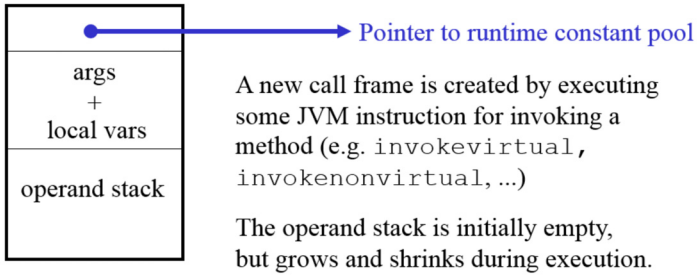
Stack Frames

The Java stack consists of frames. The JVM specification does not say exactly how the stack and frames should be implemented.

The JVM specification specifies that a stack frame has areas for:



A new call frame is created by executing some JVM instruction for invoking a method (e.g. `invokevirtual`, `invokenonvirtual`, ...)

The operand stack is initially empty, but grows and shrinks during execution.

Arithmetic Operations

Unary arithmetic operations

```
ineg        [...:i] -> [...:-i]
i2c         [...:i] -> [...:i%65536]
```

Binary arithmetic operations

```
iadd        [...:i1:i2] -> [...:i1+i2]
isub        [...:i1:i2] -> [...:i1-i2]
imul        [...:i1:i2] -> [...:i1*i2]
idiv        [...:i1:i2] -> [...:i1/i2]
irem        [...:i1:t2] -> [...:i1%i2]
```

Direct operations

```
iinc k a    [...]  -> [...]
            local[k] = local[k] + a
```


Constant Loading Operations

```
iconst_0       [...]  -> [...:0]
iconst_1       [...]  -> [...:1]
iconst_2       [...]  -> [...:2]
iconst_3       [...]  -> [...:3]
iconst_4       [...]  -> [...:4]
iconst_5       [...]  -> [...:5]

aconst_null    [...]  -> [...:null]

ldc_int i      [...]  -> [...:i]
ldc_string s   [...]  -> [...:String(s)]
```


Locals Operations

```
iload k           [...]  -> [...:local[k]]
istore k          [...:i] -> [...]
                  local[k] = i

aload k           [...]  -> [...:local[k]]
astore k          [...:o] -> [...]
                  local[k] = o
```

Field operations

```
getfield f sig    [...:o] -> [...:o.f]
putfield f sig    [...:o:v] -> [...]
                  o.f = v
```

Branch Operations

Nullary branch operations

```
goto L          [...]  -> [...]
                branch always
```

Unary branch operations

```
ifeq L          [...:i] -> [...]
                branch if i == 0
ifne L          [...:i] -> [...]
                branch if i != 0
```

There are also other comparators `ifgt`, `ifge`, `iflt`, `ifle` for unary branching

```
ifnull L        [...:o] -> [...]
                branch if o == null
ifnonnull L     [...:o] -> [...]
                branch if o != null
```


Branch Operations

Binary branch operations

```
if_icmpeq L     [...:i1:i2] -> [...]
                branch if i1 == i2
if_icmpne L     [...:i1:i2] -> [...]
                branch if i1 != i2
```

There are also other comparators `if_icmpgt`, `if_icmpge`, `if_icmplt`, `if_icmple` for binary branching

```
if_acmpeq L     [...:o1:o2] -> [...]
                branch if o1 == o2
if_acmpne L     [...:o1:o2] -> [...]
                branch if o1 != o2
```


Stack Operations

```
dup             [...:v1] -> [...:v1:v1]
pop             [...:v1] -> [...]
swap            [...:v1:v2] -> [...:v2:v1]
nop             [...]  -> [...]
```


Class Operations

```
new C                               [...]  -> [...:o]
invokespecial C/<init>()V           [...:o] -> [...]
```

Class properties of an object

```
instance_of C   [...:o] -> [...:i]
                if (o == null) i = 0
                else i = (C <= type(o))

checkcast C     [...:o] -> [...:o]
                if (o != null && !C <= type(o))
                throw ClassCastException
```

Method operations

```
invokevirtual m sig     [...:o:a1:...:an] -> [...]
invokespecial m sig     [...:o:a1:...:an] -> [...]
```

Return Operations

```
ireturn     [...:<frame>:i] -> [...:i]
            pop stack frame,
            push i onto frame of caller

areturn     [...:<frame>:o] -> [...:o]
            pop stack frame,
            push o onto frame of caller

return      [...:<frame>] -> [...]
            pop stack frame
```


Example Java Method

Consider the following Java method from the `Cons` class

```
public boolean member(Object item) {
   if (first.equals(item))
      return true;
   else if (rest == null)
      return false;
   else
      return rest.member(item);
}
```

Write the corresponding Java bytecode in Jasmin syntax

Example Java Method

Corresponding bytecode (in Jasmin syntax)

```
.method public member(Ljava/lang/Object;)Z
.limit locals 2               // local[0] = o
                              // local[1] = item
.limit stack 2                // [ * * ]
   aload_0                    // [ o * ]
   getfield Cons.first Ljava/lang/Object;
                              // [ o.first *]
   aload_1                    // [ o.first item]
   invokevirtual java/lang/Object/equals(Ljava/lang/Object;)Z
                              // [ b * ] for some boolean b
   ifeq else_1                // [ * * ]
   iconst_1                   // [ 1 * ]
   ireturn                    // [ * * ]
else_1:
   aload_0                    // [ o * ]
   getfield Cons.rest LCons;  // [ o.rest * ]
   aconst_null                // [ o.rest null]
   if_acmpne else_2           // [ * * ]
   iconst_0                   // [ 0 * ]
   ireturn                    // [ * * ]
else_2:
   aload_0                    // [ o * ]
   getfield Cons.rest LCons;  // [ o.rest * ]
   aload_1                    // [ o.rest item ]
   invokevirtual Cons/member(Ljava/lang/Object;)Z
                              // [ b * ] for some boolean b
   ireturn                    // [ * * ]
.end method
```