Moopl compared with Java

Moopl is a simple OO language. Although there are many syntactic differences between Moopl and Java, the semantics of Moopl can be readily understood by comparison with Java. Making that comparison is the purpose of this document. This document does not attempt to provide complete coverage of Moopl's syntax (consult the Moopl grammar document for that).

Compare the Java program on the left with the Moopl program on the right:

```
class Counter {

  int count;

  public Counter() {
    count = 0;
  }

  public void click() {
    ++count;
    System.out.println(getCount());
  }

  public int getCount() {
    return count;
  }

  public static void main(String[] a) {
    int x = Integer.parseInt(a[0]);
    Counter counter = new Counter();
    for (int i = 0; i < x; ++i) {
      counter.click();
    }
  }

}
```

```
proc test(int x) {
  Counter counter;
  counter = new object Counter();
  int i;
  i = 0;
  while (i < x) do {
    counter.click();
    i = i + 1;
  }
}

class Counter {

  int count;

  proc Counter() {
    count = 0;
  }

  proc click() {
    count = count + 1;
    output self.getCount();
  }

  fun int getCount() {
    return count;
  }

}
```

- A Moopl program starts with one or more *top-level procedures*, which play a similar role to `main` methods in Java. In this example the top-level procedure is called test. Note that, in contrast with a `main` method in Java, which must have a single parameter of type `String[]`, a Moopl top-level procedure can have multiple parameters of various types (there is no `String` type in Moopl, however).
- Moopl distinguishes between *procedures* (`proc`) and *functions* (`fun`). A Moopl procedure corresponds to a Java method with `void` return type.
- The Moopl command `output` behaves the same way as Java's `System.out.println` but it can *only* be applied to expressions of type `int`.

- The Moopl keyword `self` corresponds to the Java keyword `this`. In Moopl, all method calls *must* specify an explicit target, even when the target is `self` (compare the method calls in the bodies of the `click` methods above).
- Moopl doesn't have a special syntax for constructors. A constructor declaration is just a `proc` with the same name as its enclosing class. Since Moopl doesn't support method overload, a class can have at most one constructor. (A Moopl class can have no constructor at all, but then it will be impossible to create any instances of the class. Unlike Java, there is no default no-args constructor.)
- All methods in Moopl are implicitly public. There is no syntax for accessing a field outside of the classes to which it belongs (that's "classes", plural, because a field is accessible both in the class where it is declared and in any classes which extend that class).

Additional features of Moopl

- Numbers and operators: **`int`** is the only numeric type in Moopl. **`div`** is integer division (**`/`** in Java). **`and`** is Boolean conjunction (**`&&`** in Java).
- Arrays: Array creation works in essentially the same way as in Java, though the syntax is different. For example, **`new arrayof`**`(boolean)[5]` creates a new array of booleans, with five elements (all initially containing the value false).
- Default field values: as in Java, a field which is not explicitly initialized will start life containing a default value according to its type. This does not apply to local variables.
- Null pointers: **`isnull`** tests if a reference is null; it can only be applied to expressions of class type or array type. The Java equivalent of (**`isnull`** *e*) would be (*e* **== null**) but there is no explicit literal for null in Moopl. (However, as in Java, the default value for fields of class type or array type is null.)