Lecture 1

1D Arrays

(Searching and Sorting Methods)
- An array is a very useful data structure provided in programming languages. This is because we can have a quick and direct access to a specific element in the array by using its index.
• However, it has at least two limitations

  1. its size has to be known at compilation time,

  2. the data in the array are consecutive, which means that inserting an item into the array and deleting an item from the array require shifting other data in the array backwards and forwards, respectively.

• These limitations can be overcome by using linked structures (i.e., linked lists).
1. Searching Methods on Array
   1.1 Linear/Sequential Search Method
   1.2 Binary Search Method

2. Sorting Methods on Array
   2.1 Selection Sort
   2.2 Insertion Sort
Lecture Content

2. Sorting Methods on Array
   2.3 Bubble Sort
   2.4 Shaker Sort
   2.5 Shell Sort

3. String Class

4. Array of Objects

5. Pseudo Random Number Generator (PRNG)
1. Searching Methods on 1D Array

1.1 Linear/Sequential Search Method

1.2 Binary Search Method
Linear/Sequential Search Method

- **Problem:** A given array $a$ is $a[0], \ldots, a[n-1]$ and a given element (also called search key or target) $x$. Specify whether $x$ exists in the array $a$.

- Basically, we have two methods of searching, namely **linear** (also called **sequential**) search method and **binary** search method.
Main idea: We look through the array sequentially to make the comparison between \( x \) and each element \( a[i] \) of the given array. The search method returns

-1 if not found

\[ i \ (0 \leq i \leq n - 1) \] if \( x \) is located at position \( i \) in the array \( a \)

• The time efficiency of linear search is in \( \Theta(n) \).
Algorithm `linearSearch`

Input: An array `a[0..n - 1]` of `n` elements and an element `x`.

Output: `i` if `x = a[i]`, `0 ≤ i ≤ n - 1`, and `-1` otherwise.

1. `i ← 0`
2. while `(i < n) and (x ≠ a[i])`
3. `i ← i + 1`
4. end while
5. if `i < n` then return `i` else return `-1`
Algorithm **linearSearch**

**Input:** An array \( a[0..n - 1] \) of \( n \) elements and an element \( x \).

**Output:** \( i \) if \( x = a[i] \), \( 0 \leq i \leq n - 1 \), and -1 otherwise.

1. for \( i \leftarrow 0 \) to \( n - 1 \)
2. if \( x = a[i] \) then
3. return \( i \)
4. return -1
Binary Search Method

**Precondition:** the given array must be sorted (i.e., elements are in order).

**Main idea:**
- Use a search procedure based on applying the standard divide-and-conquer paradigm.
Main idea:

- Divide the set of items into two parts, determine to which of the two parts the search key belongs, then concentrate on that part.

- A reasonable way to divide the sets of items into parts is to keep the items sorted, then to use indices into the sorted array to delimit the part of the array being worked on.
Binary Search Method

Main idea:
- This search technique is called binary search. The search method returns
  -1 if not found
  \( i \) (\( 0 \leq i \leq n - 1 \)) if \( x \) is placed at position \( i \) in the array \( a \)
- The time efficiency of binary search is in \( \Theta(\log n) \) (i.e., at most \( \lceil \log n \rceil + 1 \) comparisons).
Binary Search Method

\[ m = \lfloor (L + r) / 2 \rfloor \]

If \( x = a[m] \), return \( m \)

If \( x < a[m] \), search \( x \) in left half \( a[0..m-1] \)

If \( x > a[m] \), search \( x \) in right half \( a[m+1..r] \)
Algorithm binarySearch

Input: An array $a[0..n - 1]$ of $n$ elements sorted in nondecreasing order and an element $x$.

