Mapl compared with Java

Mapl is a cut-down version of Moopl, with classes and objects removed. As for Moopl, the semantics of Mapl 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 Mapl's syntax (consult the Mapl grammar document for that).

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

```
class Main {

  public static void main(String[] a) {
    int x = Integer.parseInt(a[0]);
    int[] b = new int[] { x, x };
    inc(b);
    System.out.println(sumTwo(b));
  }

  static void inc(int[] a) {
    for(int i = 0; i < a.length; ++i) {
      a[i] = a[i] + 1;
    }
  }

  static int sumTwo(int[] a) {
    return a[0] + a[1];
  }

}
```

```
proc test(int x) {
  local int[] b;
  b = new int[2];
  b[0] = x; b[1] = x;
  inc(b);
  output sumTwo(b);
  outchar 13; outchar 10;
}

proc inc(int[] a) {
  local int i; i = 0;
  while (i < (a.length)) do {
    a[i] = (a[i]) + 1;
    i = i + 1;
  }
}

fun int sumTwo(int[] a) {
  return (a[0]) + (a[1]);
}
```

- A Mapl program starts with an initial *procedure declaration*, which plays a similar role to a `main` method in Java. In this example the initial procedure is called `test`. In contrast with a `main` method in Java, which must have a single parameter of type `String[]`, the initial procedure can have multiple parameters of various types (there is no `String` type in Mapl, however).
- The rest of a Mapl program is a sequence of zero or more method declarations. Mapl distinguishes between *procedures* (`proc`) and *functions* (`fun`). All methods in Mapl are like static methods in Java. A Mapl procedure corresponds to a Java method with `void` return type.
- The Mapl command `output` behaves the same way as `System.out.print` in Java, but it can *only* be applied to expressions of type `int`.
- Mapl has an `outchar` command for printing single characters, where the character is specified by its Unicode code (which coincides with ASCII for ASCII characters). So `outchar 13; outchar 10;` prints a newline (a CR followed by a LF).
- Because there is no class structure, Mapl doesn't have visibility modifiers.

Additional features of Mapl

- Numbers and operators: **`int`** is the only numeric type in Mapl. **`div`** is integer division (**`/`** in Java). **`and`** is Boolean conjunction (**`&&`** in Java) with short-circuit semantics (like **`&&`** in Java, if its first argument evaluates to false, **`and`** returns false immediately, without evaluating its second argument).
- Arrays: Array creation works in essentially the same way as in Java. Array elements are initialized to zero (note that zero represents `false` in an array of booleans and `null` in an array of arrays). For example, **`new`** `boolean[5]` creates a new array of booleans, with five elements (all initially containing the value `false`).

  Unlike Java, there is no special syntax for initialising multi-dimensional arrays. For example, in Java you could write

  ```
  int[][] a = new int[2][5];
  a[1][4] = 77;
  ```

  but in Mapl you would have to do something like:

  ```
  local int[][] a;
  a = new int[][2];
  a[0] = new int[5]; a[1] = new int[5];
  (a[1])[4] = 77;
  ```

- Null pointers: **`isnull`** tests if a reference is null; it can only be applied to expressions of array type. The Java equivalent of (**`isnull`** *e*) would be (*e* **== null**) but there is no explicit literal for null in Mapl. As in Java, all elements in a newly created array of arrays will contain null (see above); this is the only way to obtain a null reference in Mapl.