(Lab 5)
Methods, Classes, & APIs: Overloading

P3 Solutions limited in scope to:		
● P1 Concepts ● P2 Concepts	● Methods Declarations ○ Pass data in ○ Return data out ○ Method overloading	● Method Invocations ○ Use Java API methods ○ Build API Util classes

Submission Rules:

1. Submissions must be zipped into a **handin.zip** file. Each problem must be implemented in its own class file. Use the name of the problem as the class name.

2. You must use standard input and standard output for ALL your problems. It means that the input should be entered from the keyboard while the output will be displayed on the screen.

3. Your source code files should include a comment at the beginning including your name and that problem number/name.

4. The output of your solutions must be formatted exactly as the sample output to receive full credit for that submission.

5. Compile & test your solutions before submitting.

6. Each problem is worth up to 10 points total. The breakdown is as follows: 2 points for compiling, 3 points for correct output with sample inputs, 5 points for additional inputs.

7. You will get full marks if you get 40 marks or more. *To get bonus marks, you must solve problems 7 & 8.*

8. Submission:

- You have unlimited submission attempts until the deadline passes
- You'll receive your lab grade immediately after submitting

Problem 1: Datatype Util (10 points) Make API

(API design) Java contains several primitive data types built into the programming language. All algorithms fundamentally rely on these types to model all possible software objects. You're tasked to create a Datatype utility class that allows a developer to get the type of data for any primitive value.

Recall that Java is an extensible language, which means you can expand the programming language with new functionality by adding new classes. Utility classes are typically helper classes that contain a collection of related static methods. For example, Math is a utility class.

DataType Util Method API:

Modifier and Type	Method and Description
`static String`	**`getType`**`(double data)` Returns "double" as a String
`static String`	**`getType`**`(float data)` Returns "float" as a String
`static String`	**`getType`**`(int data)` Returns "int" as a String
`static String`	**`getType`**`(long data)` Returns "long" as a String
`static String`	**`getType`**`(char data)` Returns "char" as a String
`static String`	**`getType`**`(boolean data)` Returns "boolean" as a String
`static String`	**`getType`**`(String data)` Returns "String" as a String

Facts

- Your **`DatatypeUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`DatatypeUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`DatatypeUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`getType(1.0);`
`getType(1.0f);`
`getType(1);`
`getType(1L);`
`getType('1');`
`getType(true);`
`getType("Hello");`
	`"double"`
	`"float"`
	`"int"`
	`"long"`
	`"char"`
	`"boolean"`
	`"String"`

Problem 2: Radix Util (10 points) Make API

(API design) In mathematical numeral systems, the radix or base is the number of unique digits, including the digit zero, used to represent numbers in a positional numeral system. For example, in the decimal system (base 10), uses ten digits from 0 through 9. But it is not uncommon in computer science to also use base 2, base 8, base 16 numbers. You're tasked to implement a Radix Utility class for Java that

includes the following API *(Application Programming Interface)*. Recall that Java is an extensible language, which means you can expand the programming language with new functionality by adding new classes. Utility classes are typically helper classes that contain a collection of related static methods. For example, Math is a utility class.

Radix Util Method API:

Modifier and Type	Method and Description
`static int`	**`base2`**`(String binary)` Returns decimal integer value given a String with a binary representation
`static String`	**`base2`**`(int decimal)` Returns a String with binary representation given a decimal integer value
`static int`	**`base8`**`(String octal)` Returns decimal integer value given a String with a octal representation
`static String`	**`base8`**`(int decimal)` Returns a String with octal representation given a decimal integer value
`static int`	**`base16`**`(String hexadecimal)` Returns decimal integer value given a String with a hexadecimal representation
`static String`	**`base16`**`(int decimal)` Returns a String with hexadecimal representation given a decimal integer value

Facts

- Java **`Integer`** class contains ***`toString`*** and ***`parseInt`*** methods that uses: ***`number`*** and ***`radix`***
  - https://docs.oracle.com/javase/10/docs/api/java/lang/Integer.html
- Your **`RadixUtil`** class implementation should ***not*** have a **`main`** method.
- **NO `Scanner`** for input & **NO `System.out`** for output!

Input

The **`RadixUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`RadixUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`base2("111");`
`base2(7);`
`base8("10");`
`base8(8);`
`base16("f");`
`base16(15);`
	`7`
	`"111"`
	`8`
	`"10"`
	`15`
	`"f"`

Problem 3: Logical Util (10 points) Make API

(API design) Java provides the basic logical operations: and (&&), or (||), exclusive-or (^), not (!). From these basic logical operators, there are often more complex forms of logical expressions that are commonly needed. You're tasked to implement a Logical Utility class for Java that includes the following API *(Application Programming Interface)*. Utility classes are typically helper classes that contain a collection of related static methods. For example, Math is a utility class.

