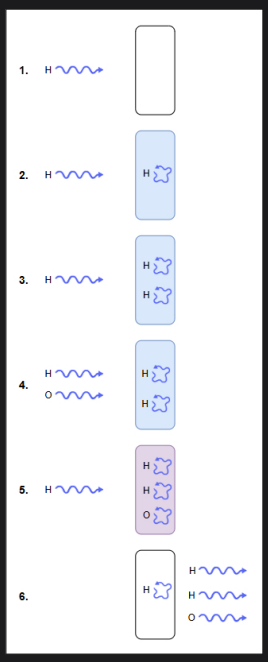
# Build a Molecule

This problem simulated the creation of water molecule by grouping three threads representing Hydrogen and Oxygen atoms.

## Problem Statement

Suppose we have a machine that creates molecules by combining atoms. We are creating water molecules by joining one Oxygen and two Hydrogen atoms. The atoms are represented by threads. The machine will wait for the required atoms (threads), then group one Oxygen and two Hydrogen threads to simulate the creation of a molecule. The molecule then exists the machine. You have to ensure that one molecule is completed before moving onto the next molecule. If more than the required number of threads arrive, they will have to wait. The figure below explains the working of our machine:




Two Hydrogen threads are admitted in the machine as they arrive but when the third thread arrives in step 3, it is made to wait. When an Oxygen thread arrives in step 4, it is allowed to enter the machine. A water molecule is formed in step 5 which exists the machine in step 6. That is when the waiting Hydrogen thread is notified and the process of creating more molecules continues. The threads can arrive in any order which means that HHO, OHH and HOH are all valid outputs.

The code for the class is as follows:

```
class H2OMachine {

    public H2OMachine() {

    }

    public void HydrogenAtom() {

    }

    public void OxygenAtom() {

    }
}
```

The input to the machine can be in any order. Your program should enforce a 2:1 ratio for Hydrogen and Oxygen threads, and stop more than the required number of threads from entering the machine.

## Solution

Our molecule making machine is represented by the class `H2OMachine`, which contains two main methods; `HydrogenAtom()` and `OxygenAtom()`.

The problem is solved by using basic utility functions like `notify()` and `wait()`. The class consits of 3 private members: `sync` for synchronization, `molecule` which is a string array with a capacity of 3 elements (atoms) and `count` to store the current index of the molecule array.

```
class H2OMachine {
```

```
    Object sync;
    String[] molecule;
    int count;

    public H2OMachine() {
    }

    public void HydrogenAtom() {
    }

    public void OxygenAtom() {
    }
}
```

The constructor initializes the `molecule` array with a capacity of 3 atoms and the integer `count` is initialized with 0.

```
public H2OMachine() {
    molecule = new String[3];
    count = 0;
    sync = new Object();
}
```

For synchronization purpose, the entire logic of `HydrogenAtom()` is wrapped in `sync`. First of all, we check the frequency of Hydrogen atom in the `molecule` array by using `frequency()` function found in the **Collections** library of Java. The function deals the array `molecule` as an ArrayList and checks the count of Hydrogen atoms in it. If the array has reached its capacity of 2 Hydrogen atoms, then the thread should wait for space in a new molecule. If the frequency of Hydrogen is less than 2 it means space is available in the current molecule. Hence, H is placed in the array and `count` is incremented. So far, the code of `HydrogenAtom()` is as follows:

```
public void HydrogenAtom() {
   synchronized (sync) {
      
    // if 2 hydrogen atoms already exist
    while (Collections.frequency(Arrays.asList(molecule), "H") == 2) {
        sync.wait();
    }

    molecule[count] = "H";
    count++;
   }
}
```

In case `molecule` is full and `count` is 3, then print the `molecule` and exit the machine. The array `molecule` is reset (initialized with null) and `count` goes back to 0 for a new molecule to be built. At the end of the method, the waiting threads (atoms) are notified using `notifyAll()`. The complete code for `HydrogenAtom()`is given below:

```
public void HydrogenAtom() {
    synchronized (sync) {

    // if 2 hydrogen atoms already exist
    while (Collections.frequency(Arrays.asList(molecule),"H") == 2) {
        sync.wait();
    }
    molecule[count] = "H";
    count++;

    // if molecule is full, then exit.
    if(count == 3) {
        for (String element: molecule) {
            System.out.print(element);
        }
        Arrays.fill(molecule,null);
        count = 0;
    }
    sync.notifyAll();
    }
}
```

The second method `OxygenAtom()` is the same as `HydrogenAtom()` with the only difference of the atom frequency check in the array `molecule`. If it contains one Oxygen atom, then the calling thread goes into `wait()`. If the count of Oxygen atom is not equal to 1 in the `molecule`, then an Oxygen atom “O” is placed in the next available space. The complete code of `OxygenAtom()` is shown below:

```
public void oxygen() {
    synchronized (sync) {

    // if 1 oxygen atom already exists
    while (Collections.frequency(Arrays.asList(molecule),"O") == 1) {
        sync.wait();
    }

    molecule[count] = "O";
    count++;

    // if molecule is full, then exit.
    if(count == 3) {
        for (String element: molecule) {
            System.out.print(element);
        }
        Arrays.fill(molecule,null);
        count = 0;
    }
    sync.notifyAll();
    }
}
```