Output: $i$ if $x = a[i]$; $0 \leq i \leq n - 1$, and -1 otherwise.
Binary Search Method

1. \( L \leftarrow 0; \, r \leftarrow n - 1; \, i \leftarrow -1; \)
   
   // \( L = \) left, \( r = \) right, \( m = \) middle

2. while \((L \leq r) \) and \((i = -1)\)

3. \( m \leftarrow \lfloor (L + r) / 2 \rfloor \)

4. \( \text{if } x = a[m] \text{ then } i \leftarrow m \)

5. \( \text{else if } x < a[m] \text{ then } r \leftarrow m - 1 \)

6. \( \text{else } L \leftarrow m + 1 \)

7. end while

8. return \( i \)
**Binary Search Method**

1. \( L \leftarrow 0; \ r \leftarrow n - 1; \)
   
   // \( L = \) left, \( r = \) right, \( m = \) middle

2. while \((L \leq r)\)

3. \( m \leftarrow \left\lfloor (L + r) / 2 \right\rfloor \)

4. if \( x = a[m] \) then return \( m \)

5. else if \( x < a[m] \) then \( r \leftarrow m - 1 \)

6. else \( L \leftarrow m + 1 \)

7. end while

8. return -1
2. Sorting Methods on 1D Array

• The efficiency of data handling can often be substantially increased if the data are sorted according to some criteria of order.

• Two critical properties of sorting algorithms are the number of comparisons and the number of data movements.

• Problem: Assume that we want to sort the array \( a \) containing \( a_0, a_1, \ldots, a_{n-1} \) in ascending order.
Selection Sort

• Selection sort scans the array to find its smallest element and swap it with the first element. Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second elements. Generally, on pass $i$ ($0 \leq i \leq n - 2$), find the smallest element in $a[i..n - 1]$ and swap it with $a[i]$.

• The time complexity of selection sort is in $\Theta(n^2)$ (i.e., exactly $n(n - 1)/2$ comparisons).
Selection Sort

Algorithm

Step 1: Find the smallest (largest) element in the current array.

Step 2: Swap it with the first (last) element.

Step 3: Eliminate the smallest (largest) element from the current array.

Step 4: Repeat Steps 1 - 3 until the number of elements in the current array is equal to 1.
Selection Sort

Notes:

Initially \((i = 0)\): the array has \(n\) elements.

After loop 1 \((i = 1)\): array has \((n - 1)\) elements.

... 

After loop \(n - 1\) \((i = n - 1)\): array has 1 element.

Therefore, we just repeat \((n - 1)\) times and the last element of the array at time \((n - 1)\) is maximal, otherwise this element must be selected and removed at the time \((n - 2)\).
Algorithm selectionSort

Input: An array $a[0..n - 1]$ of $n$ elements.
Output: $a[0..n - 1]$ sorted in nondecreasing order.
Selection Sort

1. for $i \leftarrow 0$ to $n - 2$
2. $k \leftarrow i$
3. for $j \leftarrow i + 1$ to $n - 1$ // find $i$th smallest
4. if $a[j] < a[k]$ then $k \leftarrow j$
5. end for
6. if $k \neq i$ then interchange $a[i]$ and $a[k]$
7. end for
Example of Selection Sort

<table>
<thead>
<tr>
<th>89</th>
<th>45</th>
<th>68</th>
<th>90</th>
<th>29</th>
<th>34</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>45</td>
<td>68</td>
<td>90</td>
<td>29</td>
<td>34</td>
<td>89</td>
</tr>
<tr>
<td>17</td>
<td>29</td>
<td>68</td>
<td>90</td>
<td>45</td>
<td>34</td>
<td>89</td>
</tr>
<tr>
<td>17</td>
<td>29</td>
<td>34</td>
<td>90</td>
<td>45</td>
<td>68</td>
<td>89</td>
</tr>
<tr>
<td>17</td>
<td>29</td>
<td>34</td>
<td>45</td>
<td>90</td>
<td>68</td>
<td>89</td>
</tr>
<tr>
<td>17</td>
<td>29</td>
<td>34</td>
<td>45</td>
<td>68</td>
<td>90</td>
<td>89</td>
</tr>
<tr>
<td>17</td>
<td>29</td>
<td>34</td>
<td>45</td>
<td>68</td>
<td>89</td>
<td>90</td>
</tr>
</tbody>
</table>
Insertion Sort

• This sorting method is analogous to the way many people sort their hands when playing cards: pick up one item at a time and move it into place among the items already picked up and sorted.

• Assume that we have \((i - 1)\) cards sorted, now we pick up the \(i\)th card and how to sort \(i\) cards. We can compare the \(i\)th card with previous cards of \((i - 1)\), \((i - 2)\), ..., 0 to find out the right position of the \(i\)th card and insert it into this location.
Insertion Sort

• We insert $a_i$ into the sorted array $a$ containing $a_0, a_1, \ldots, a_{i-1}$ so that after inserting the new array $a$ containing $a_0, a_1, \ldots, a_{i-1}, a_i$ remains in order.

