The basics of the array data structure

- **Storing information**
  - **Computer programs** (and humans) ***cannot* operate** without **information**.

    Example:

    - **CEO's** of companies need **sales information** to make the correct decisions
    - **Computer programs** (such as the `min` **method**) need **input data** to operate

  - ***Very often***, a **computer program** is used to **process a *large amount* information** of **similar *structure***

    Example:

    - A **banking computer program** processes a **large number** of **bank account information** .... but **each bank account** contains **similar information**
    - A **health care administration program** processes a **large number** of **patient record information** .... but **each patient record** contains **similar information**
    - And so on...

  - The ***array* data structure** is **very suitable** for **storing** a **collection of *similar* data items**

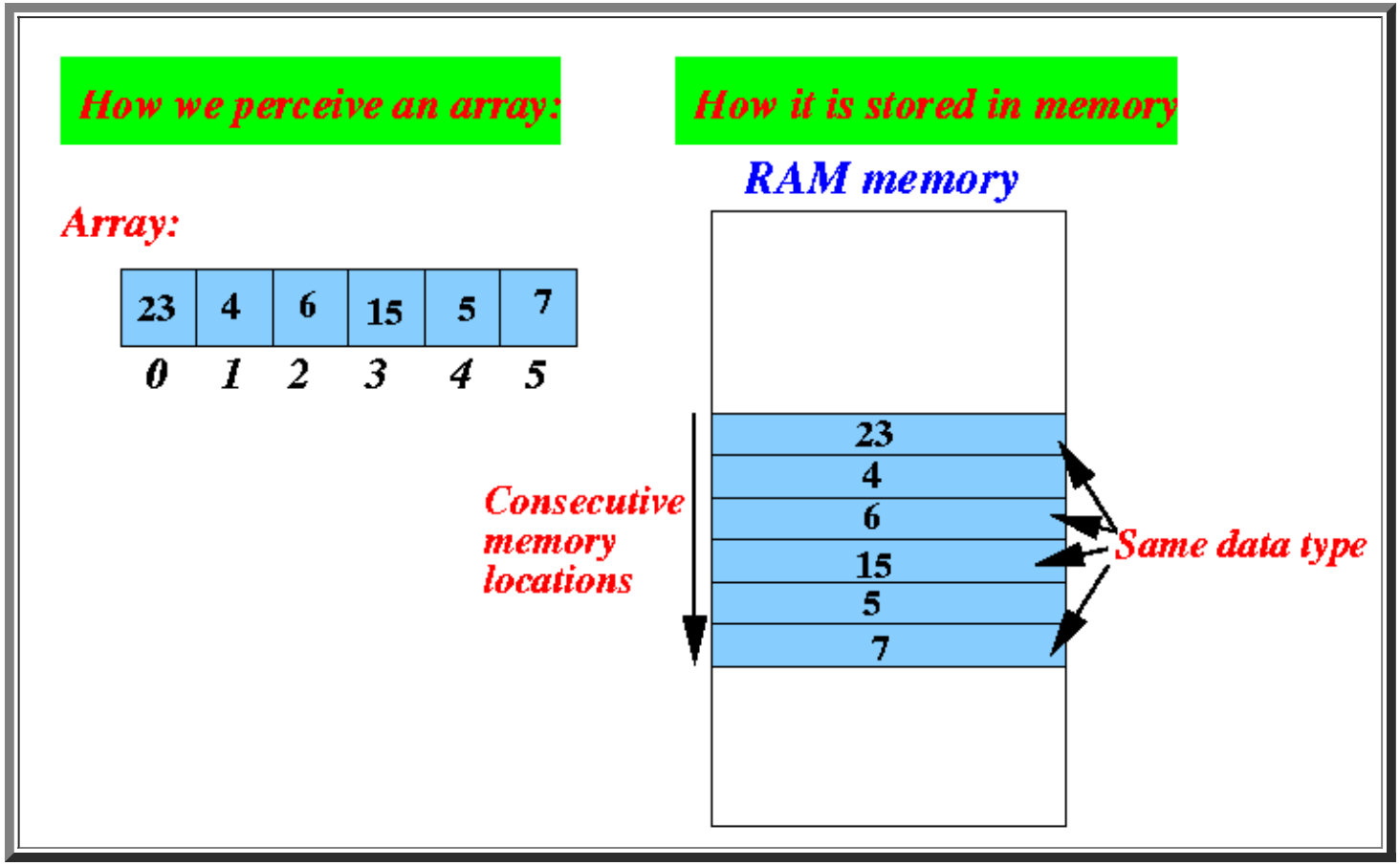
- **The *array* data structure**
  - An **array** is a ***collection* (multiple) of variables** where:

    - **Each variable** in the **collection** is of the ***same* data type**

      Note: that means that the **size (number of bytes)** of each variable is **the same**

- The **variables** are **placed (stored)** ***consecutively*** **in memory**

Example:



Properties of arrays:

- The **array** contains a ***fixed*** **number elements**
  - **length** of the array = ***number*** **of elements** in the array

    You can **specify** the **length** when you **create** the **array**
- Each **array element** is ***identified*** by an **index**

- The ***first*** **element** of an **array** has the **index 0**

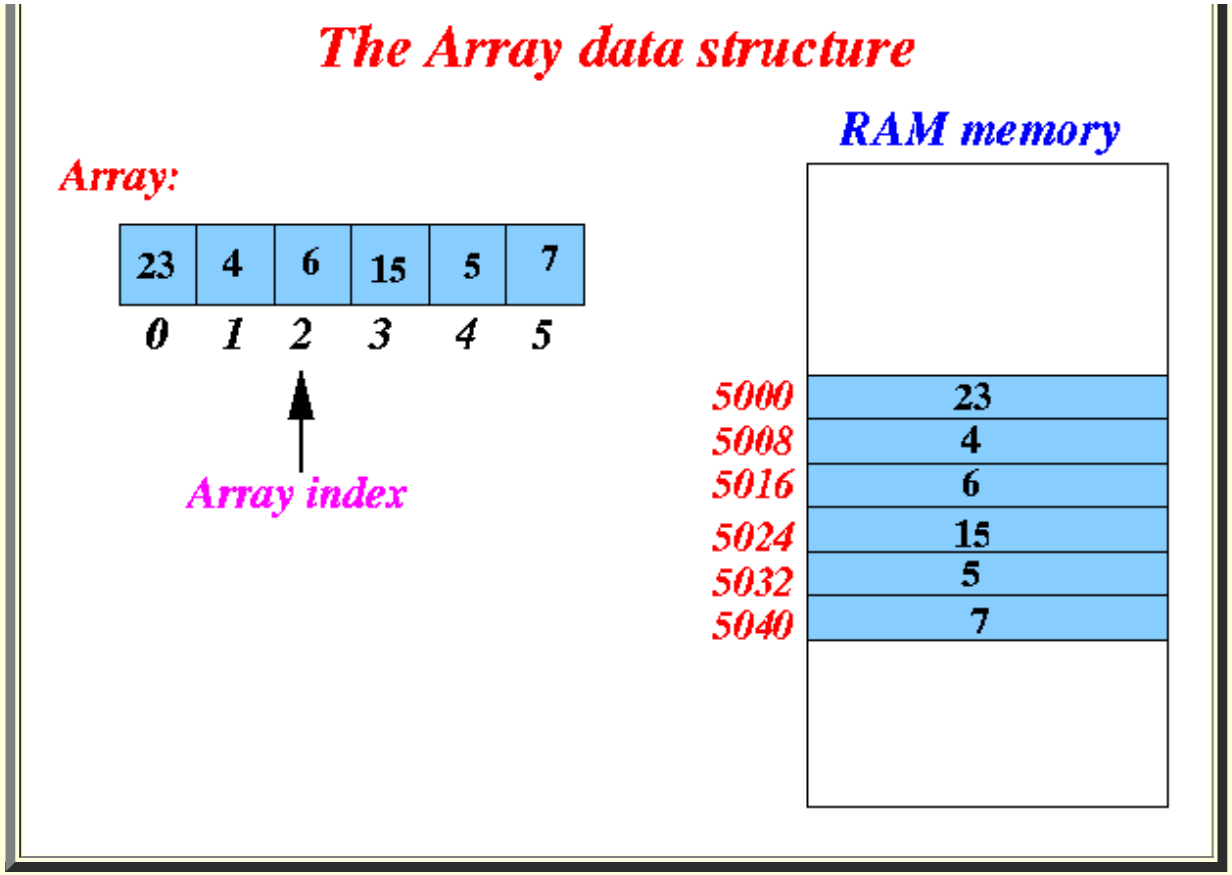
- Because **each variable** is of the **same size**, if we know the **location of the *first* element** of an **array**, we can locate **any other element of the array** through its **index**

  Example:

  - We define an array of `double` **typed** variables.
  - Now, suppose the ***first*** **element** of this array is located at **address 5000**

    Because a **double** types variable uses **8 bytes** of memory to store the data, each **"element" of the array** is **8 bytes** long.

