Memory Diagrams for Arrays

Franck van Breugel and Hamzeh Roumani

June 12, 2007

Abstract

In his book "Java by abstraction," Roumani introduces *memory diagrams*. These diagrams provide a model of the memory used by a Java application. In this note, we extend the diagrams so that they also reflect *arrays*.

1 Introduction

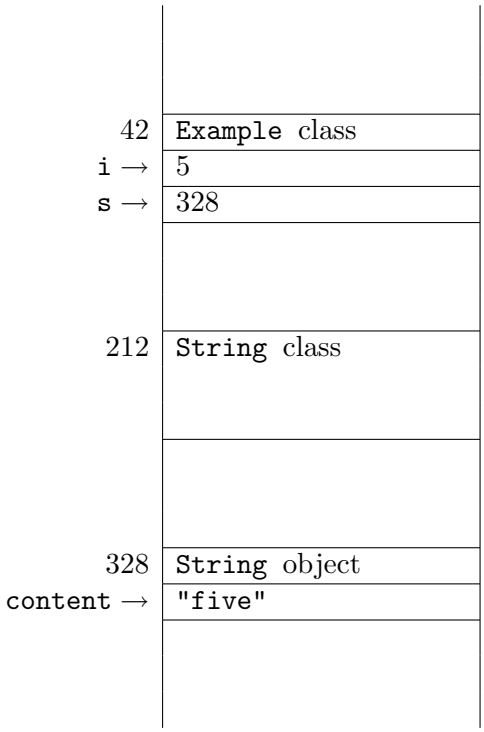
We assume that the reader is already familiar with the memory diagrams as introduced by Roumani in [1]. Such a memory diagram provides a pictorial abstraction of a snapshot of the memory used by a Java application at some point during its execution. In a memory diagram we can depict the following:

- classes loaded in memory;
- objects stored in memory;
- constructors, attributes and methods of a class loaded in memory;
- non-static attributes and methods of an object stored in memory;
- values of static attributes of a class loaded in memory;
- values of non-static attributes of an object stored in memory;
- variables of the `main` method of the application;
- values of variables of the `main` method of the application.

Consider, for example, the following (meaningless) application.

```
1 public class Example
2 {
3    public static void main(String[] args)
4    {
5       int i = 5;
6       String s = "five";
7    }
8 }
```

When the execution of the application has reached the end of line 6, the memory used by the application can be modelled by the following memory diagram.



The above diagram reflects the following.

- The `Example` class has been loaded into memory at the (fictitious) address 42. The `String` class has been loaded into memory at address 212.
- A `String` object, with literal `"five"`, is stored at address 328.
- The variable `i` has the value 5 and the variable `s` refers to the object stored at address 328.

For more details about memory diagrams, we refer the reader to [1]. In the rest of this note, we will discuss how arrays can be reflected in these diagrams.

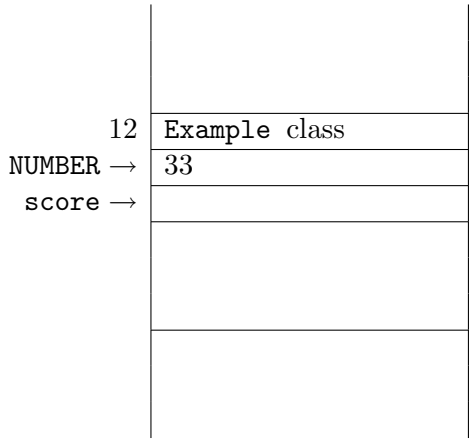
2 Arrays

By means of two simple examples, we will show how arrays can be reflected in memory diagrams. Consider the following application.

```
1 public class Example
2 {
3    public static void main(String[] args)
4    {
5       final int NUMBER = 33;
6       int[] score;
7       score = new int[NUMBER];
```

