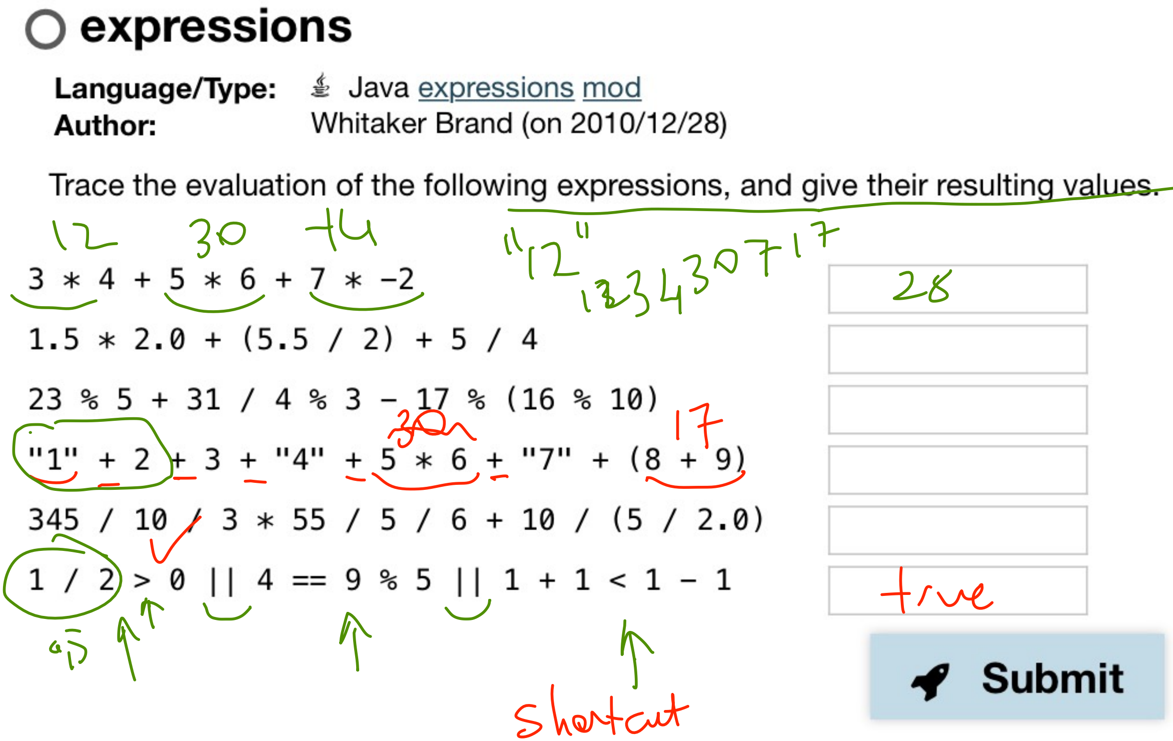
https://practiceit.cs.washington.edu ✓

Since your midterm exam results are "awfull" and your homeworks are not good also I've decided to dedicate last two weeks for practice this is.

login create an occunt



expressions

Language/Type: Java expressions mod
Author: Whitaker Brand (on 2010/12/28)

Trace the evaluation of the following expressions, and give their resulting values.

12 30 -14 "12" 123430717

```
3 * 4 + 5 * 6 + 7 * -2
1.5 * 2.0 + (5.5 / 2) + 5 / 4
23 % 5 + 31 / 4 % 3 - 17 % (16 % 10)
"1" + 2 + 3 + "4" + 5 * 6 + "7" + (8 + 9)
345 / 10 / 3 * 55 / 5 / 6 + 10 / (5 / 2.0)
1 / 2 > 0 || 4 == 9 % 5 || 1 + 1 < 1 - 1
```