  - Therefore:



- Then:
  - Array element **0** is located at address **5000**
  - Array element **1** is located at address **5008**
  - Array element **2** is located at address **5016**
  - ...
  - Array element **i** is located at address **5000 + 8×i**

- **Terminology: base address of an array**
  - **Base Address:**

> - **Base Address** of an array = the **address** of the **first element** of the array

Because we know the **size** of each array element, if we know the **base address**, we can **compute** the **address** of ***any*** **array element** if we are given its **array index**.

- **Implementation note**
  - What I have shown above is the **most essential part** of an array.
  - Different programming languages can **add additional information** to the representation to provide additional features.

    Example 1:

    - **Java** includes a field that contains the **length (= number of elements)** of the array.
    - With this `length` field, a **Java program** can make sure that the programmer cannot use an **illegal index**

    Example 2:

    - The **Pascal** programming language maintain the **start index** and the **end index** of the array.

      With these 2 pieces of information included in the array, you can define an array of arbitrary index, e.g.:

      ```
      type a = array [4..6] of integer;
      ```

      This results in 3 array elements: `a[4]`, `a[5]` and `a[6]`. (So array indices in **Pascal** do not need to start at **0** !)

- **Defining an *array* in Java**
  - In **most programming languages**, ***defining*** **an array** is a **one-step process**

    In **Java**, however, ***defining*** **an array** is a **two-step process**

  - **Defining an array in Java:**

    - **Step 1:**

- Define a **variable** to store the **location of the *first* element** of the array.

  This **variable** is called an **(array) object *reference* variable**

  (Recall that in **Computer Science**, a ***reference*** is an **address** of a variable or method)

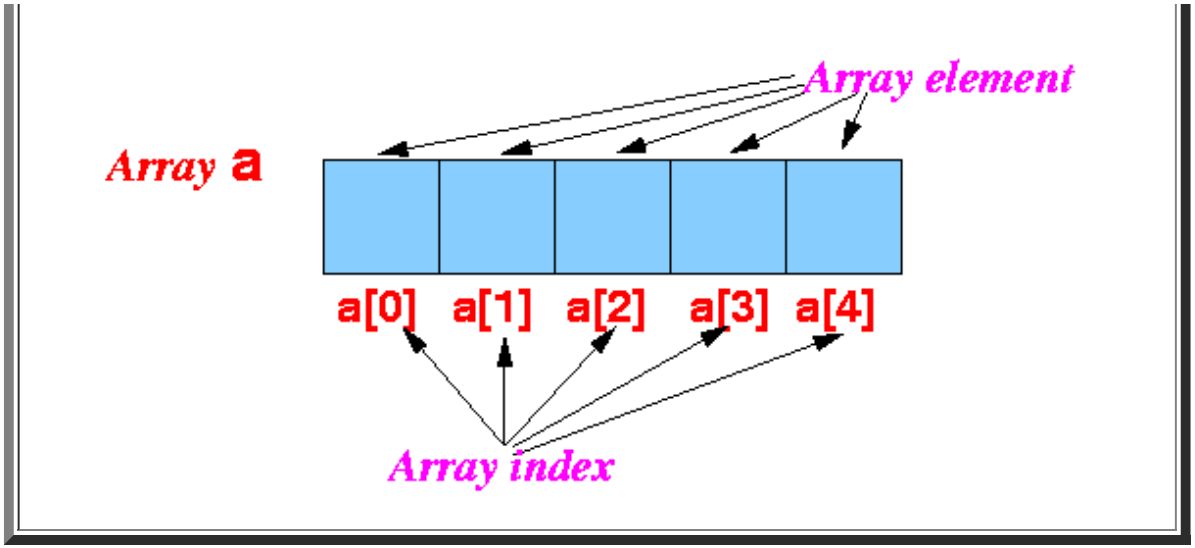
- **Step 2:**

  - Create the **array** and store the **location of the *first* element** of the array in the **(array) object *reference* variable**

Example:

```
Step 1:

  double[] a;            // The type "double[]" is the
                         // "array object reference" type
                         // The variable a contains an address of
                         // an array of doubles

---

Step 2:

  a = new double[5] ;    // new double[5] creates an array
                         // of 5 elements of double type
                         // new double[5] returns the address
                         // of the first element of the array
                         // which is stored in variable a
```

Result of the **array definition**:



Explanation:

- The **array definition**:

```
double[] a;
a = new double[5];
```

creates an **array** of **5 `double` typed variables**

- These **5 `double` typed variables** have the following **name**:

```
a[0]
a[1]
a[2]
a[3]
a[4]
```

These **5 `double` typed variables** are **called array elements**.

Each **array element** is a **(simple) variable** of the **type `double`**.

- **Complete Java Example: defining an array**

- **Program example with array definition:**

  Focus *only* on the definition an **array of 5 *doubles*** in the previous example

```
public class Avg2
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      double[] a;         // Define an (array) object variable named "a"
      a = new double[5];  // new double[5]: create an array of 5 double variables
                          // and returns the location of the first element

      ****************************************
      Ignore the rest of the program
      ****************************************

      double sum, avg;
      int i;                           // index

      for ( i = 0; i <= 4; i++ )
      {
         System.out.print("Enter a number: ");
         a[i] = in.nextDouble();            // Read in number
      }

      /* ------------------------------------------------
         Use the "running sum" algorithm to compute total
         ------------------------------------------------ */
      sum = 0.0;

      for ( i = 0; i <= 4; i++ )
      {
         sum = sum + a[i];
      }
      avg = sum/5;
      System.out.println(avg);        // Print average
   }
}
```