Notes

- $a_0$ is an ordered array so we consider from the second element $a_1$ (i.e., $1 \leq i \leq n - 1$).
- Insert $a_i$ into $a_0, a_1, \ldots, a_{i-1}$: successively compare $a_i$ with $a_{i-1}$, if $a_i < a_{i-1}$ then exchange the position of two cards $a_i$ and $a_{i-1}$.
Algorithm insertionSort

Input: An array $a[0..n - 1]$ of $n$ elements.
Output: $a[0..n - 1]$ sorted in nondecreasing order.
Insertion Sort

1. for $i \leftarrow 1$ to $n - 1$
2. $x \leftarrow a[i]$
3. $j \leftarrow i - 1$
4. while $(j \geq 0)$ and $(a[j] > x)$
5. $a[j + 1] \leftarrow a[j]$
6. $j \leftarrow j - 1$
7. end while
8. $a[j + 1] \leftarrow x$
9. end for
Example of Insertion Sort

- The time complexity of insertion sort is in $\Theta(n^2)$ (i.e., at most $n(n-1)/2$ comparisons).

<table>
<thead>
<tr>
<th>89</th>
<th>45</th>
<th>68</th>
<th>90</th>
<th>29</th>
<th>34</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>89</td>
<td>68</td>
<td>90</td>
<td>29</td>
<td>34</td>
<td>17</td>
</tr>
<tr>
<td>45</td>
<td>68</td>
<td>89</td>
<td>90</td>
<td>29</td>
<td>34</td>
<td>17</td>
</tr>
<tr>
<td>45</td>
<td>68</td>
<td>89</td>
<td>90</td>
<td>29</td>
<td>34</td>
<td>17</td>
</tr>
<tr>
<td>45</td>
<td>68</td>
<td>89</td>
<td>90</td>
<td>34</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>45</td>
<td>68</td>
<td>89</td>
<td>90</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>34</td>
<td>45</td>
<td>68</td>
<td>89</td>
<td>90</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>29</td>
<td>34</td>
<td>45</td>
<td>68</td>
<td>89</td>
<td>90</td>
</tr>
</tbody>
</table>
Main idea: move backwards from the last element (i.e., $n - 1$) to the second one (i.e., $a[1]$). In other words, move up from the bottom element to the second top element.

(Alternatively, move forwards from the first element (i.e., $a[0]$) to the second last element (i.e., $n - 2$). In other words, move down from the top element to the second last last element.)
Bubble Sort

Loop 1: move up to the 2nd element $a[1]$

Loop 2: move up to the 3rd element $a[2]$

...

Loop n - 1: move up to the n-th element $a[n - 1]$

While moving up, compare each pair of neighboring elements. If the below element is lighter than the above element then interchange them. This process is repeated $(n - 1)$ times then the given array will be sorted.
Algorithm 1 bubbleSort

Input: An array $a[0..n-1]$ of $n$ elements.
Output: $a[0..n-1]$ sorted in nondecreasing order.

1. for $i \leftarrow 1$ to $n-1$ // $(n-1)$ iterations
2. for $j \leftarrow n-1$ downto $i$ // move up
3. if $a[j] < a[j-1]$ then interchange $a[j-1]$ and $a[j]$ // push small element up
4. end for
5. end for
Algorithm 2 bubbleSort

Input: An array $a[0..n - 1]$ of $n$ elements.

Output: $a[0..n - 1]$ sorted in nondecreasing order.

1. for $i ← 0$ to $n - 2$ // $(n - 1)$ iterations
2. for $j ← 0$ to $n - 2 - i$ // move down
3. if $a[j] > a[j + 1]$ then
   interchange $a[j]$ and $a[j + 1]$
4. end for
5. end for
Algorithm 3 bubbleSort

Input: An array $a[0..n - 1]$ of $n$ elements.
Output: $a[0..n - 1]$ sorted in nondecreasing order.