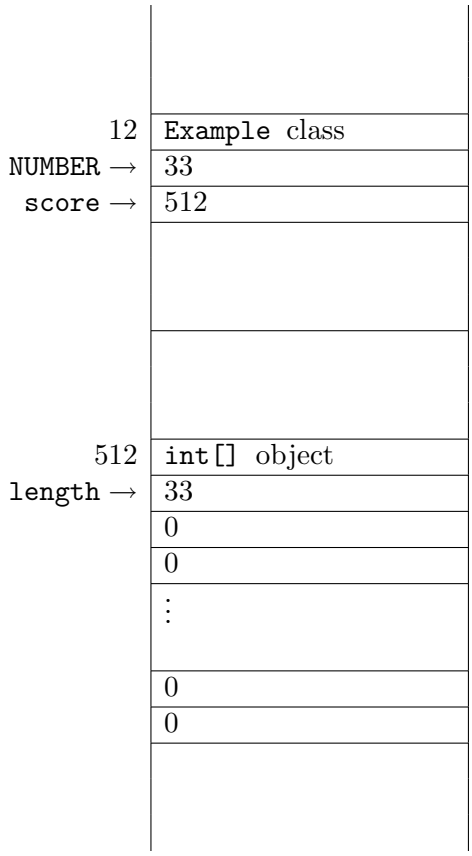
```
8         for (int i = 0; i < score.length; i++)
9         {
10           score[i] = 1;
11        }
12    }
13 }
```

When the execution of the application has reached the end of line 6, the memory used by the application can be modelled by the following memory diagram.



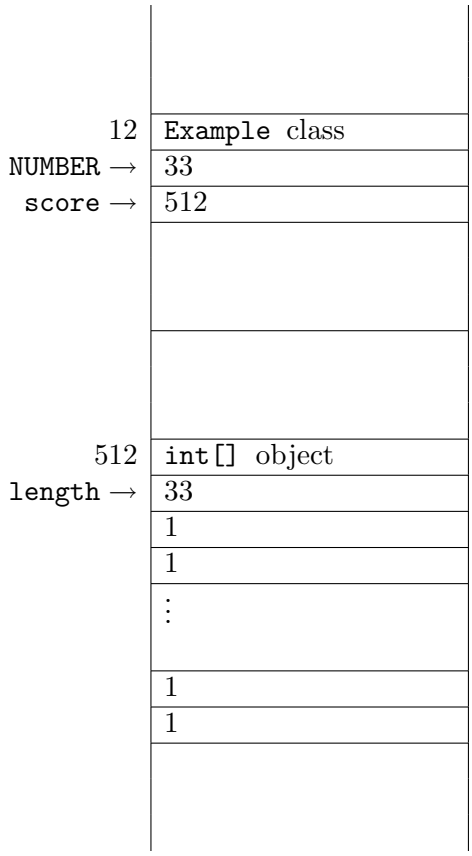
The above memory diagram reflects that the class `Example` has been loaded into memory. Furthermore, the diagram tells us that variable `NUMBER` has value 33 and variable `score` has no value yet.

Once the execution of the application has reached the end of line 7, the above diagram would be extended as follows.



At this point, the variable `score` refers to the `int[]` object stored at address 512. This object contains 33 cells (not all are depicted) and the attribute `length`. At creation, all cells of an array contain default values. Since the base type of the array `score` is `int`, all cells contain 0.

Once the execution of the application has reached the end of line 11, the above diagram would be extended as follows.

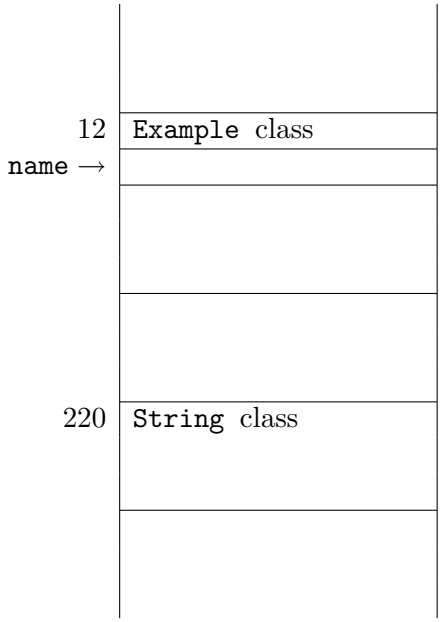


Note that all cells now contain 1.