30 17

28

true

.5

shortcut

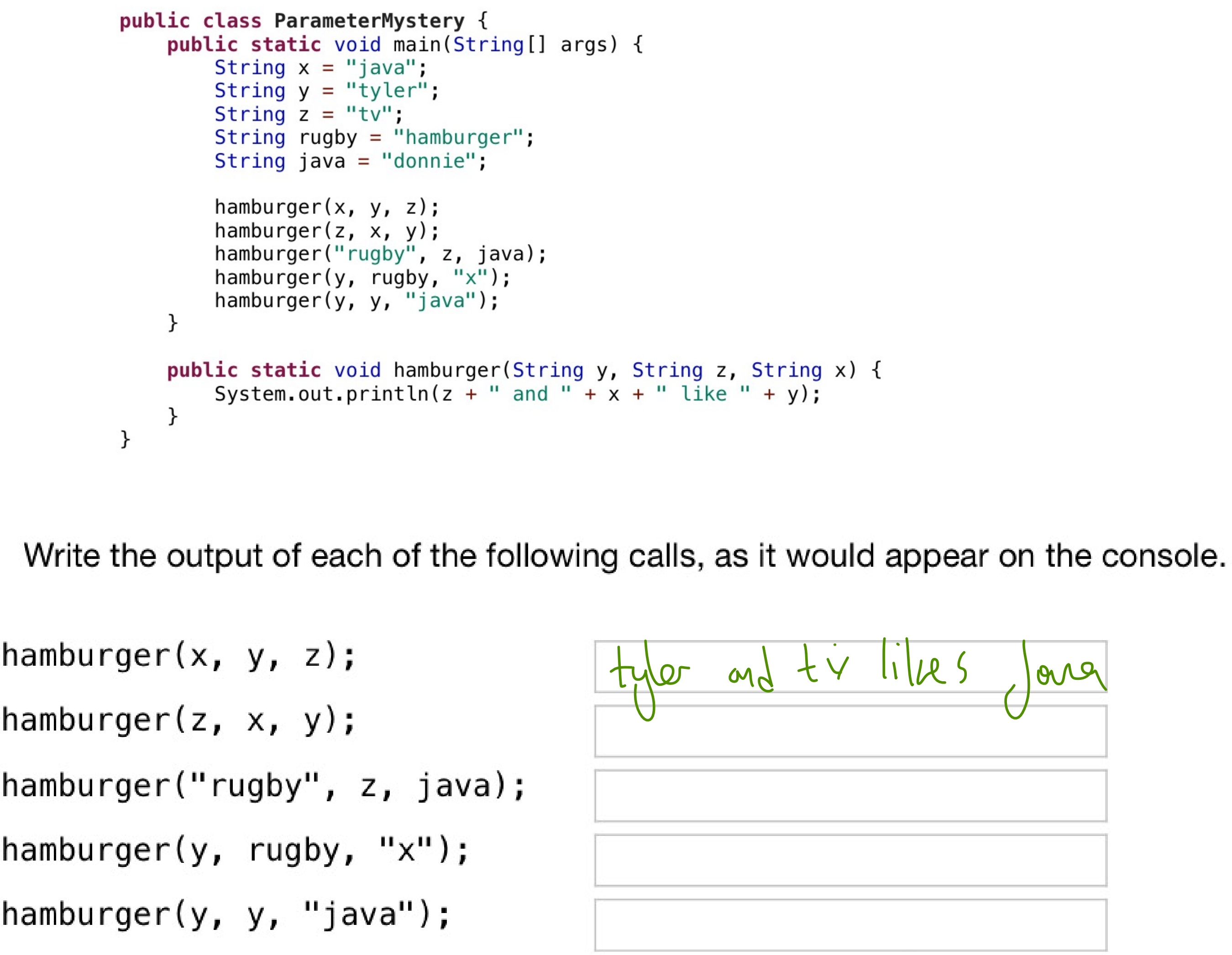
Left Right

+ + + +

Left Right

Table 2-11 D Operator Precedence and Associativity

Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - * & (*type*) sizeof stringof offsetof xlate	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
^^	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= <<= >>=	right to left
,	left to right



```
public class ParameterMystery {
    public static void main(String[] args) {
        String x = "java";
        String y = "tyler";
        String z = "tv";
        String rugby = "hamburger";
        String java = "donnie";

        hamburger(x, y, z);
        hamburger(z, x, y);
        hamburger("rugby", z, java);
        hamburger(y, rugby, "x");
        hamburger(y, y, "java");
    }

    public static void hamburger(String y, String z, String x) {
        System.out.println(z + " and " + x + " like " + y);
    }
}
```

Write the output of each of the following calls, as it would appear on the console.

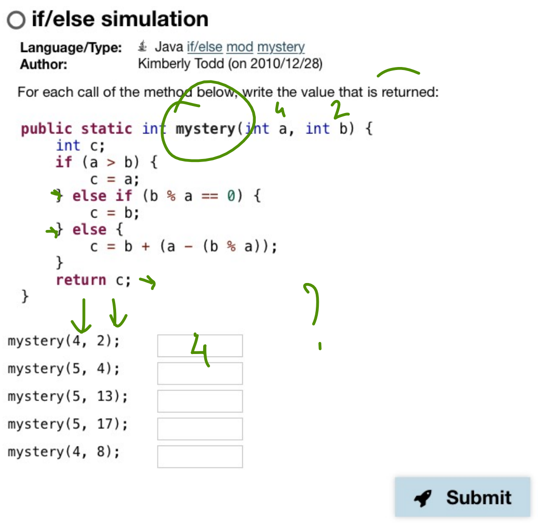
hamburger(x, y, z); tyler and tv likes Java

hamburger(z, x, y);

hamburger("rugby", z, java);

hamburger(y, rugby, "x");

hamburger(y, y, "java");



○ **if/else simulation**

Language/Type: Java if/else mod mystery
Author: Kimberly Todd (on 2010/12/28)

For each call of the method below, write the value that is returned:

4 2

```
public static int mystery(int a, int b) {
    int c;
    if (a > b) {
        c = a;
    } else if (b % a == 0) {
        c = b;
    } else {
        c = b + (a - (b % a));
    }
    return c;
}
```