1. $i \leftarrow 1$; $\text{sorted} \leftarrow \text{false}$
2. while $(i \leq n - 1)$ and (not $\text{sorted}$)
3. \hspace{1em} $\text{sorted} \leftarrow \text{true}$
4. for $j \leftarrow n - 1$ downto $i$
5. \hspace{1em} if $a[j] < a[j - 1]$ then
6. \hspace{2em} interchange $a[j - 1]$ and $a[j]$
Bubble Sort

7. \( \text{sorted} \leftarrow \text{false} \)
8. end if
9. end for
10. \( i \leftarrow i + 1 \)
11. end while

- The time complexity of bubble sort is in \( \Theta(n^2) \).
Comments on Bubble sort method:
+ Light (small) elements move up quickly, heavy (large) ones go down slowly.
+ Still performs on a sorted array.

Thought: Two tasks are performed in one iteration

- The time complexity of shaker sort is in \( \Theta(n^2) \).
Shaker Sort

Task 1: (move up (↑)): push the light element up and store the last position where swapping is performed to change the water surface down to this position.

Task 2: (move down (↓)): push the heavy element down and save the last position where swapping is performed to change the bottom up to this position.

The algorithm is terminated when water surface $L = 1$ and bottom $r = n - 1$ pass through.
Algorithm in the form of pseudocode

while (L <= r) {
    // push the light element up
    // store the last swapping position
    // push the heavy element down
    // store the last swapping position
} // while (L <= r)
Algorithm `shakerSort`

Input: An array `a[0..n - 1]` of `n` elements.

Output: `a[0..n - 1]` sorted in nondecreasing order.
Shaker Sort (Cocktail Sort)

1. $L \leftarrow 1; r \leftarrow n - 1; k \leftarrow n - 1$;

2. while $(L \leq r)$

3. for $i \leftarrow r$ downto $L$

4. if $a[i] < a[i - 1]$ then

5. interchange $a[i - 1]$ and $a[i]$

6. $k \leftarrow i$

7. end if

8. end for

9. $L \leftarrow k + 1$
Shaker Sort (Cocktail Sort)

10. for $j \leftarrow L$ to $r$
11. if $a[j - 1] > a[j]$ then
12. interchange $a[j - 1]$ and $a[j]$
13. $k \leftarrow j$
14. end if
15. end for
16. $r \leftarrow k - 1$
17. end while
Shell Sort

- Shell sort solves the sorting problem by dividing the original array into subarrays, sorting them separately, and then dividing them again to sort the new subarrays until the whole array is sorted. The goal was to reduce the original problem to subproblems that can be solved more easily and quickly.
Shell Sort

• The time complexity of shell sort is in $\Theta(n^{3/2})$ for the Hibbard’s sequence $1, 3, 7, ..., 2^k - 1$.

• The time complexity of shell sort is in $\Theta(n^{4/3})$ for the Sedgewick’s sequence $1, 8, 23, ..., 4^{k+1} + 3 \times 2^k + 1$. 
Shell Sort

Algorithm

Step 1: Divide the given array $a_0, \ldots, a_{n-1}$ into subarrays. The step (distance) between elements in each subarray is $h_i$ ($h_i = 9, 5, 3, 1$)

Step 2: Sort each subarray using insertion sort.

Step 3: Repeat Steps 1 and 2 for the next value of $h_i$. 
Example of Shell Sort

• Sort the array $a$ containing numbers 10 8 6 20 4 3 22 1 0 15 16

Assume that $h_i = 5$. 
Example of Shell Sort

| 10  | 8  | 6  | 20 | 4  | 3  | 22 | 1  | 0  | 15 | 16 |

Five subarrays before sorting

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Five subarrays after sorting

<table>
<thead>
<tr>
<th>3</th>
<th>10</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

The given array becomes 3 8 1 0 4 10 22 6 20 15 16
Example of Shell Sort

3  8  1  0  4  10  22  6  20  15  16

<table>
<thead>
<tr>
<th>Three subarrays before sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Three subarrays after sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

The given array becomes 0 4 1 3 6 10 15 8 20 22 16

47
Example of Shell Sort

0  4  1  3  6  10  15  8  20  22  16

$h_i = 1$

The sorted array is

0  1  3  4  6  8  10  15  16  20  22
3. String Class

- A sequence of characters separated by double quotes is **String** constants.

