

**PFP**  
**Quiz 01 - Sample**

**Terminology**

A class is to a blueprint as an object is to \_\_\_\_\_

An object is an \_\_\_\_\_ of a class.

Give 2 reasons for making an instance variable "private":

1.

2.

**Variables, Types, and Values**

Recall that Java has 8 primitive types (int, double, bool, char, and 4 others), as well as reference types. After each code fragment, give the type and value of the variable indicated. Assume that the code fragment exists within code that compiles successfully.

a)

```
int x;  
x = 10;
```

Variable: x. Type: \_\_\_\_\_ Value: \_\_\_\_\_

b)

```
x = 20;  
done = (x > 0);
```

Variable: done Type: \_\_\_\_\_ Value: \_\_\_\_\_

c)

```
House house = new House;
```

Variable: house Type: \_\_\_\_\_ Value: \_\_\_\_\_

d)

```
int num;  
num = 10 / 3;
```

Variable: num Type: \_\_\_\_\_ Value: \_\_\_\_\_

## Expressions and Statements

To the right of each code fragment, if it is a statement write S; if it is an expression write E and its value (or a description of its value).

a) 5

b) `int x;`

c) `x = 5`

d) `x = 5;`

e) `x >= 5`

f) `if (x >= 5)`  
`x = 10;`

g) `return x;`

h) `Math.PI`

i) `house.turnLightsOn();`

j) `house.lightsAreOn()`

## Control Structure (loops and conditional)

For the problem below write down what the code prints.

```
int count = 1;
char prev = '*';
String msg = "";

while(count <= 8){
    msg = msg + prev;
    if(prev == '*'){
        prev = '+';
    }
    else{
        prev = '*';
    }
    count = count + 1;
}
System.out.println(msg);
```

## Code Analysis: LemonadeStand

It's summertime and the kids in the neighborhood want you to code up a LemonadeStand program so they can sell lemonade over the internet. Your friend completed the program and printed out DrJava interactions to demo it, but accidentally deleted some of the code.

DrJava Interactions:

Welcome to DrJava.

```
> LemonadeStand biz = new LemonadeStand();
```

```
> biz.report();
```

Ounces in stock: 400 Cash: 0

```
> biz.buy(1);
```

```
> biz.report();
```

Ounces in stock: 392 Cash: 25

```
> biz.buy(49);
```

```
> biz.report();
```

Ounces in stock: 0 Cash: 1250

```
> biz.restock(320);
```

```
> biz.report();
```

Ounces in stock: 320 Cash: 1250

```
>
```

Fill in Supplied Methods:

Fill in the missing code (marked FILL IN #1/2/3) so that it matches the DrJava interactions. Assume that all input is valid (there is no need to check that method arguments are valid).

I. Write another Constructor (5 pts)

Write another Constructor that takes 4 arguments which initialize the price, ounces per serving, ounces in stock, and cash. Although constructors usually are put at the top of a class, write it at the bottom of page.

II. Write another buy() method (5 pts)

Write another buy() method to replace the existing one which check for one type of invalid input: positive input for a number of glasses in excess of the amount of lemonade available. The method should allocate as many (full) glasses as possible and return the number of glasses sold. Write the buy method at the bottom of the page, or use the reverse side if necessary.

## LemonadeStand.java

```
public class LemonadeStand {
    private int priceInCents;
    private int ouncesPerServing;
    private int ouncesInStock;
    private int cash;

    public LemonadeStand(){
        // FILL IN #1:
        priceInCents =
        ouncesPerServing =
        ouncesInStock =
        cash =
    }

    public void report() {
        System.out.println("Ounces in stock: " + ouncesInStock + "
        Cash: " + cash);
    }

    public void buy(int numberOfGlasses){
        // FILL IN #2:

    }

    public void restock(int ounces){
        // FILL IN #3:

    }
}
```