- **The data type `double[]`**
  - The **data type `double[]`**

    - **Note:**

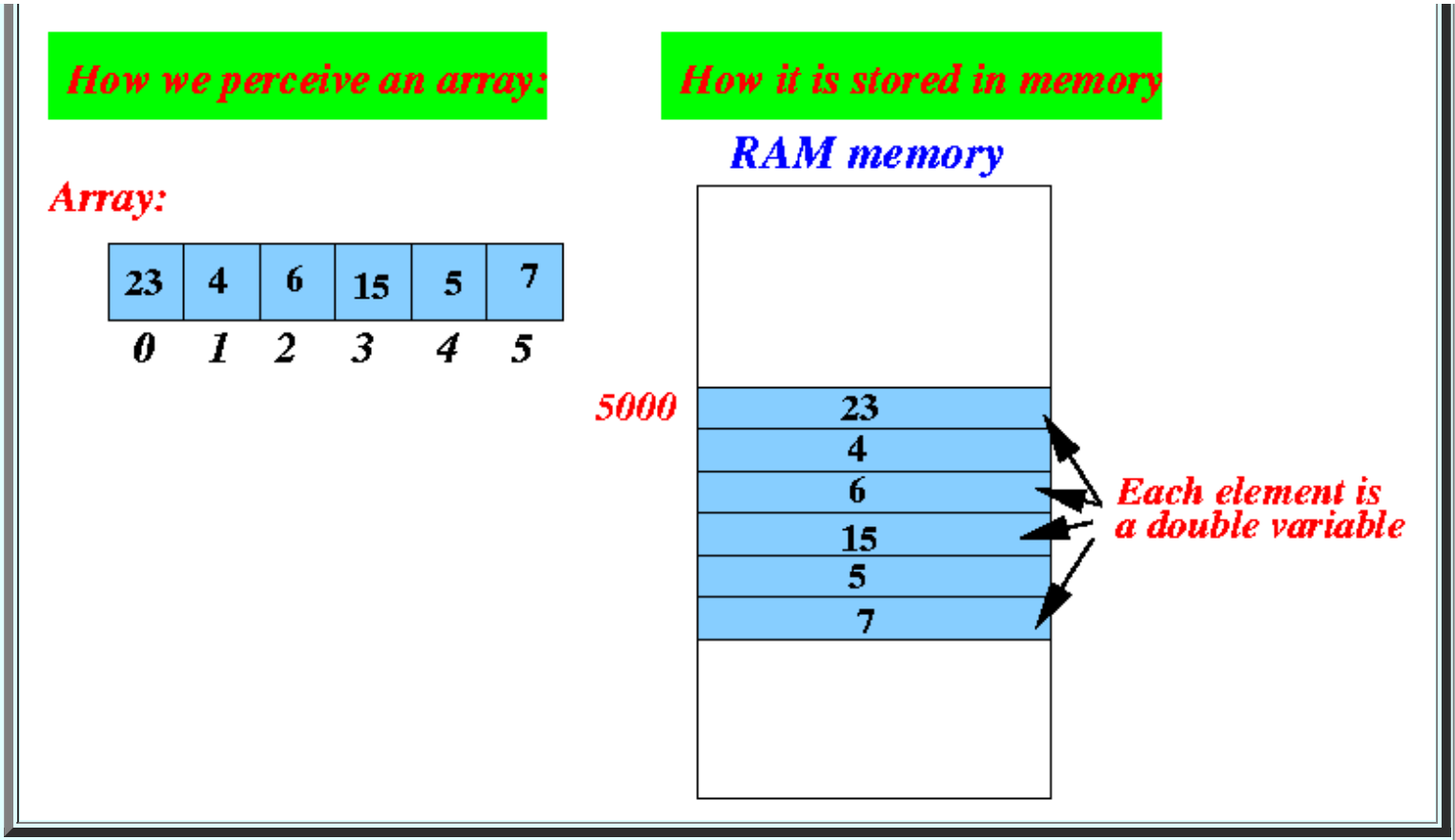
      - You must read `double[]` as a ***one* whole word**.

        Do not read it as: `double` + []

    - A **`double[]` typed variable** contains the ***address*** of the **start of an *array* of `double` variables**

    Example:

    - **Suppose** we have an ***array* of `double` variables** in **RAM memory:**

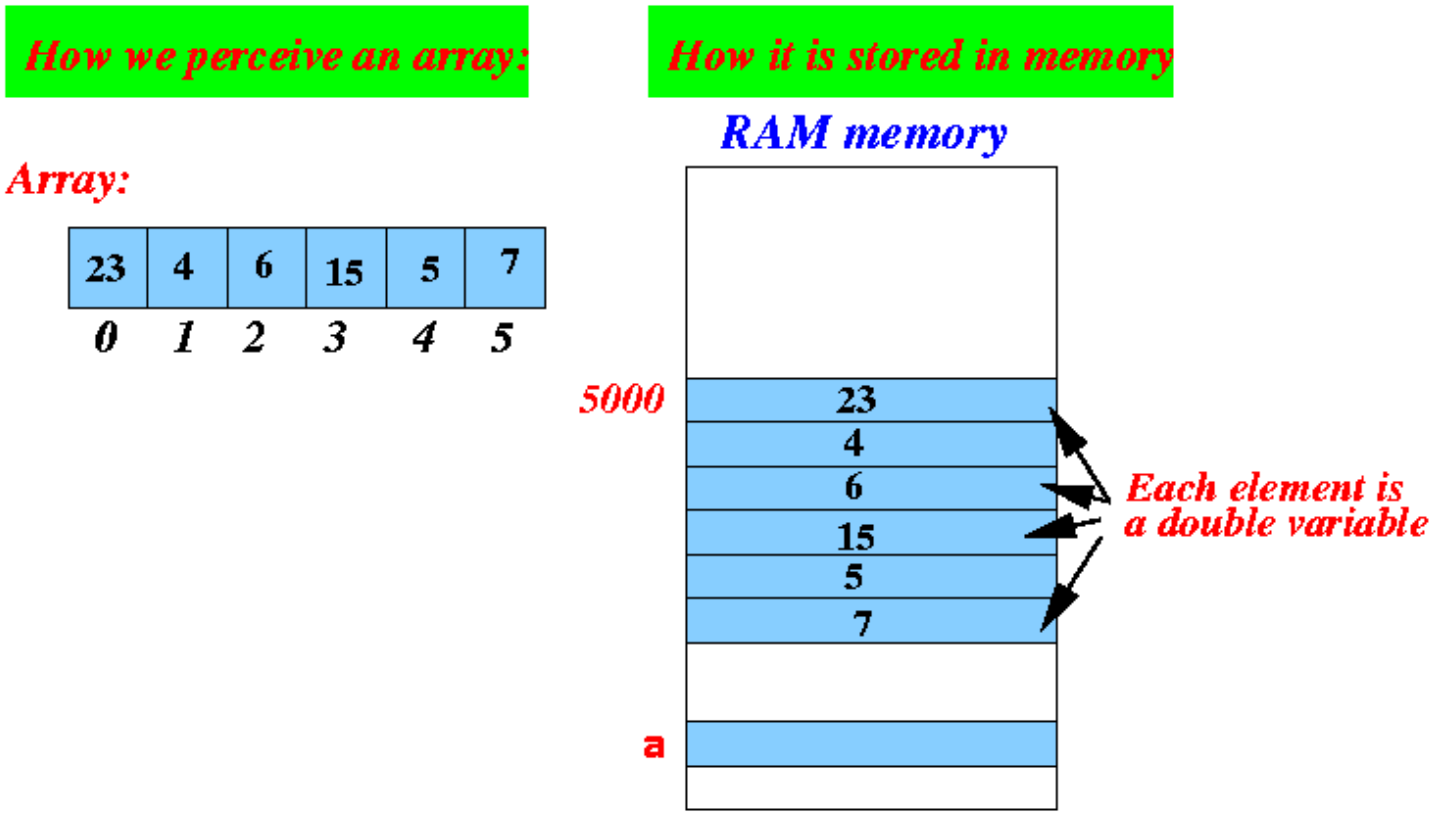


Suppose also that the *address* of the **start of this *array* of `double` variables** is address *5000*