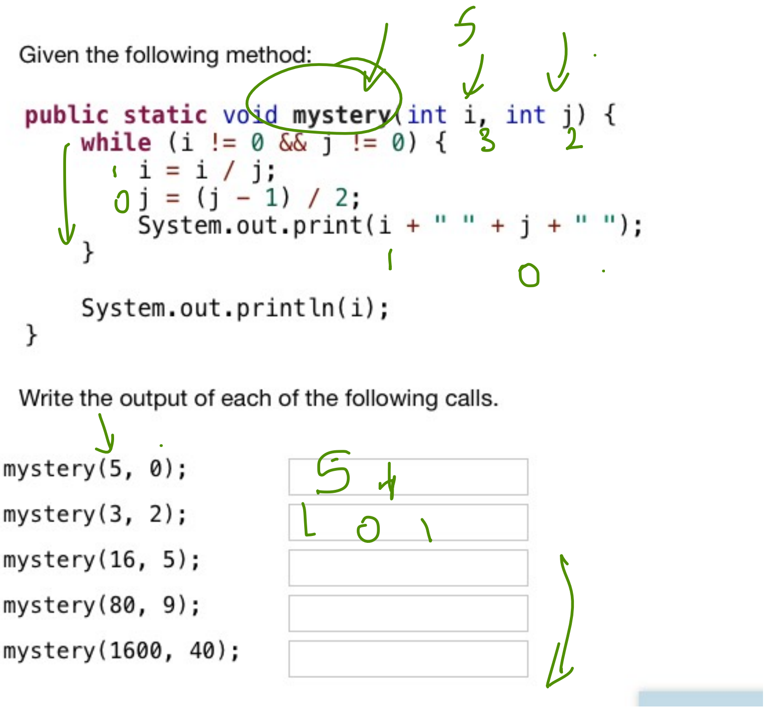
mystery(4, 2); 4

mystery(5, 4);

mystery(5, 13);

mystery(5, 17);

mystery(4, 8);



Given the following method:

5

3 2

```
public static void mystery(int i, int j) {
    while (i != 0 && j != 0) {
        i = i / j;
        j = (j - 1) / 2;
        System.out.print(i + " " + j + " ");
    }

    System.out.println(i);
}
```

1 0

Write the output of each of the following calls.

mystery(5, 0); 5 4

mystery(3, 2); 1 0 1

mystery(16, 5);

mystery(80, 9);

mystery(1600, 40);

For each of the five points labeled by comments, identify each of the assertions in the table below as either being *always* true, *never* true, or *sometimes* true / sometimes false.

```
public static int mystery(int x) {
    int y = 1;
    int z = 0;

    // Point A
    while (y <= x) {
        // Point B
        y = y * 10;
        z++;

        // Point C
    }

    // Point D
    z--;

    // Point E
    return z;
}
```

x = 2

1000

2

100

100

1000

Fill in each box of the the table below with ALWAYS, NEVER or SOMETIMES.

Sometime

Never

never

	y > x	z < 0	z > 0
Point A	(choose)	(choose)	(choose)
Point B	(choose)	(choose)	(choose)
Point C	(choose)	(choose)	(choose)
Point D	(choose)	(choose)	(choose)
Point E	(choose)	(choose)	(choose)

~~Always~~.

☑ Sound F/X

Author: Whitaker Brand (on 2010/12/28)

Write a method named `hasMidpoint` that accepts three integers as parameters and returns `true` if one of the integers is the midpoint between the other two integers; that is, if one integer is exactly halfway between them. Your method should return `false` if no such midpoint relationship exists.

The integers could be passed in any order; the midpoint could be the 1st, 2nd, or 3rd. You must check all cases.

Calls such as the following should return `true`:

```
hasMidpoint(4, 6, 8)
hasMidpoint(2, 10, 6)
hasMidpoint(8, 8, 8)
hasMidpoint(25, 10, -5)
```

true

true

true

2 → 6 ← 10

Calls such as the following should return `false`:

```
hasMidpoint(3, 1, 3)
hasMidpoint(1, 3, 1)
hasMidpoint(21, 9, 58)
hasMidpoint(2, 8, 16)
```

Type your solution here:

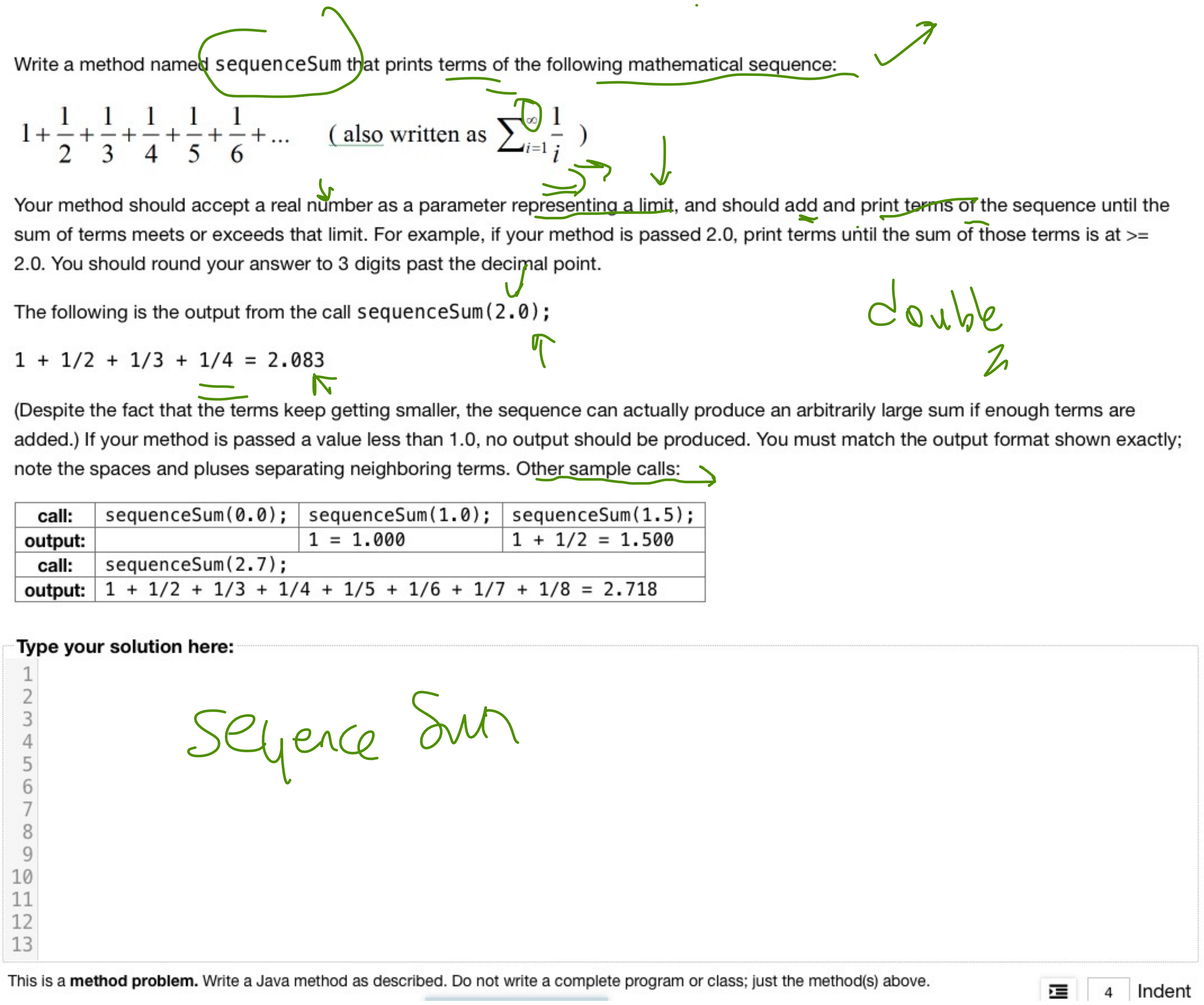
This is a **method problem.** Write a Java method as described. Do not write a complete program or class; just the method(s) above.

Submit

4 Indent

☑ Sound F/X

☑ Highlighting



Write a method named `sequenceSum` that prints terms of the following mathematical sequence:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\ldots \qquad (\text{ also written as } \sum_{i=1}^{\infty}\frac{1}{i}\ )$$

Your method should accept a real number as a parameter representing a limit, and should add and print terms of the sequence until the sum of terms meets or exceeds that limit. For example, if your method is passed 2.0, print terms until the sum of those terms is at >= 2.0. You should round your answer to 3 digits past the decimal point.

double

2

The following is the output from the call `sequenceSum(2.0);`

```
1 + 1/2 + 1/3 + 1/4 = 2.083
```

(Despite the fact that the terms keep getting smaller, the sequence can actually produce an arbitrarily large sum if enough terms are added.) If your method is passed a value less than 1.0, no output should be produced. You must match the output format shown exactly; note the spaces and pluses separating neighboring terms. Other sample calls:

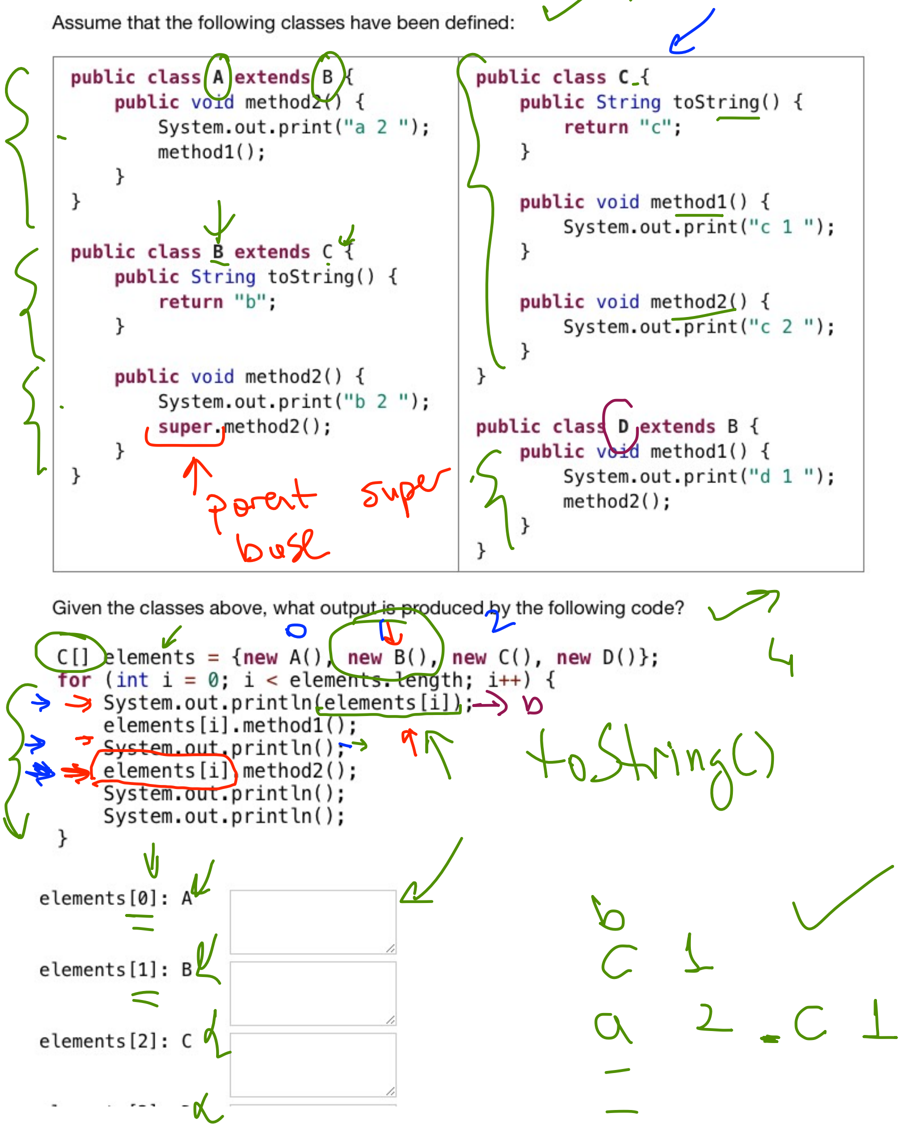
call:	`sequenceSum(0.0);`	`sequenceSum(1.0);`	`sequenceSum(1.5);`
output:		`1 = 1.000`	`1 + 1/2 = 1.500`
call:	`sequenceSum(2.7);`		
output:	`1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 + 1/7 + 1/8 = 2.718`		

