Arrays

- **array**: a structure composed of a contiguous sequence of variables, all of the same type.
- Individual variables are identified by *index values* or *indexes*.
- Index values are integers, (beginning with 0 in Java).
- *Components* or *elements* are the variables that comprise the array.
- They are a primitive type (*char*, *int*, *boolean*, *double*, etc.) or a reference type (reference to an object).
Array length

- The *length* of the array is the number of component variables that comprise it.
- With length $n$, the component variables are indexed 0 through $n-1$. 
Array accessing

- The length of an array is fixed when the array is created.
- Since contiguous memory is allocated for the variables, accessing a particular variable in the array requires a constant amount of time independent of the array length.
Array space allocation

- Suppose each variable in a particular array may take up four bytes, and memory is allocated starting at location 100.
- The address of the variable with index $i$ is $100 + 4 \times i$.
- The calculation of the address requires constant time, independent of the index and independent of the size of the array.
Arrays in Java are encapsulated in objects.

Array objects have a public `int component length`.

The class of an array is determined by the type of its component variables.

The class is written as the component type followed by a pair of brackets.

- `int[]`
- `Student[]`
Defining arrays

```java
int[] grades;
Student[] cs2125;
```

```java
int grades[];
Student cs2125[];
```

- The variable contains a reference to an array object.
- We use a constructor to create the array object.

```java
grades = new int[5];
cs2125 = new Student[5];
```
Defining arrays (cont.)
array components

- The component variables of the arrays are initialized with standard default values. e.g. 0 for `int`, `null` for references.
- The length given in the constructor need not (and generally should not) be a literal.

```java
cs2125 = new Student[studentCount];
grades = new int[4*cs2125.length];
```
Accessing array components

- A reference to the array is followed by an index value in brackets.

  ```java
  grades[3] = 100;
  grades[4] = grades[3]/2;
  cs2125[0] = new Student(…);
  cs2125[0].payFees(100);
  int i;
  for(i = 0; i < grades.length; i=i+1)
      grades[i] = 100;
  ```

- The index value must be an `int`, non-negative, and less than the length of the array.
Illegal array components

- If an array is referenced with an index that is negative or is not less than the array length, an ArrayIndexOutOfBoundsException is thrown.
Arrays with Objects

- **Component variables:**
  
  ```java
  private Object[] elements;
  private int size;
  ```

  - `elements` will hold the components of the array.
  - `size` will hold the current length of the list (not the maximum number of elements).
Accessing in loops

- For instance, if we write

```plaintext
for (j = i; j < size; j = j+1)
    elements[j+1] = elements[j];
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Index = 2: Value = A
- Index = 3: Value = B
- Index = 4: Value = C
- Index = 5: Value = D

![Diagram showing the process of accessing elements in a loop](image_url)
Advantages of arrays

- Elements can be accessed efficiently; get, append, and set all operate in constant time.
Limitations of arrays

- The array is static. The client must have a good idea of the ultimate size of a list when the list is created.
  - Too large means wasted space.
  - Too small causes failure.
- Operations **remove**, **add**, and **indexOf** all are linear.
- Conclusions
  - Not the best implementation for dynamic lists.
  - Often a good choice for static lists.
Dynamic arrays

- A *bounded* list has a maximum size.
- A *dynamic* list has no maximum size.
- A static array can be made into a dynamic array by simply creating a bigger array when the original array is full.
Vector class

- Vector class is a container class that encapsulates a “dynamic” use of arrays.

```java
public Vector ();
public Vector (int initialCapacity);
public Vector (int initialCapacity, int capacityIncrement);

New Vector (100,20)

default initialCapacity = 10
default capacityIncrement = 0
capacityIncrement of 0 means it doubles everytime it outgrows capacity.