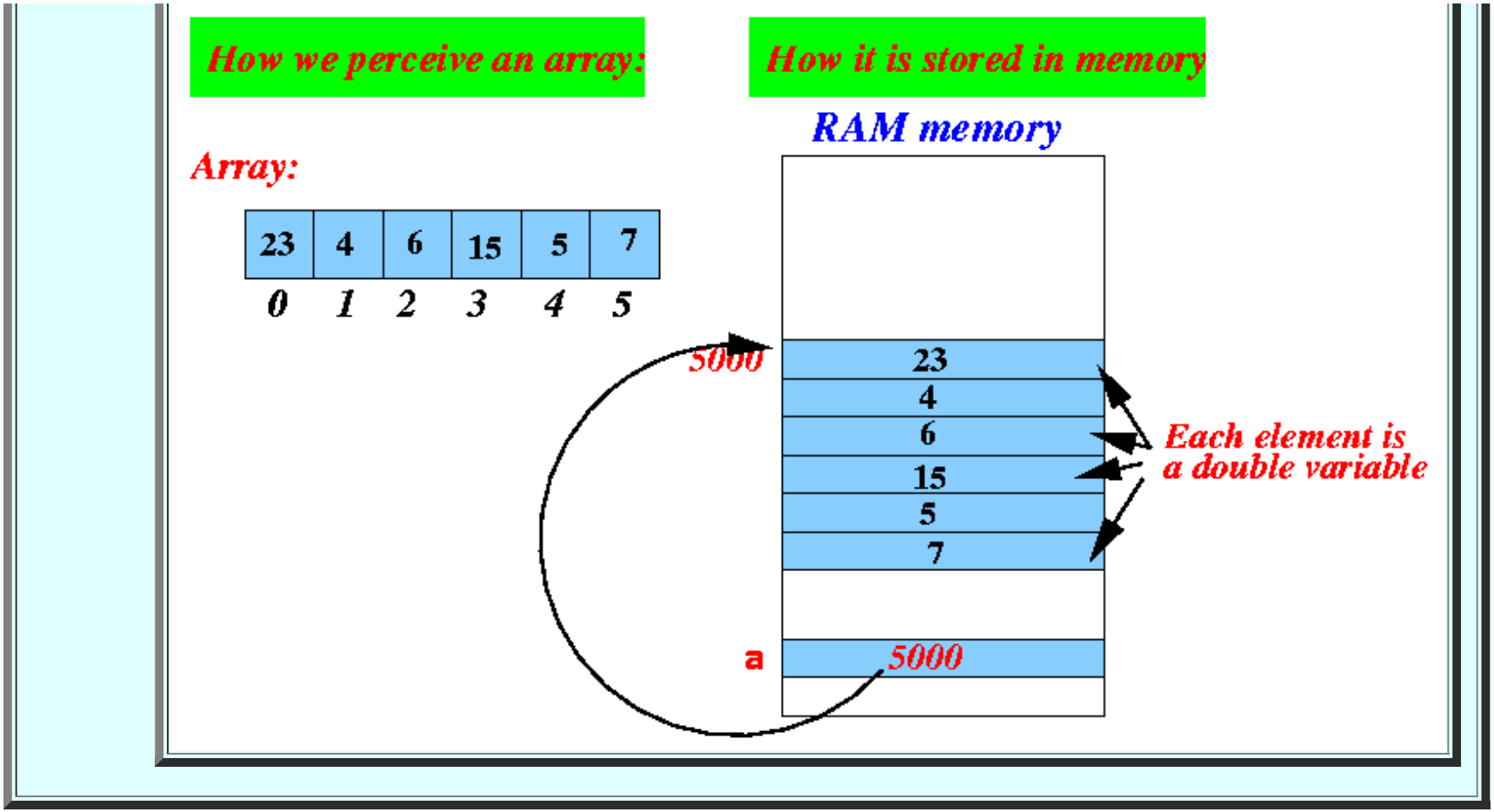
- **Suppose** we have defined the following variables:

```
double[] a;
```

Situation:



- The **data type** `double[]` allows us to **store** into the **variable** `a` the *address* of the **start of the array** (which is *5000*):



We can **now** use the **reference (5000)** stored in the **variable a** to ***locate*** the ***elements* of the array** !!!

- **Note:**

  - A **variable** of the type `double[]` **does *not*** contain a `double` **typed value**

    A `double[]` contains a **binary *integer* number** (an **address** is an ***integer*** !!) that is used to ***access* the computer RAM memory**

  - An **address** is **completely *different*** from a `double` **typed value**

    E.g., you would **not** perform a **multiplication** on the address ***5000***... it does not make any sense.

- **Other types of array object reference variables**
  - For **every *built-in* data type** of **Java**, there is a **corresponding reference type**

    Examples:

    - `double[]` = a **variable** of this type contains an **address** of an **array of `double` variables**
    - `int[]` = a **variable** of this type contains an **address** of an **array of `int` variables**
    - `boolean[]` = a **variable** of this type contains an **address** of an **array of `boolean` variables**
    - Etc.

  - In fact, you can ***append*** **"[ ]"** to ***any*** **data type**.

    BUT: It will **change** the ***meaning*** of the **data type** ***completely*** **Examples:**

    - `double a`: defines a variable named **`a`** that can contains a **double precision floating point number**

      As you know, the size of a `double` typed variable is **8 bytes**

    - `double[] a`: defines a variable named **`a`** that can contains an **address (reference)** of a collection (array) of double precision floating point numbers.

      The size of a **reference variable** is **4 bytes** (assuming that we are using a 32 bit address) !!!

- **The *new* operator**
  - We saw the usage of a ***new*** **Java operator** in the above example:

    ```
    a = new double[5] ;
    ```

  - ***Syntax*** **of the `new` operator:**

```
  new  ObjectType

or:

  new  DataType[ expression ]
```

Notes:

- The **data types** ***other than*** the **Java's built-in types** are known as ***object* types**
  - You **must** use an ***object* type** or an ***array* type** with the `new` **operator**

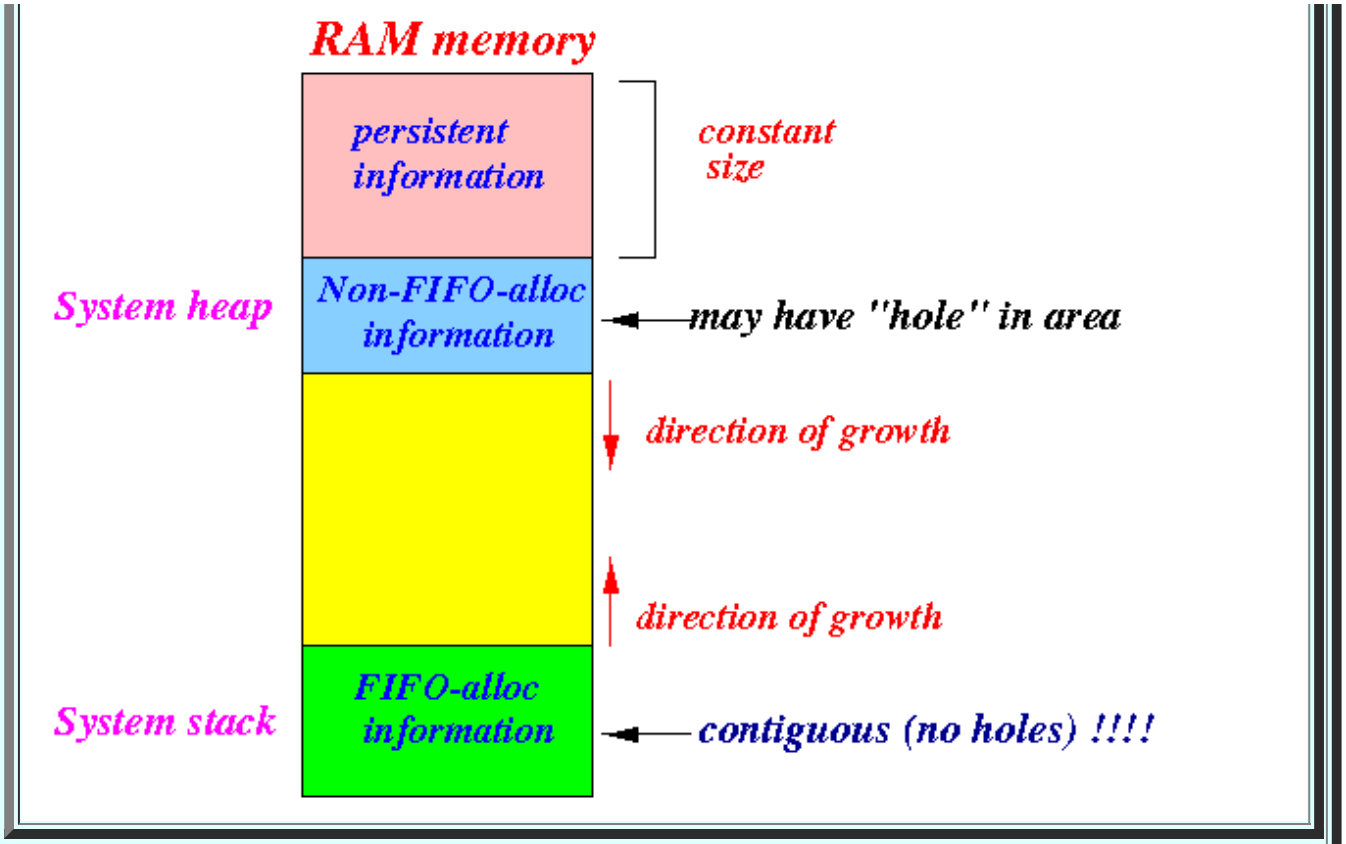
  (We will learn about ***object* types later**. For now, we use `new` on ***array* types**)

- **Warning:**
  - The `new` **operator** ***cannot*** be applied to **Java's built-in types** (such as `int`, `double`, etc.)