Type your solution here:

Seqence Sum

This is a **method problem.** Write a Java method as described. Do not write a complete program or class; just the method(s) above.

4 Indent



Assume that the following classes have been defined:

```
public class A extends B {
    public void method2() {
        System.out.print("a 2 ");
        method1();
    }
}

public class B extends C {
    public String toString() {
        return "b";
    }

    public void method2() {
        System.out.print("b 2 ");
        super.method2();
    }
}
```

parent
base
super

```
public class C {
    public String toString() {
        return "c";
    }

    public void method1() {
        System.out.print("c 1 ");
    }

    public void method2() {
        System.out.print("c 2 ");
    }
}

public class D extends B {
    public void method1() {
        System.out.print("d 1 ");
        method2();
    }
}
```

Given the classes above, what output is produced by the following code?

```
C[] elements = {new A(), new B(), new C(), new D()};
for (int i = 0; i < elements.length; i++) {
    System.out.println(elements[i]);
    elements[i].method1();
    System.out.println();
    elements[i].method2();
    System.out.println();
    System.out.println();
}
```

0 1 2 4

→ b

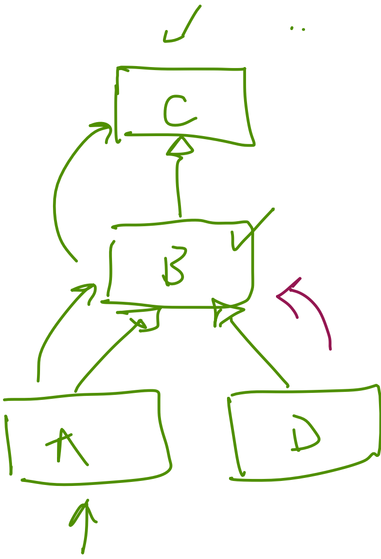
toString()

i=3

i=1

i=2

```
elements[0]: A  □

elements[1]: B  □

elements[2]: C  □

elements[3]: D  □
```