In the second example, we consider an array the base type of which is a reference type. Consider the following application.

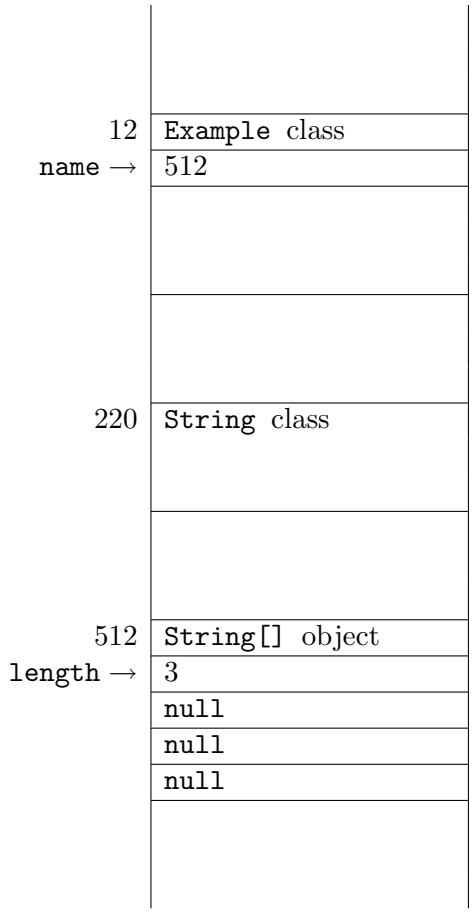
```
1  public class Example
2  {
3     public static void main(String[] args)
4     {
5        String[] name;
6        name = new String[2];
7        name[0] = "John Doe";
8        name[1] = "Jane Doe";
9        name[2] = "Baby Doe";
10    }
11 }
```

When the execution of the application has reached the end of line 5, the memory used by the application can be modelled by the following memory diagram.



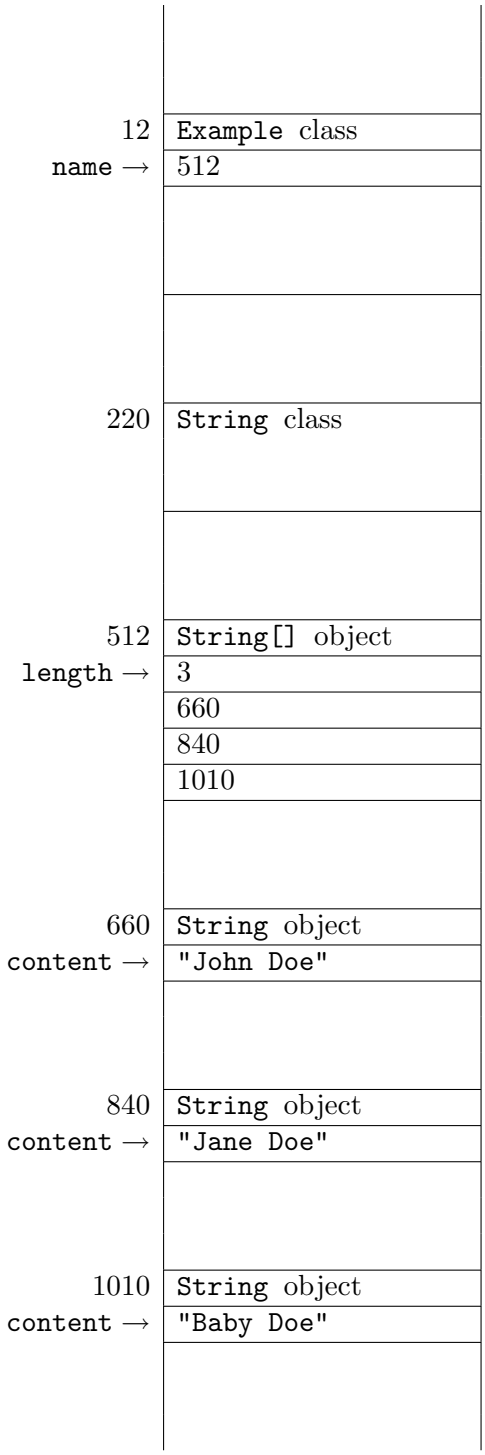
The above memory diagram reflects that the class `Example` has been loaded into memory. Furthermore, the diagram tells us that variable `name` has no value yet.