***Effect* of the `new` operator:**

- It ***first* creates (= reserve memory)** a **variable** of the **data type** `ObjectType`

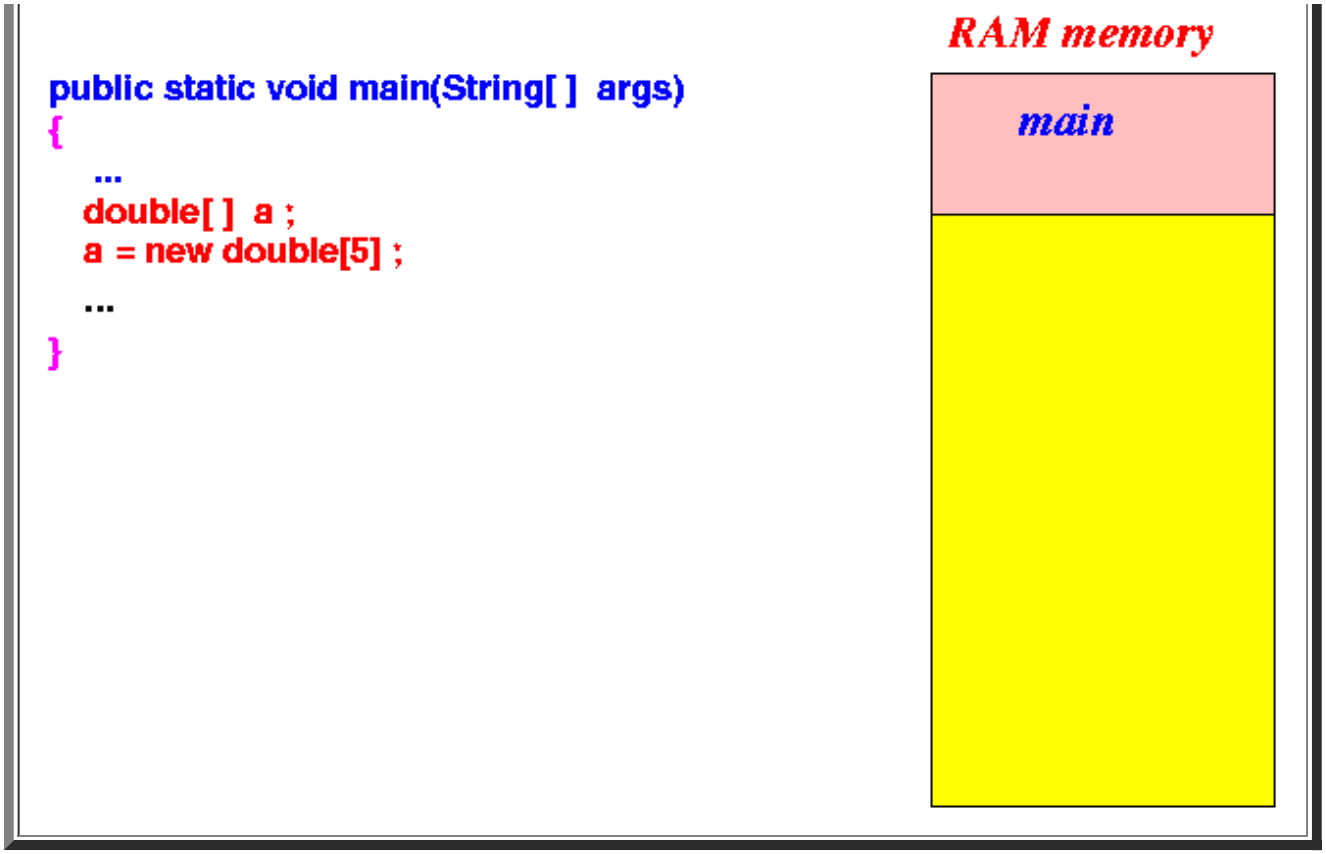
  The **variable** created by `new` **operator** will reside in the ***System Heap area*** (See: click here)



- Then it *return* (just like in a **method call**) the *address* **of the** *first* **allocated memory cell**.

We will **illustrate** the usage of the `new` **operator** by executing the above **example program**.

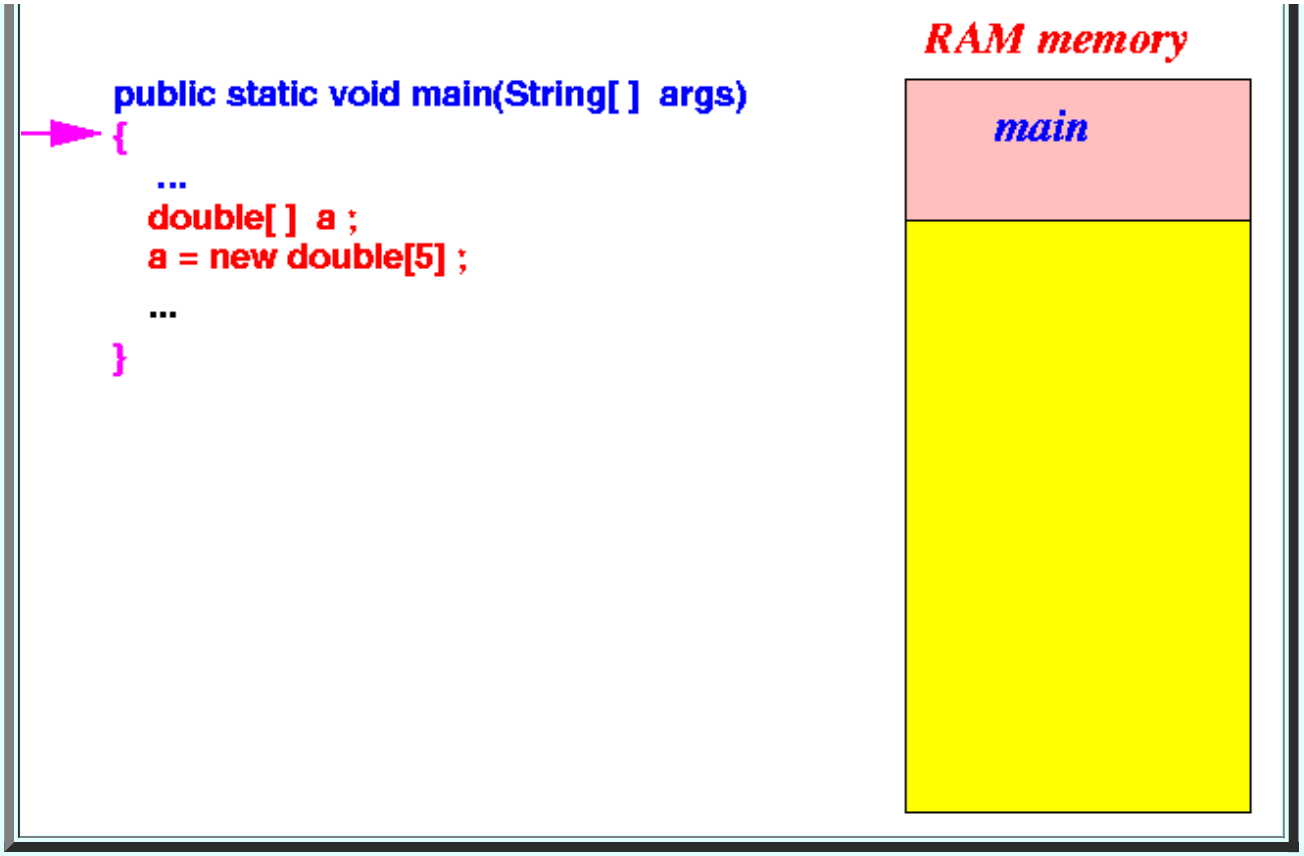
- **Illustration: definition of an array variable in Java**
  - **Example program: array definition in Java**



