Review of String Methods

Identify the output of the following code segments:

Output:

1.
```
String message = "AP Computer Science";
System.out.println(message.substring(4));
```
"omputer Science"

2.
```
String message = "AP Computer Science";
String topic = message.substring(message.length() - 6);
System.out.println(topic);
```
message.substring(13)

cience

3.
```
String word1 = "Hello";
String word2 = word1;
word1 = word1.substring(1, 2);
System.out.println(word2)
```
word2 → "Hello"

word1 → "e"

"Hello"

4.
```
String name1 = "Ada Lovelace";
String word1 = name1.substring(0, 3) + name1.substring(4);

System.out.println(word1);
```
AdaLovelace

5.
```
int num1 = 2;
double num2 = 3.5;
System.out.println(num1 + num2);
```
5.5

6.
```
int num1 = 2;
double num2 = 3.5;
System.out.println(num1 + " " + num2);
```
2 3.5

Writing Code with the Math Class

For questions 1 - 3, convert the equation into its equivalent in Java statement. Use methods from the Math class and be careful with your data types. Assume that all variables have already been declared.

7. $d = \sqrt{x^2 + y^2}$ Java Statement: d = Math.sqrt(Math.pow(x,2) + Math.pow(y,2));

8. volume $= \frac{4}{3}\pi r^3$ Java Statement: volume = (4.0/3) * Math.PI * Math.pow(r,3);

Introduction to "Casting"

What is Type Casting?

In programming, type casting is the process of "casting" a value of a certain data type, into a storage variable that was designed to store a different data type. Casting here means the conversion of a data value to another version that could fit the variable of the different data type. Type casting in certain cases could lead to loss of information, loss of precision and errors, if not performed carefully.

Type casting an `int` to `double` and a `double` to `int`

To cast a value into a certain variable, we identify the type of data we wish to cast our current data into. Then, we put the data type in parentheses, directly next to the data we wish to cast.

Lets visit the following example:

```
public class TypeCasting1 {
    public static void main(String[]args){

        int x = 13;

        double y = (double)x; // here we type cast the value 13 to a double

        System.out.println("value of x : "+ x); // 13
        System.out.println("value of y : "+ y); // 13.0
    }
}
```

The program above is an example of a type-casting operation. We first initialize an integer with the value of 13. We next type cast the value 13 into a double data type, and assign it to the variable y. The output of running the program is as follows:

value of x : 13
value of y : 13.0

Notice how the value 13 has a floating point (a decimal), after being assigned to y. Please note that Java supports automatic type casting of integers to floating points since there is no loss of precision problems associated with it. On the other hand, Type casting floating points to integers is mandatory when assigning integer variables floating point values. For example, the code below will not compile because you are attempting to assign double data into a variable that stores integer data.

```
int num_people = 13.5;          // Compile error!!
```

You can get around this problem with a "cast" to an integer, which will turn 13.5 into integer data that can then be assigned to num_people. For example:

```
int num_people = (int) 13.5;          // num_people now stores 13
```

Lets examine the another example:

```
public class TypeCasting2 {
      public static void main(String[]args){

            double x = 13.4;

            //int y = x; // if we use this, a "loss of precision" error will
occur
            int y = (int)x; //type casting x to an int , type casting floating
points to integers is necessary

            System.out.println("value of x : "+ x); // 13.4
            System.out.println("value of y : "+ y); // 13.0
      }
   }
```

In the above example, we type casted the floating point value "13.4" to an integer. Doing so will result in the deletion of the ".4", in other words, this will cause the dismissal of the floating point. When assigning floating point values to integer variables, type casting is necessary. Failing to do so will result in the program not compiling with a "possible loss of precision error". A loss of precision error is a warning to the programmer that loss of information could occur due to the programmer's operation. Type casting is a way to enforce the programmer's orders, where the programmer acknowledges the acceptance of the potential loss of information. Running the program above will cause the following output:

value of x : 13.4
value of y : 13

Casting and Java Precedence

In Java, a cast will take precedence over any mathematical operators within a statement. You will still execute any code in parentheses first, then evaluate your 'casts', and finally execute any remaining mathematical operations. Try the following example, then check you work on the next page.

Question 9 refers to the following code segment:

```
int w = 6;
int z = 3;
double x = (int) ((w + z)/2.0);
System.out.println(x);
```

(int)(9/2.0)
↳ (int)(4.5)
↳ 4.0

9. What will be the output of the code above?

(A) 4.5
(B) 3.5
(C) 4.0 ← prints as a decimal because x stores double data
(D) 3.0

ANSWER: Questions 9 results in 4. The code adds w and z to get 9, divides by 2.0 to get 4.5, which is then casted to an integer.

Try the following questions:

Question 10 refers to the following code segment:

```
int w = 6;
int z = 3;
double x = (int) (2.5 + w + z)/2.0;
System.out.println(x);
```

→ (int)(2.5+6+3)/2.0
(int)(11.5)/2.0 ★ cast happens before division
11/2.0
5.5

10. What will be the output of the code above?

(E) 5.75
(F) 5.5
(G) 5.0
(H) 4.0

11. What will the value of the num variable be in the following code segment? 3.0

```
int x = 1;
int y = 5;
int z = 2;
double num = x + y/z;
```

1 + 5/2
1 + 2 → 3.0

12. What will the value of the num variable be in the following code segment? 3.0

```
int x = 1;
double y = 5;
int z = 2;
double num = x + (int)y/z;
```

1 + (int) 5.0/2 ; → 1 + 5/2 → 3.0

13. What will be the output of the following code segment? 3.0

```
int x = 1;
int y = 3;
double z = 3.5;
double balance = (int)(x + y + z)/2;
System.out.println(balance);
```

(int)(1 + 3 + 3.5)/2 → (int)(7.5)/2 → 7/2 → 3.0

```
double x = 2;
int y = 3;
int z = x+y;
```