Logical Util Method API:

Modifier and Type	Method and Description
`static boolean`	**`thereExists`**`(boolean p, boolean q, boolean r)` Returns true if at least one condition is true
`static boolean`	**`forAll`**`(boolean p, boolean q, boolean r)` Returns true only if all conditions are true
`static boolean`	**`majority`**`(boolean p, boolean q, boolean r)` Returns true only if a majority of the conditions are true
`static boolean`	**`minority`**`(boolean p, boolean q, boolean r)` Returns true only if a majority of conditions are false
`static boolean`	**`implies`**`(boolean p, boolean q)` Returns true unless p is true and q is false. This is: p implies q
`static boolean`	**`implies`**`(boolean p, boolean q, boolean r)` Returns true unless both p,q are true and r is false. This is: p implies q implies r

Facts

- Your **`LogicalUtil`** class implementation should ***not*** have a **`main`** method.
- **NO `Scanner`** for input & **NO `System.out`** for output!

Input

The **`LogicalUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`LogicalUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`thereExists(false, false, true);`
`forAll(true, true, true)`
`majority(true, true, false);`
`minority(false,false,false);`
`implies(true, false);`
`implies(true, true, false);`
	`true`
	`true`
	`true`
	`true`
	`false`
	`false`

Problem 4: Relational Util (10 points) Make API

(API design) Java provides the basic relational operations: < (greater than), <= (greater than or equal), > (less than), >= (less than or equal). From these basic relational operators, there are often more complex forms of relational expressions that are commonly needed. You're tasked to implement a Relational Utility class for Java that includes the following API *(Application Programming Interface)*. Utility classes are typically helper classes that contain a collection of related static methods. For example, Math is a utility class.

Relational Util Method API:

Modifier and Type	Method and Description
`static boolean`	**`isIncreasing`**`(int x, int y, int z)` Returns true if x is smaller than y and y is smaller than z, exclusive
`static boolean`	**`isDecreasing`**`(int x, int y, int z)` Returns true if x is larger than y and y is larger than z, exclusive
`static boolean`	**`isBetween`**`(int x, int y, int z)` Returns true if y is between x and z, inclusive
`static boolean`	**`isPositive`**`(int x)` Returns true if the number is positive
`static boolean`	**`isNegative`**`(int x)` Returns true if the number is negative
`static boolean`	**`overlaps`**`(int min1, int max1, int min2, int max2)` Returns true if two line segments, min to max, overlap with one another

Facts

- Your **`RelationalUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`RelationalUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`RelationalUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***
`isIncreasing(1,2,3);`	`true`
`isDecreasing(3,2,1);`	`true`
`isBetween(-1,0,1);`	`true`
`isPositive(1);`	`true`
`isNegative(-1);`	`true`
`overlaps(0,1,-1,2);`	`true`
`overlaps(0,1,2,3);`	`false`

Problem 5: Date Util (10 points) Make API

(API design) Dates and Times are commonly used in software to log the activity of data such as when it was created, modified or accessed. However, Dates have no universal formatting scheme. They may be represented using either a set of integers or as words. You're tasked to implement a Date Utility class for Java that includes the following API *(Application Programming Interface)*. Utility classes are typically helper classes that contain a collection of related static methods. For example, Math is a utility class.

Date Util Method API:

Modifier and Type	Method and Description
`static String`	**`format`**`(int month, int day, int year)` Returns String of date, formatted as mm/dd/yyyy
`static String`	**`format`**`(String date, int year)` Returns String of date
`static String`	**`format`**`(String month, int day, int year)` Returns String of date, formatted as month dd yyyy
`static String`	**`format`**`(String month, String day, String year)` Returns String of date, formatted as month day year

Facts

- Java **`String`** class contains ***`format`*** methods that allows formatted Strings similar to printf
  - https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
- Your **`DateUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`DateUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`DateUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`format(1,1,2000);`
`format("Sept 30", 1999);`
`format("October", 31, 2018);`
`format("Oct", "31st", twenty-18);`
	`"01/01/2000"`
	`"Sept 30, 1999"`
	`"October 31, 2018"`
	`"Oct 31st, twenty-18"`

Problem 6: String Util (10 points) Make API

(API design) All software rely on data modeling to represent the things and objects within the algorithm. It's important that developers and end users can inspect the state of these data models to verify the software's results. Humans read data as text, so it is important that developers can translate data into text to evaluate the state. This is commonly done using a method to stringify the data.You're tasked to implement a String Utility class for Java that includes the following API *(Application Programming Interface)*. Utility classes are typically helper classes that contain a collection of related static methods. For example, Math is a utility class.