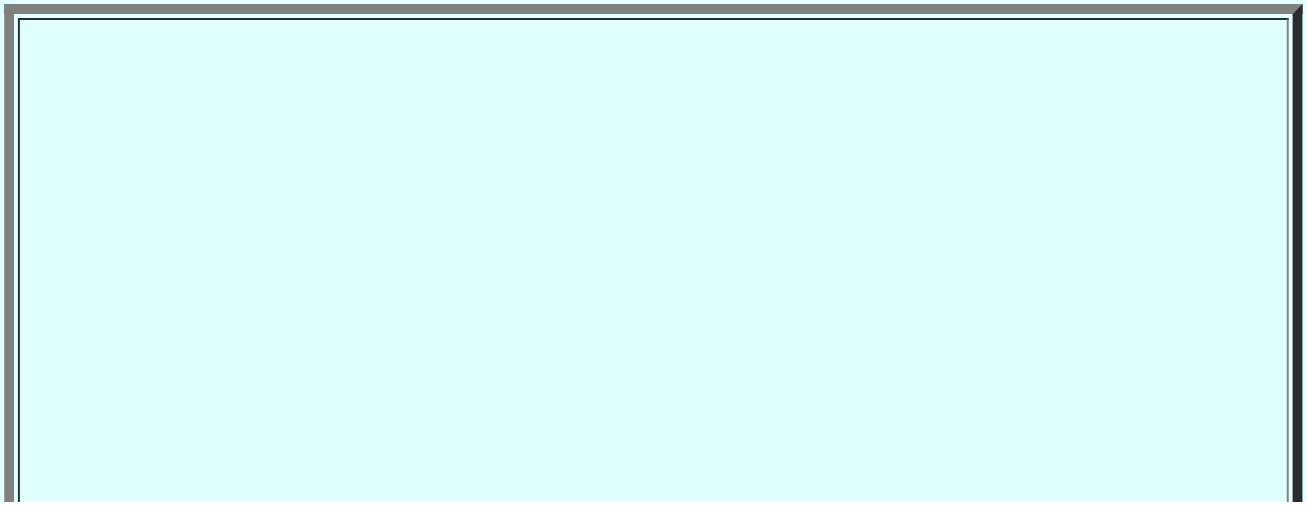
(The **enclosing *class* definition** has been **omitted** for **brevity**)

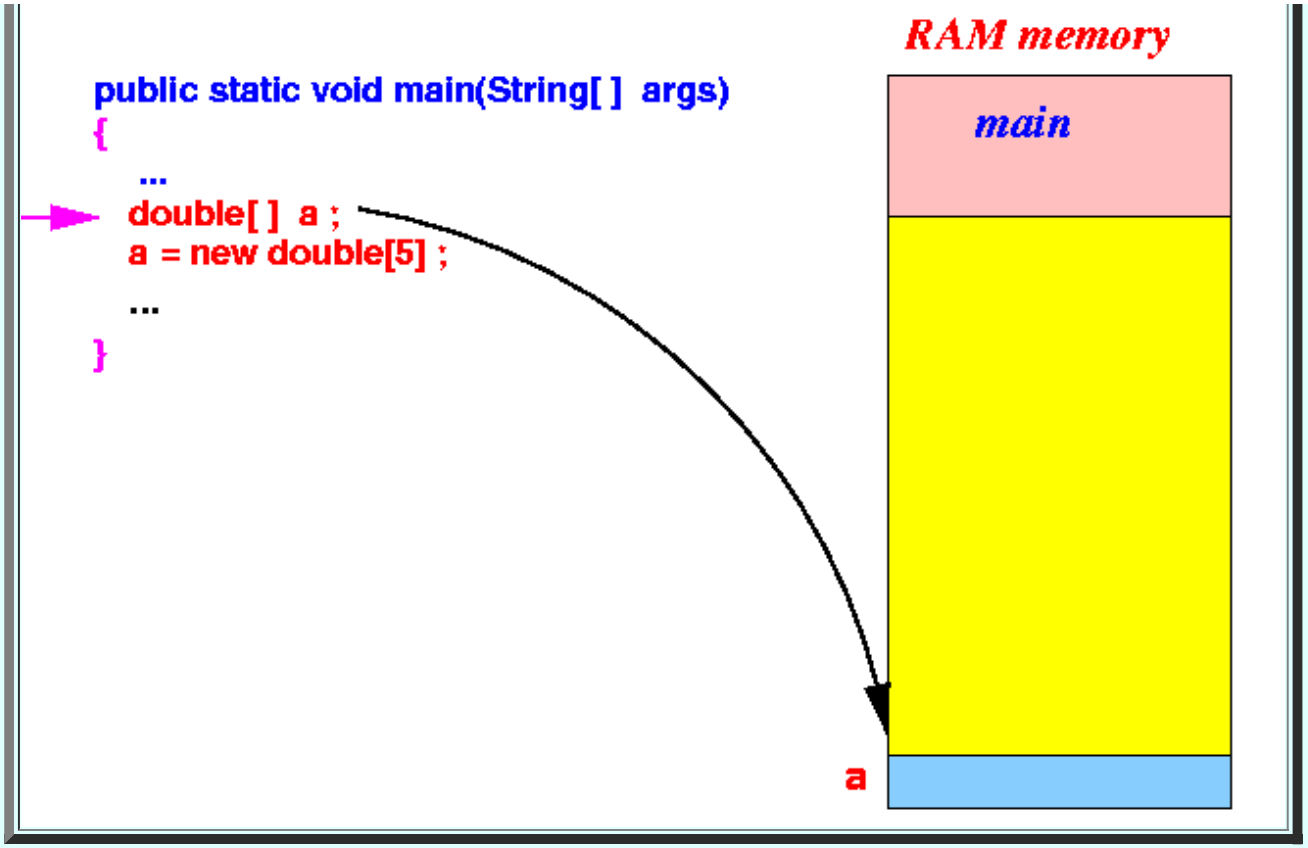
- **What happens when we define an *array* in Java:**

  - The **program execution** begins with the **`main` method**:

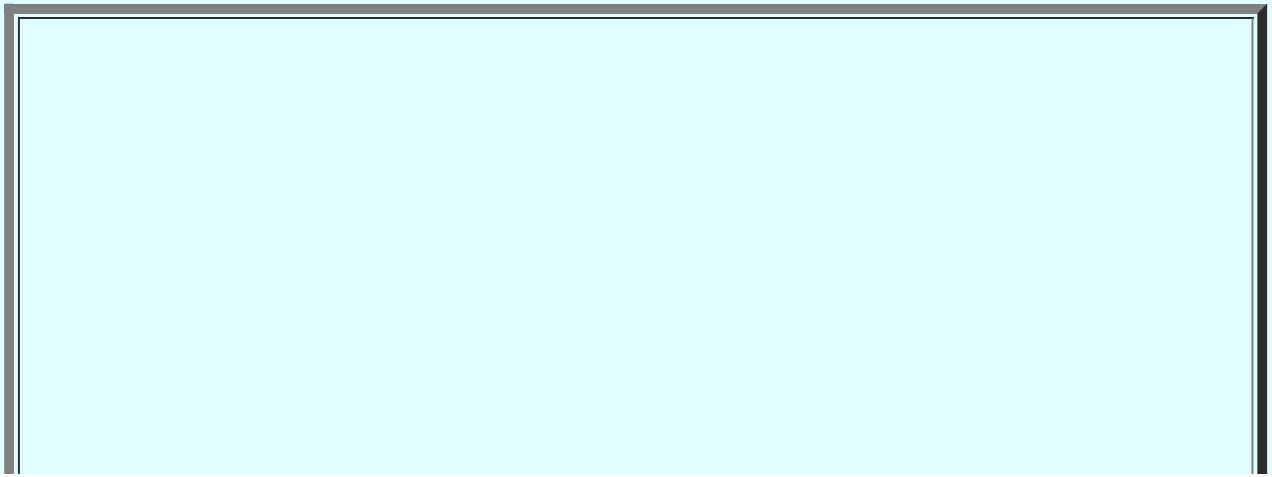


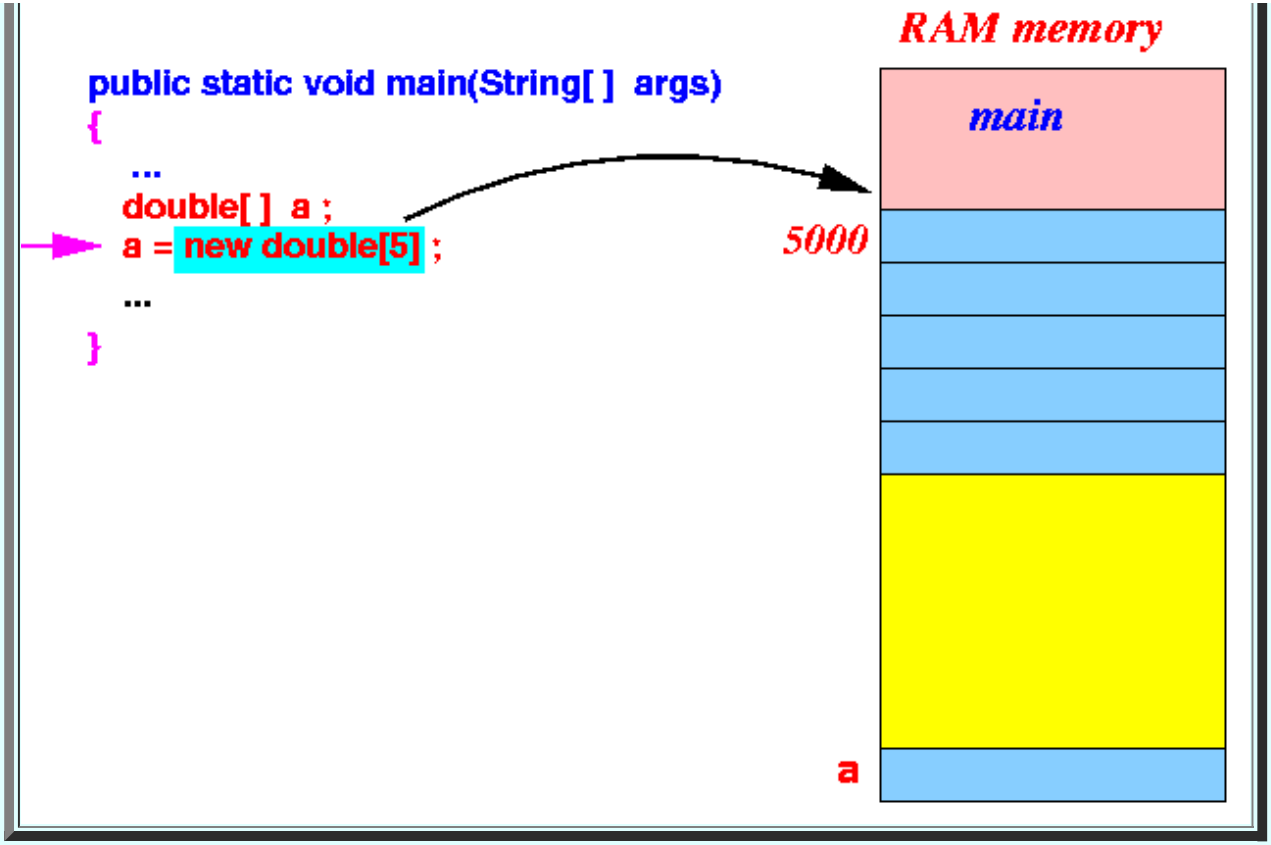
- When the **execution** reaches the definition `double[] a;`, the ***local variable*** `a` is **created** on the System Stack:





- When the **execution** reaches the definition `a = new double[5];` **statement**, the `new` **operator** will ***first*** create an ***array*** **of 5** `double` **variables** in the **System Heap**:



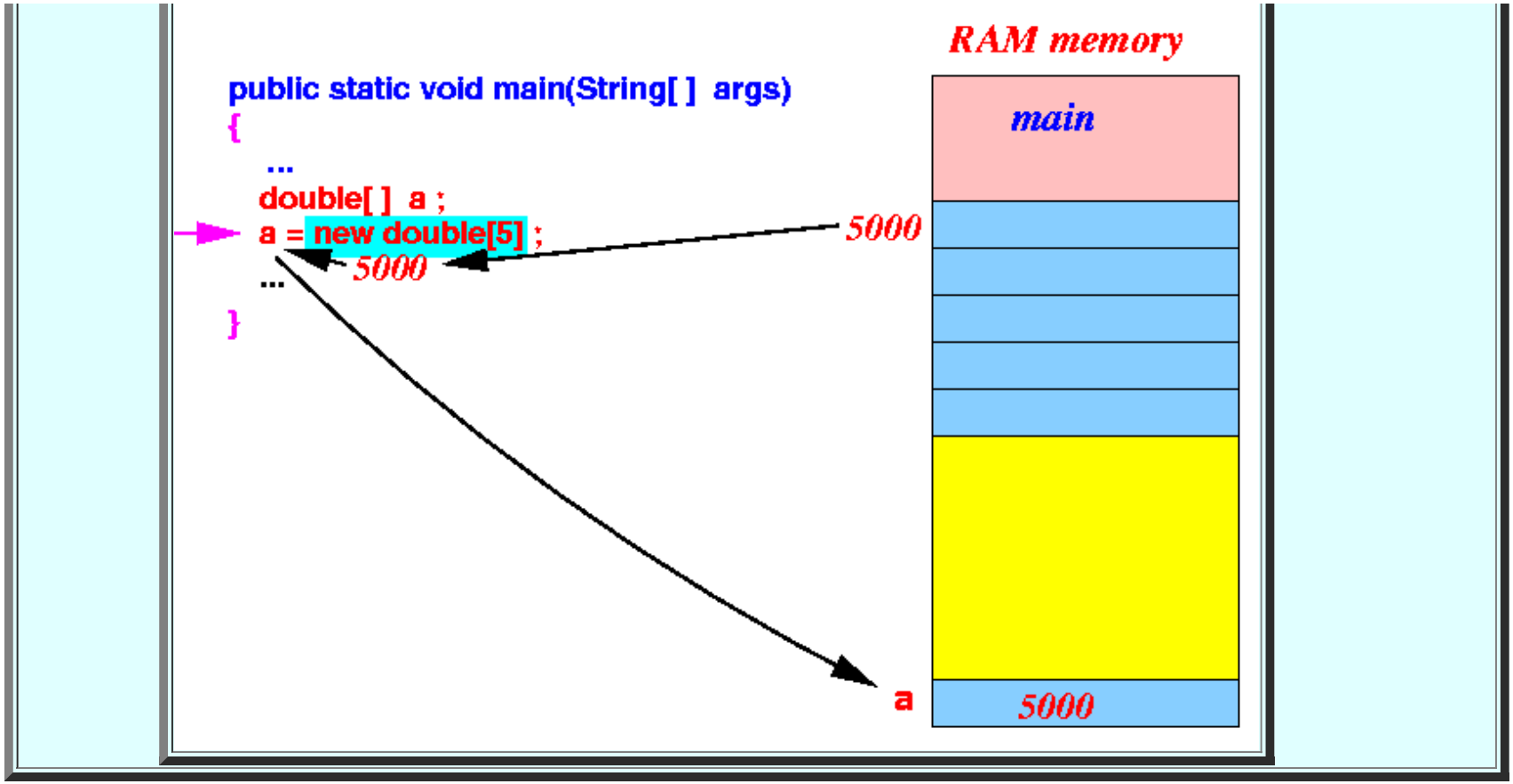


Suppose that the ***address*** of the **first element** of the array is ***5000***