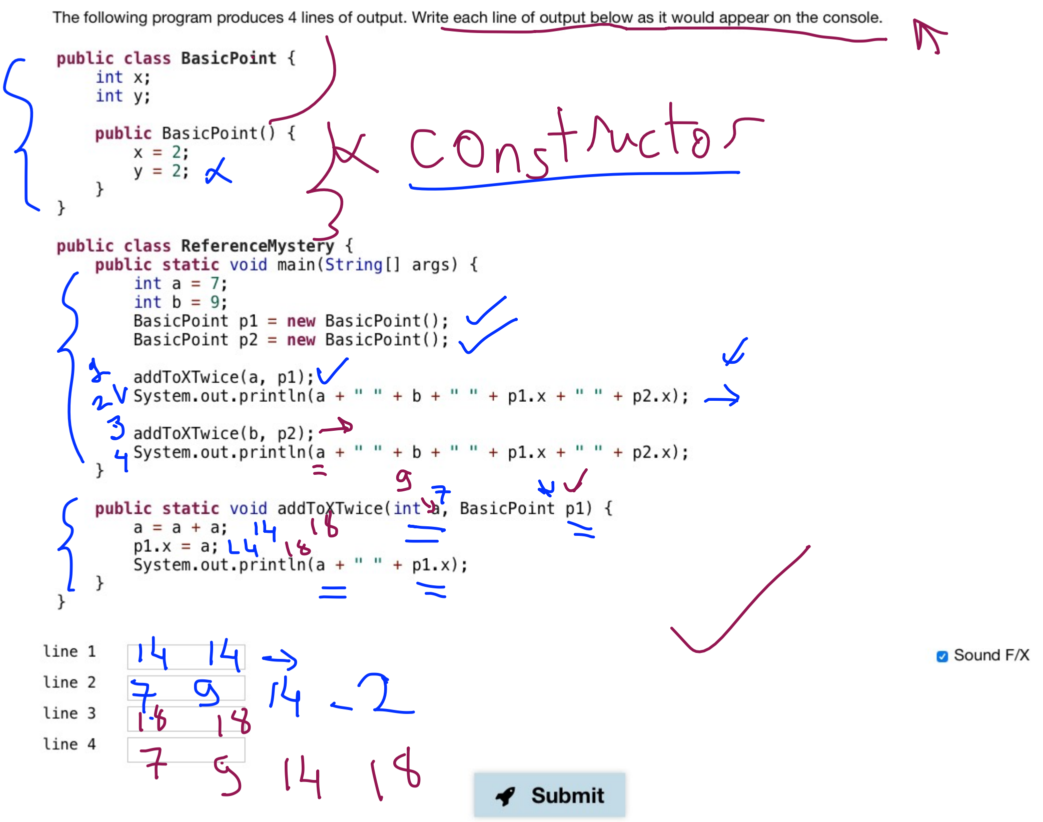


D
C 1
b 2 C 2
–
–

b
C 1
a 2 . C 1
–
–

b
d 1 _ b 2 C 2
b 2 C 2
–
–

C
C 1
C 2
–
–



The following program produces 4 lines of output. Write each line of output below as it would appear on the console.

```
public class BasicPoint {
    int x;
    int y;

    public BasicPoint() {
        x = 2;
        y = 2;
    }
}

public class ReferenceMystery {
    public static void main(String[] args) {
        int a = 7;
        int b = 9;
        BasicPoint p1 = new BasicPoint();
        BasicPoint p2 = new BasicPoint();

        addToXTwice(a, p1);
        System.out.println(a + " " + b + " " + p1.x + " " + p2.x);

        addToXTwice(b, p2);
        System.out.println(a + " " + b + " " + p1.x + " " + p2.x);
    }

    public static void addToXTwice(int a, BasicPoint p1) {
        a = a + a;
        p1.x = a;
        System.out.println(a + " " + p1.x);
    }
}
```

constructor

1 2 3 4

9 7

14 14 18 18

line 1 14 14

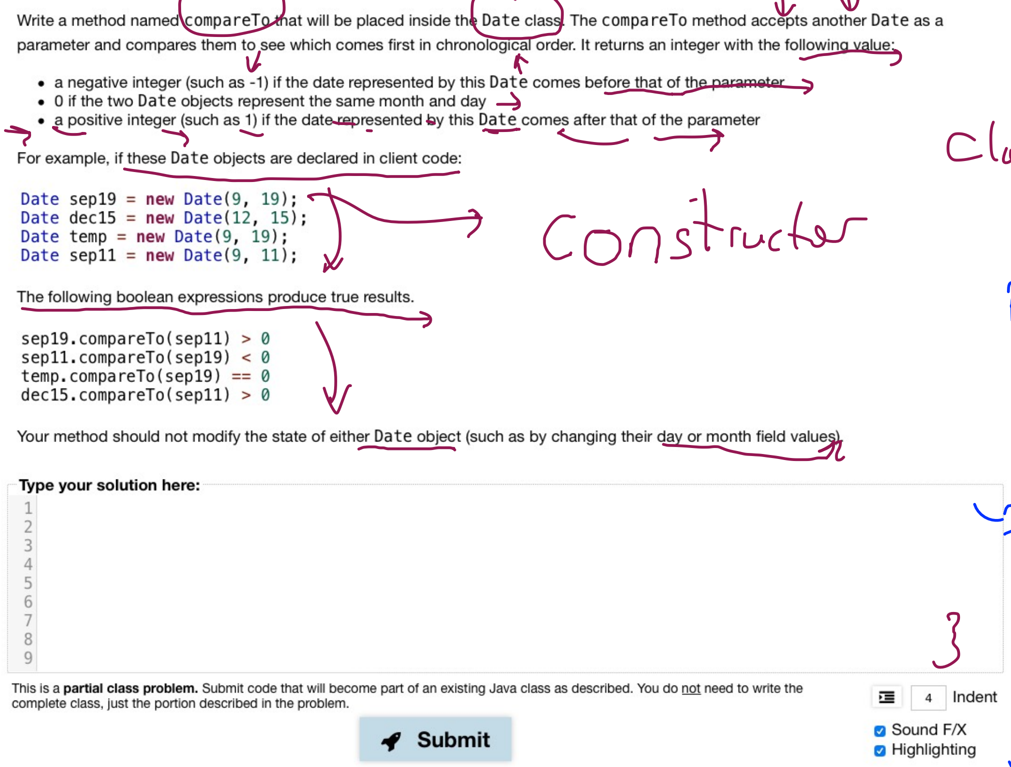
line 2 7 9 14 -2

line 3 18 18

line 4 7 9 14 18

☑ Sound F/X

Submit



Write a method named compareTo that will be placed inside the Date class. The compareTo method accepts another Date as a parameter and compares them to see which comes first in chronological order. It returns an integer with the following value:

- a negative integer (such as -1) if the date represented by this Date comes before that of the parameter
- 0 if the two Date objects represent the same month and day
- a positive integer (such as 1) if the date represented by this Date comes after that of the parameter

For example, if these Date objects are declared in client code:

```
Date sep19 = new Date(9, 19);
Date dec15 = new Date(12, 15);
Date temp = new Date(9, 19);
Date sep11 = new Date(9, 11);
```

constructor

The following boolean expressions produce true results.

```
sep19.compareTo(sep11) > 0
sep11.compareTo(sep19) < 0
temp.compareTo(sep19) == 0
dec15.compareTo(sep11) > 0
```

Your method should not modify the state of either Date object (such as by changing their day or month field values).

Type your solution here:

}

This is a **partial class problem.** Submit code that will become part of an existing Java class as described. You do not need to write the complete class, just the portion described in the problem.

4 Indent

☑ Sound F/X
☑ Highlighting

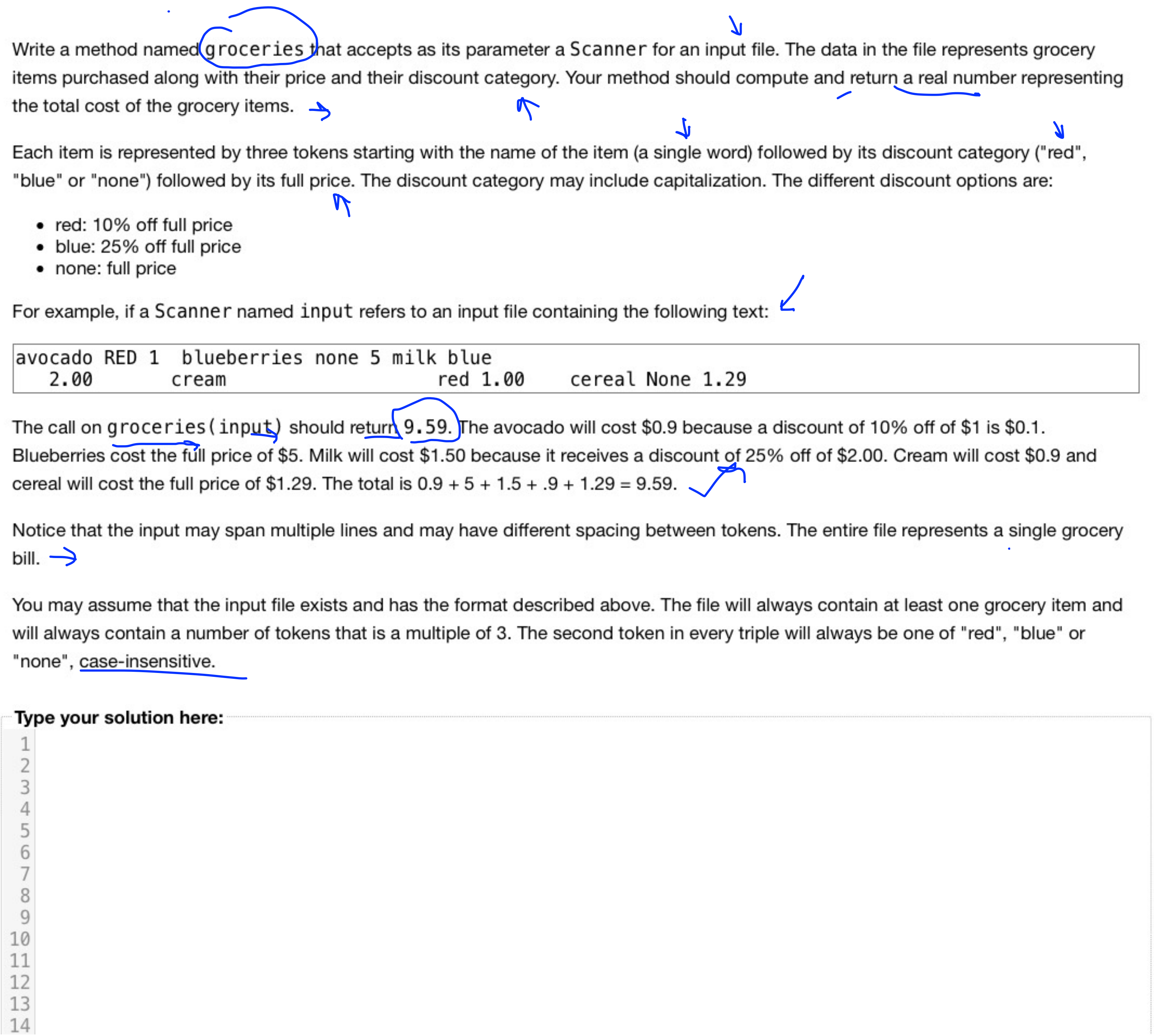
Submit

```
class Date {
  int m;
  int d;
public Date (int m, int d)
{ this.m = m;
  this.d = d }
```

A

A public int compareTo(Date td){
if (this.m > td.m)
return +1;

```
m     d          )

if (this.m < td.m
        return -1;

if (this.m == td.m) {
    if (this.d > td.d)
        return 1
    if (this.d < td.d)
        return -1

    return 0;

}
```



Write a method named `groceries` that accepts as its parameter a `Scanner` for an input file. The data in the file represents grocery items purchased along with their price and their discount category. Your method should compute and return a real number representing the total cost of the grocery items.