StringUtil Method API:

Modifier and Type	Method and Description
`static String`	**`toString`**`(double data)` Returns data as a String
`static String`	**`toString`**`(float data)` Returns data as a String
`static String`	**`toString`**`(int data)` Returns data as a String
`static String`	**`toString`**`(long data)` Returns data as a String
`static String`	**`toString`**`(char data)` Returns data as a String
`static String`	**`toString`**`(boolean data)` Returns data as a String

Facts

- Your **`StringUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`StringUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`StringUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`toString(1.0);`
`toString(1.0f);`
`toString(1);`
`toString(1L);`
`toString('1');`
`toString(true);`
	`"1.0"`
	`"1.0"`
	`"1"`
	`"1"`
	`"1"`
	`"true"`

Problem 7: Flip It (10 points)

(Text Processing) Reversing the order of letters in a String is a very common coding challenge that appears in many hiring interviews. Consider how you could achieve this task strictly with a Scanner object with the original text and concatenation onto an empty String.

Facts

- Java **`Scanner`** class has a method **`useDelimiter`** that can change how much text to read at a time, consider changing to empty string **`""`** as your delimiter.
- **`Scanner`** has a method that returns a boolean indicating whether a next value exists in its inputstream (**`hasNext()`**)

- `Scanner` objects can be initialized to to scan String data as input.
- Prepend the new letter in front of the existing text with concatenation.

Input
First line is the number of test cases. Each line thereafter is a line of text.

Output
For each test case, display the text in reverse.

Sample Input	Sample Output
3 Hello World 12345 I can read this	dlroW olleH 54321 siht daer nac I

Problem 8: ASM Emulator (10 points)

(Assembly Programming) High-level programming languages such as Java must be translated into a low-level assembly language (ASM) in order to be executed by the underlying computational hardware. Unlike Java, ASM is very specific to the hardware. ASM syntax consists of only three types: opcodes, registers, and literal values. The opcode refers to the instruction that the processor must perform. Registers represent the available storage. Literal values are the integer numbers. You must implement a Simple Pseudo ASM (SPASM) emulator based on the following specifications

ASM Emulator (Fields):

Modifier and Type	Method and Description

`private static int`	**`eax`** general purpose 32-bit storage for int value, (extended accumulator register)
`private static int`	**`ebx`** general purpose 32-bit storage for int value , (extended base register)
`private static int`	**`ecx`** general purpose 32-bit storage for int value, (extended counter register)
`private static int`	**`edx`** general purpose 32-bit storage for int value, (extended data register)

ASM Emulator (Methods):

Modifier and Type	Method and Description
`static void`	**`mov`**`(String reg1, int val)` moves int data into specified register via a String label ("eax", "ebx", "ecx", "edx")
`static void`	**`mov`**`(String reg1, String reg2)` moves data from register to register via String labels ("eax", "ebx", "ecx", "edx")
`static void`	**`add`**`(String reg1, int val)` adds value with specified register, outputs result to accumulator (eax)
`static void`	**`add`**`(String reg1, String reg2)` adds the values from two registers, outputs result to accumulator (eax)
`static void`	**`imul`**`(String reg1, int val)` multiplies a value with specified register, outputs result to accumulator (eax)
`static void`	**`imul`**`(String reg1, String reg2)` multiples the values from two registers, outputs result to accumulator (eax)
`static void`	**`cmp`**`(String reg1, int val)` compares value to register, (0=false, 1=true) , outputs result to accumulator (eax)
`static void`	**`cmp`**`(String reg1, String reg2)` compares two registers, (0=false, 1=true) , outputs result to accumulator (eax)
`static void`	**`call`**`(String proc, String reg1)` call to external procedure, "PRINT" to console value in a specified register

Facts

- Assembly code uses only four fixed storage locations (i.e. variables) to hold all data values which are represented in this emulator with static fields.
    - eax, ebx, ecx, edx
- Assembly instructions are called opcodes which are represented in this emulator with static methods.
- The call opcode in assembly invokes procedures (i.e. methods). This emulator should have a "PRINT" method that uses System.out to display the contents of a register.
- Your **`ASMEmulator`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input

Input

The **ASMEmulator** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **ASMEmulator** class should only display the state of one of its registers when the call method is invoked with the parameters "PRINT", and one of the register names: "eax", "ebx", "ecx", "edx"

Sample Input	Sample Output
MOV ecx 3 MOV ebx 5 CALL PRINT eax ADD ecx ebx CALL PRINT eax IMUL eax 2 CALL PRINT eax CMP eax 16 CALL PRINT eax	[eax]: 0 [eax]: 8 [eax]: 16 [eax]: 1