arr2);
    System.out.print(arr1[0] + " " + arr1[1] + " " + arr2[0] + " " +
        arr2[1]);
}

public static void foo(int[] a, int[] b) {
    a[0] = b[1];
    a[1] = b[0];
    b[0] = a[1];
    b[1] = a[0];
}
```

What prints?

■ 4 1 1 4

Rough work

arr1, arr2 are references, the elements of the arrays might be changed by the method.

a and arr1 both point to {2,3},  
b and arr2 both point to {1,4}

- 1) a[0]=b[1] → {**4**,3} {1,4}
- 2) a[1]=b[0] → {4,**1**} {1,4}
- 3) b[0]=a[1] → {4,1} {**1**,4}
- 4) b[1]=a[0] → {4,1} {1,**4**}

```
//create the array
int[][] arr = {{1,2,3},{4,5,6}};

int[] first = arr[0];
int x = first[1];

int y = arr[1][2];
System.out.println(x + " " + y);
//what prints?
```

2 6

What prints?

```
int [][] arr = {{2,4,6,8},{3,6},{1,2,3}};  
  
for(int i=0; i<arr.length; i++){  
    for(int j=0; j<arr[i].length; j++){  
        System.out.print(arr[i][j] + "  
");  
    }  
}
```

2 4 3 6 1 2

## From Midterm Winter 2017

25. What prints? (We are not interested in the specific formatting of `deepToString`)

```
public static void main(String[] args) {  
    int[] a = {1,2,3};  
    int[] b = {9,8,7};  
    int[][] c = {a,b};  
    method(c);  
    System.out.println(Arrays.deepToString(c));  
}  
public static void method(int[][] a){  
    int temp = a[0][1];  
    a[0][1] = a[1][2];  
    a[1][2] = temp;  
}
```

`[[1, 7, 3], [9, 8, 2]]`

```
public static void main(String[] args) {  
    int[] a = {1,2,3};  
    int[] b = {9,8,7};  
    int[][] c = {a,b};  
    swap(c[0][1], c[1][2]);  
    System.out.println(Arrays.deepToString(c));  
}  
  
public static void swap(int x, int y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}
```

[[1, 2, 3], [9, 8, 7]]

```
int[] a = {1, 2, 3};
```

```
System.out.println("Array a: " + a);
```

What prints?

Array a: [I@7347c5f3

This is the **address** of the data within a



```
int[] a = {1, 2, 3};
```

Address	Variable Type	ID	Value
1171...	int []	a	@7347...
...			
7347...	int	a[0]	1
7347...	int	a[1]	2
7347...	int	a[2]	3

- The array variable stores the address where the data starts
- When we index, we are looking up the address in the computer's memory

```
int[] a = {1, 2, 3, 4, 5};  
int[] b = a;  
b[0] = 22;  
System.out.println(a[0]);
```

22

```
public static void main(String[] args){  
    int[] a = {1,2,3,4,5};  
    change(a);  
    System.out.println(a[0]);  
}  
public static void change(int[] arr){  
    arr[0] = 5;  
}
```

```
public static void main(String[] args){  
    int[] b = {1,2,3,4,5};  
    change(b);  
    System.out.println(b[0]);  
}  
public static void change(int[] arr){  
    arr = new int[5];  
    arr[0] = 55;  
}
```

- Strings are *immutable reference types*
- You cannot change the value of a `String` after you have created it

When you try to modify the value of a `String` variable, Java creates a new `String`, and puts the new value there.

```
public static void main(String[] args){  
    String s = "abcdef";  
    change(s);  
    System.out.println(s);  
}  
public static void change(String t){  
    t = t + "ghijk";  
}
```

*abcdef*

# Strings as Immutable Reference Types

You can use the `s.charAt(i)` method to access the character in `String` `s` at index `i`

However, you cannot use that method (or any method) to change a character in a `String` at some index

You must build a new `String` yourself using a *for-loop*

- 1 Write a method `getMaxIndex` that takes an array of integers as input and returns the index of the largest element inside the array.
  - `int[] a = {-2, 5, 1, 9};`
  - `int[] b = {2, 10, -12, 5};`
  - `getMaxIndex(a)` returns 3, and `getMaxIndex(b)` returns 1
- 2 Write a method `isIncreasing` that takes as input an array of integers and returns true if the elements appear in the array in an increasing order, false otherwise. Example: consider the following arrays
  - `int[] a = {-2, 1, 1, 5, 9};`
  - `int[] b = {2, 10, -12, 12};`
  - `isIncreasing(a)` returns true, and `isIncreasing(b)` returns false



- 1 Write a method `getAverage` that takes as input a 1-dimensional array of doubles. The method should compute and return the average of all the values stored in the input array.
  - `double[] grades = {78.0, 91.0, 72.5};`
  - `getAverages(grades)` should return 80.5.
- 2 Write a method `getAverage` that takes as input a 2-dimensional array of doubles where each subarray represents the grades of a student. Compute the average of each student and return an array of doubles containing all the averages.
  - `double[][] grades = {{82.5, 89.0},{78.0, 91.0, 72.5},{73.25}};`
  - `getAverages(grades)` should return `{85.75, 80.5, 73.25}`.