Once the execution of the application has reached the end of line 6, the above diagram would be extended as follows.



At this point, the variable `name` refers to the `String[]` object stored at address 512. This object contains 3 cells and the attribute `length`. At creation, all cells of an array contain default values. Since the base type of the array `name` is `String`, all cells contain `null`.

Once the execution of the application has reached the end of line 9, the above diagram would be extended as follows.



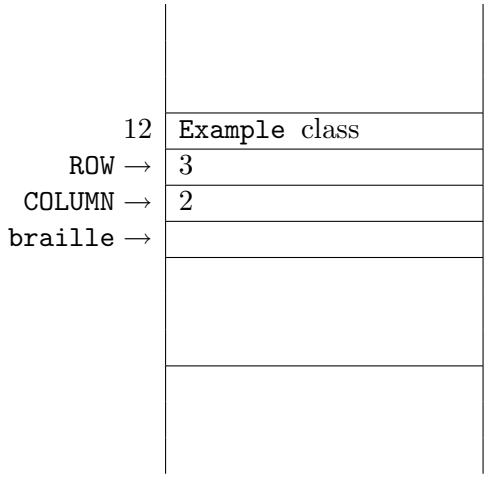
Note that the array cells contain the addresses of the `String` objects.

3 Higher-dimensional arrays

By means of a simple example, we will show how two-dimensional arrays can be reflected in memory diagrams. Consider the following application.

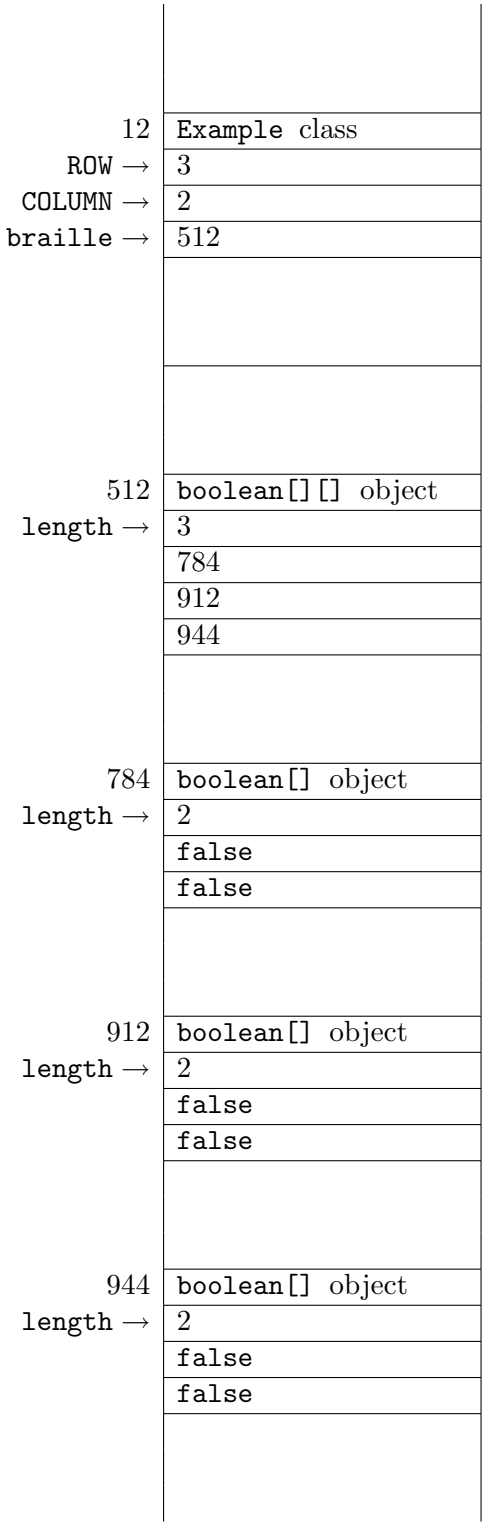
```
1  public class Example
2  {
3     public static void main(String[] args)
4     {
5        final int ROW = 3;
6        final int COLUMN = 2;
7        boolean braille[][];
8        braille = new boolean[ROW][COLUMN];
9        braille[0][0] = true;
10    }
11 }
```