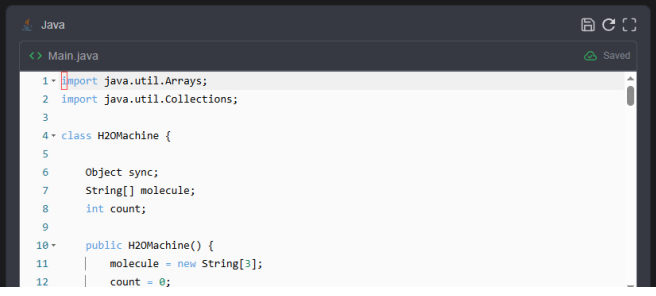
The complete code for the solution is as follows:

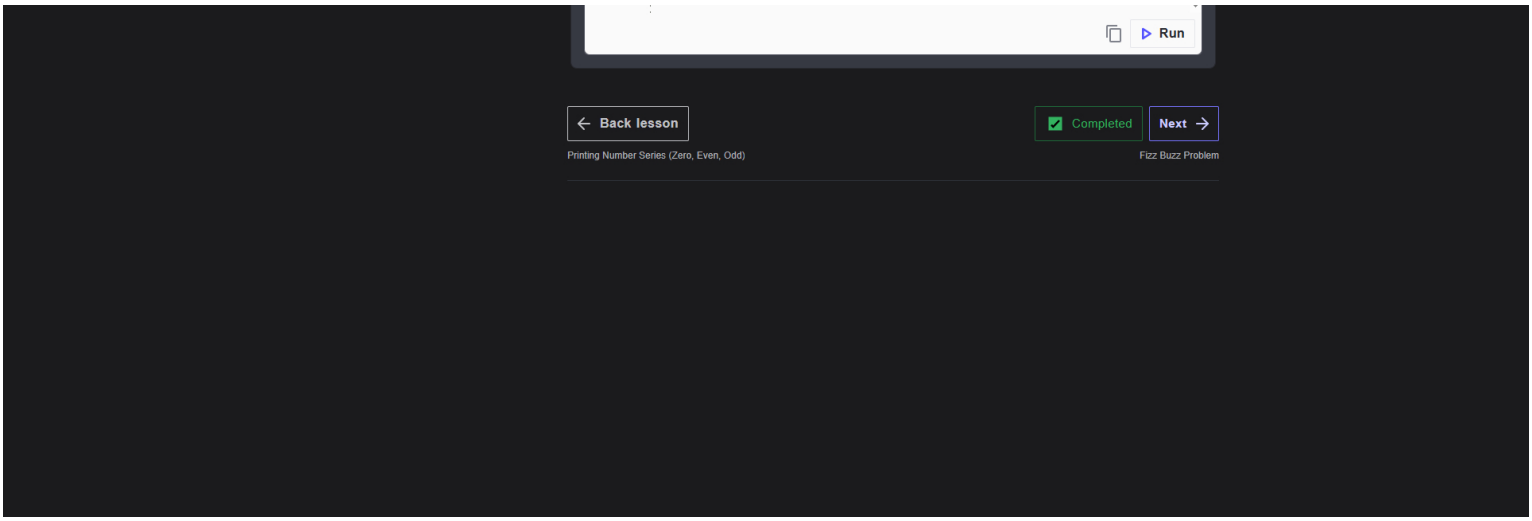
```
class H2OMachine {

    Object sync;
    String[] molecule;
    int count;

    public H2OMachine() {
        molecule = new String[3];
        count = 0;
        sync = new Object();
    }

    public void HydrogenAtom() {
        synchronized (sync) {

            // if 2 hydrogen atoms already exist
            while (Collections.frequency(Arrays.asList(molecule),"H") == 2) {
                sync.wait();
            }

            molecule[count] = "H";
            count++;

            // if molecule is complete, then exit.
            if(count == 3) {
                for (String element: molecule) {
                    System.out.print(element);
                }
                Arrays.fill(molecule,null);
                count = 0;
            }
            sync.notifyAll();
        }
    }

    public void OxygenAtom() throws InterruptedException {
        synchronized (sync) {

            // if 1 oxygen atom already exists
            while (Collections.frequency(Arrays.asList(molecule),"O") == 1) {
                sync.wait();
            }

            molecule[count] = "O";
            count++;

            // if molecule is complete, then exit.
            if(count == 3) {
                for (String element: molecule) {
                    System.out.print(element);
                }
                Arrays.fill(molecule,null);
                count = 0;
            }
            sync.notifyAll();
        }
    }
}
```

We will be creating another class `H2OMachineThread` for multi-threading purpose. It takes an object of `H2OMachine` and calls the relevant method from the string passed to it.

```
class H2OMachineThread extends Thread {

    H2OMachine molecule;
    String atom;

    public H2OMachineThread(H2OMachine molecule, String atom){
        this.molecule = molecule;
        this.atom = atom;
    }

    public void run() {
        if ("H".equals(atom)) {
            try {
                molecule.HydrogenAtom();
            }
            catch (Exception e) {
            }
        }
        else if ("O".equals(atom)) {
            try {
                molecule.OxygenAtom();
            }
            catch (Exception e) {
            }
        }
    }
}
```

We will now be creating 4 threads in order to test our proposed solution. Same object of `H2OMachine` is passed to the 4 threads: **t1**, **t2**, **t3** and **t4**. **t1** and **t3** act as Hydrogen atoms trying to enter the machine where as **t2** and **t4** act as Oxygen atoms. It can be seen from the output that only 1 molecule of H2O exits the machine while the extra Oxygen atom is not utilized.




Run
Back lesson
Printing Number Series (Zero, Even, Odd)
Completed
Next
Fizz Buzz Problem