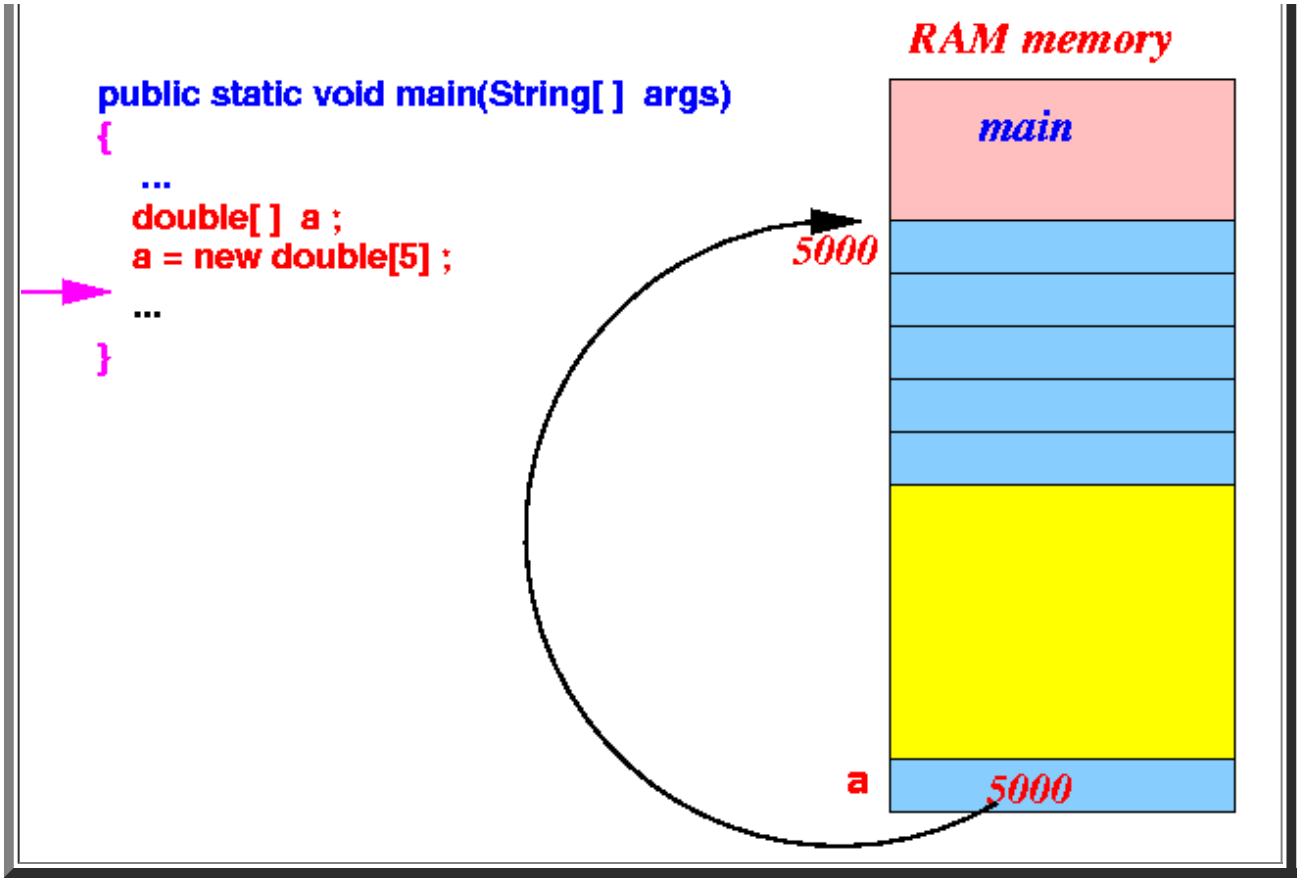
Then:

- The `new` **operator** ***returns*** the ***address*** of the **first element** of the array, which is ***5000***

The **return value** is ***subsequently*** used in the **assignment statement**:



- So.... **after** processing the **array definition instructions**, we have the following:



The **program** can ***now*** **access the array element** through its ***local variable*** `a` !!!

- **Shorter forms of array definitions**
  - **Previously**, we have seen the **2 step** array definition.

    Java allows you to **write *both* step** in ***one*** **combine statement**

  - **Example:**

```
double[] a ;                    Can be written as:
```

```
a = new double[5];
```

```
double[] a = new double[5];
```

- **Syntax of array definition in Java**
  - OK, after a length introduction, let's learn about the ***official* syntax** used to define an array in **Java**

    Syntax: defining an array

```
Two steps:

    datatype[]  variableName;              // Define a reference variable

    variableName = new datatype[ EXPR ];   // Create array of length EXPR
                                           // Assign starting address to
                                           // variableName


One step:

    datatype[] variableName = new datatype[ EXPR ];
```

- **An alternate form of array definition**
  - The **older programming language C** uses the following **syntax** to define an **array**:

```
double a[];       (instead of double[] a)
```

  - Since **Java** is **derived from C++** and **C++** is **derived from C**, the **Java compiler** will **accept** the following **syntax** to define **array variables**:

Alternate syntax to define array variable in Java

```
Two steps:
```

```
Two steps:

    datatype variableName[];              // Define a reference variable

    variableName = new datatype[ EXPR ];  // Create array of length EXPR
                                          // Assign starting address to
                                          // variableName
```

```
One step:

    datatype variableName[] = new datatype[ EXPR ];
```

- **Example:**

```
public class Avg3
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      double a[] = new double[5];      // Define an array of 5 elements

      double sum, avg;
      int i;                           // index

      for ( i = 0; i <= 4; i++ )
      {
         System.out.print("Enter a number: ");
         a[i] = in.nextDouble();            // Read in number
      }

      /* ------------------------------------------------
         Use the "running sum" algorithm to compute total
         ------------------------------------------------ */
      sum = 0.0;

      for ( i = 0; i <= 4; i++ )
      {
         sum = sum + a[i];
      }
      avg = sum/5;
      System.out.println(avg);        // Print average
   }
}
```

- ***Default* initial values in the array elements**
  - When an **array** is first **created**, ***every* element** contains an **(default) initial value**

    The **initial value *depends*** on the ***data type* of the array elements**

  - The **initial value** for each **data type**:

Data type	Initial value
A **number** type (e.g. `int, double, ...`)	**0**
`boolean`	**false**
`char`	the character **NULL**

- **Defining *initialize* array**
  - You can ***also* specify** your own **initial values** when you define an array.

    The **syntax** for defining an **initialized array** is:

```
datatype[] variableName = { list of initial values } ;
```

    Example:

```
double[] a = { 3.0, 4.5, 6.7, 3.9, 9.0 } ;
```

    Note:

    - The **length** of the array is **(automatically) determined** by the **number of values** in the list.
    - There is **no 2 steps** syntax to define an ***initialized* array**
    - **Very important**:

- you **do not (and cannot)** use the `new` **operator** in the definition of an ***initialized*** **array**