When the execution of the application has reached the end of line 7, the memory used by the application can be modelled by the following memory diagram.



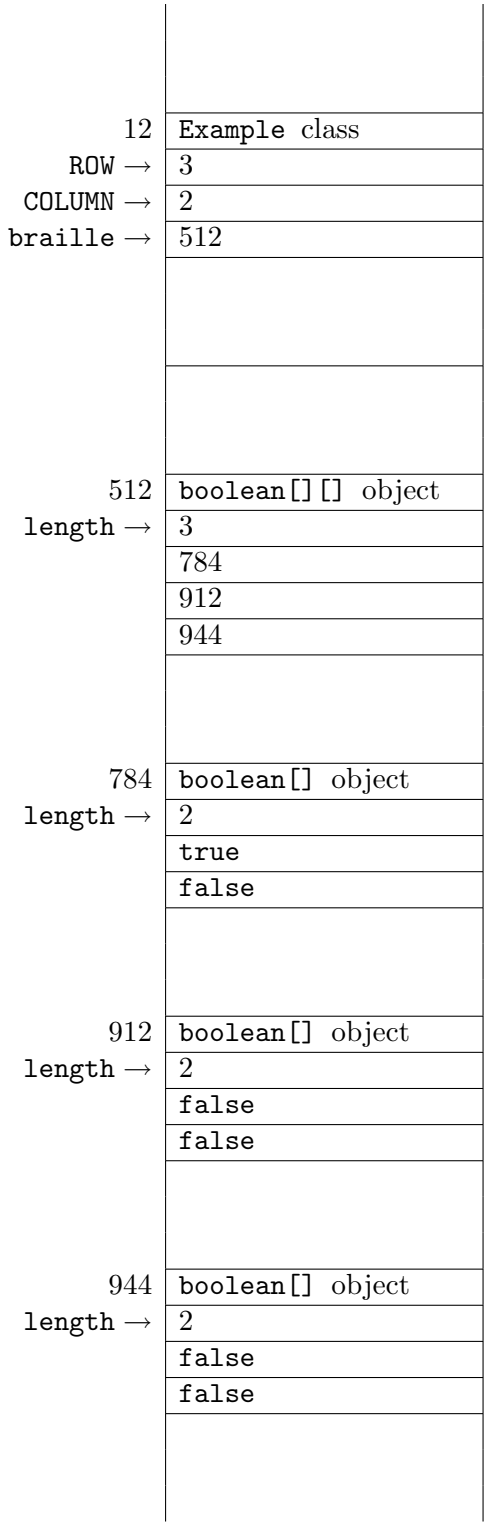
The above memory diagram reflects that the class `Example` has been loaded into memory. Furthermore, the diagram tells us that variables `ROW` and `COLUMN` have values 3 and 2, respectively, and variable `braille` has no value yet.

Once the execution of the application has reached the end of line 8, the above diagram would be extended as follows.



At this point, the variable `braille` refers to the `boolean[][]` object stored at address 512. This object contains 3 cells and the attribute `length`. The cells contain the addresses of `boolean[]` objects. These `boolean[]` objects contain 2 cells and the attribute `length`. These cells contain the default value for `boolean`, that is `false`.

Once the execution of the application has reached the end of line 9, the above diagram would be extended as follows.



The only difference with the previous diagram is the first cell of the `boolean[]` object at address 784 now contains `true`.

References

[1] Hamzeh Roumani. *Java by abstraction: a client-view approach.* Pearson Addison Wesley, Toronto, Canada, first edition, 2006.