Each item is represented by three tokens starting with the name of the item (a single word) followed by its discount category ("red", "blue" or "none") followed by its full price. The discount category may include capitalization. The different discount options are:

- red: 10% off full price
- blue: 25% off full price
- none: full price

For example, if a `Scanner` named `input` refers to an input file containing the following text:

```
avocado RED 1  blueberries none 5 milk blue
   2.00       cream                    red 1.00    cereal None 1.29
```

The call on `groceries(input)` should return `9.59`. The avocado will cost $0.9 because a discount of 10% off of $1 is $0.1. Blueberries cost the full price of $5. Milk will cost $1.50 because it receives a discount of 25% off of $2.00. Cream will cost $0.9 and cereal will cost the full price of $1.29. The total is 0.9 + 5 + 1.5 + .9 + 1.29 = 9.59.

Notice that the input may span multiple lines and may have different spacing between tokens. The entire file represents a single grocery bill.

You may assume that the input file exists and has the format described above. The file will always contain at least one grocery item and will always contain a number of tokens that is a multiple of 3. The second token in every triple will always be one of "red", "blue" or "none", case-insensitive.

Type your solution here:

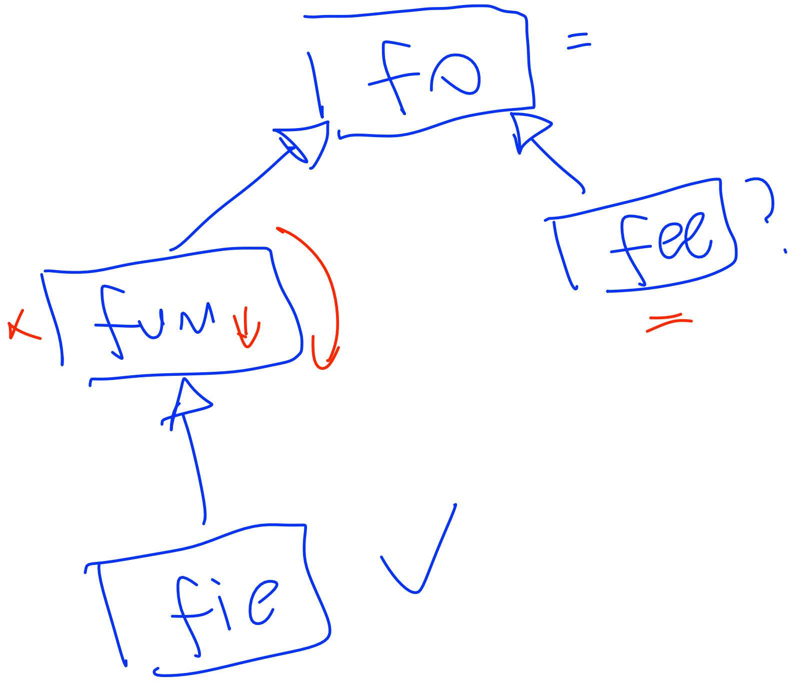
```
public class Fee extends Fo {
    public void method1() {
        System.out.println("Fee 1");
        super.method3();
    }

    public void method3() {
        System.out.println("Fee 3");
    }
}

public class Fie extends Fum {
    public void method1() {
        System.out.println("Fie 1");
    }

    public void method3() {
        System.out.println("Fie 3");
        super.method3();
    }
}

public class Fo {
    public void method2() {
        System.out.println("Fo 2");
        method3();
    }

    public void method3() {
        System.out.println("Fo 3");
    }
}

public class Fum extends Fo {
    public void method3() {
        System.out.println("Fum 3");
    }
}
```

Suppose the following variables are defined:

```
Fum var1 = new Fie();
Object var2 = new Fum();
Fo var3 = new Fee();
Object var4 = new Fie();
Object var5 = new Fo();
Fum var6 = new Fum();
```

Indicate on each line below the output produced by each statement shown. If the statement produces more than one line of output indicate the line breaks with slashes as in a/b/c to indicate three lines of output with a followed by b followed by c. If the statement causes an error, write the word error to indicate this.

`var1.method2();`	Fo 2/ Fie 3/ fum3
`var2.method2();`	error
`var3.method2();`	Fo 2/ Fee 3
`var4.method2();`	error
`var5.method2();`	error
`var6.method2();`	Fo 2/ Fum 3
`var1.method3();`	~~Fee~~ Fie3 / Fun3
`var2.method3();`	error
`var3.method3();`	Fee 3
`var4.method3();`	error
`var5.method3();`	error
`var6.method3();`	~~Fee 3~~ Fum3
`((Fee) var3).method1();`	Fee 1 / Fo 3
`((Fee) var4).method1();`	
`((Fie) var1).method2();`	
`((Fo) var3).method3();`	
`((Fie) var6).method2();`	
`((Fo) var2).method3();`	
`((Fie) var3).method1();`	
`((Fo) var5).method2();`	



Fo
fee
fum
fie