Java Memory Model

- assume each thread is implemented correctly

```
o.x++;
o.y++;
o.z = o.x = o.y;
```

- this goes from code -> byte codes -> machine code -> micro operations
- if the code is single threaded then instructions can be reordered\

exceptions:

1. volatile - every access has to be in the same order on the compiler level

```
int i = o.x;
int j = o.x;
```

i $\neq$ j is possible if x is volatile/changes

```
while (o.x == 0)
    continue;

=>

int j = o.x;
while (j ==0)
    continue;
```

this is a valid optimization if x is not volatile

```
class T {
    bool locked;
    void lock() {
        while (!locked) {
            continue;
            (*)
            locked = true;
```

```
        }
    }
}
```

- between the operations at $(*)$ the CPU can timeout thread and give time to another
- this issue is fixed with `synchronized`

2. Synchronized

	A	**B**	**C**
A	T	T	
B			
C	T		

- A - normal load + store (most common)
- B - volatile load + enter monitor
- C - volatile store + exit monitor
- left - 1st op
- up - 2nd op

Logic Programming

basic entity	**glue**	**what you give up**
imperative commands	`a; b`	
functional functions	`f(g(x))`	`i = i + 2;` (side effects)
logic predicates	& | !	functions & side effects

logic - think declaratively

- you specify what answers you want

Algorithm = Logic (what you want, spec, correctness) + control (efficiency)

Prolog Syntax (core)

term is one of:

- number

- atom `[a-z][a-zA-Z0-9]*` 'abc def'
  - individual values, not equal to anything else, only have the properties you assign them
  - equal iff their values are the same
  - never equal to a number
- variable `[A-Z\[a-zA-Z0-9]*` X Nl N_o_clock
  - become bound to terms on success
  - unbound on failure
- structure `f(T1,...,Tn)`
  - `f` - atom
  - `T1` - term
  - function symbol functor
  - n>0, n - arity -> f/n
    - pr(3, x, xz(x)) -> pr/3, xz/1
- assignments and everything else ends in a period