- **String** is a class, we can create an instance and give it a name. For example,

```java
String name;
name = new String("Andrew Rudder");
```
3. String Class

- We can also create a new `String` object, for example, in this way:

  ```java
  String name;
  name = "Kris Manohar";
  ```

- Some methods defined in the `String` class are `substring()`, `length()`, and `indexOf()`, `lastIndexOf()`.
We can extract a substring from a given string by specifying the beginning and ending positions (indexed, subscripted, or numbered from 0, 1, ..., n - 1).

For example,

```java
String text;

text = "Espresso";

System.out.println(text.substring(2, 7)); // "press"
```
3. String Class

- We can find out the number of characters in a String object by using the `length()` method.

- For example, if the name `text` refers to a string `Espresso`, then

```java
    text.length()
```

will return the value 8.
3. String Class

- To locate the index position of a substring within another string, we use the `indexOf()` method.

- For example, if the name `text` refers to a string `Hello World`, then

```
  text.indexOf("World")
```

will return the value 6, the index position of the first character of the string `World`.

- If the searched substring is not located in the string, then -1 is returned.
3. String Class

- Notice that the search is done in a case-sensitive manner. Thus,

```java
text.indexOf("world")
```

will return -1.

- If there is more than one occurrence of the same substring, the index position of the first character of the first matching substring is returned.
3. String Class

- **String concatenation**: We can create a new string from two given strings by concatenating the two strings.

- We use the plus symbol `+` for string concatenation.
3. String Class

• Two strings $s_1$ and $s_2$ can be compared by using the `compareTo()`, `compareToIgnoreCase()`, `equals()`, `equalsIgnoreCase()` or method.

$s_1$.compareToIgnoreCase($s_2$) returns

$< 0$ if $s_1 < s_2$ ; $0$ if $s_1 = s_2$; $> 0$ if $s_1 > s_2$

$s_1$.equals($s_2$) returns

`true` if $s_1 = s_2$

`false` if $s_1 \neq s_2$
3. String Class

- Example

```java
String a[] = {"Bahadoor",
             "Mathura", "Chamansingh",
             "Nanton", "Austin", "Lewis",
             "Gumaia", "Heeralal"};
```
3. String Class

• Example

```java
public static int sequentialSearch(String[] a, String key) {
    int i = 0, n = a.length;
    //while ( (i < n) &&
    (a[i].compareTo(key) != 0) )
    //while ( (i < n) &&
    (a[i].equals(key) == false) )
```
3. String Class

    //while ( (i < n) &&
    (a[i].equalsIgnoreCase(key) == false) )
    while ( (i < n) &&
    (a[i].compareToIgnoreCase(key) != 0) )
        i++;
    if (i < n) return i;
    else return -1;
}
4. Array of Objects

class Book {

    private String bt;  // book title
    private String fn;  // author’s first name
    private String ln;  // author’s last name
    private String pub;  // publisher
    private String isbn;  // ISBN
    private int py;      // publication year
    private double price;  // book price

    ...

}  // end class Book
public Book() {

    // constructor
    bt = ""; fn = ""; ln = "";
    pub = ""; isbn = "";
    py = 2000;
    price = 0.0;
}


public void inputBook() {

    // input information for one book
    Scanner in = new Scanner(System.in);
    System.out.print("Book Title: ");
    bt = in.nextLine();
    ...
    System.out.print("Pub. Year : ");
    py = in.nextInt();
    System.out.print("Price : ");
    price = in.nextDouble();
}

public void displayBook() {

    // display one book on screen
    System.out.println("Book Title: " + bt);
    System.out.println("Author : " + fn + " " + ln);
    System.out.println("Publisher : " + pub);
    System.out.println("ISBN : " + isbn);
    System.out.println("Pub. Year : " + py);
    System.out.printf("Price : %.2f", price);
    System.out.println("\n-------------------------");
}
4. Array of Objects

```java
public String getBookTitle() {
    return bt;
}

public void setBookTitle(String BookTitle) {
    bt = BookTitle;
}
```
class BookList {

    private Book[] a; // reference to array
    private int n; // number of data items

    ...

} // end class BookList
public BookList(int max) {
    // constructor
    a = new Book[max];
    // create the array
    n = a.length;
}
public void inputBookList() {
    int m;
    for (int i = 0; i < n; i++) {
        System.out.printf("Enter data for book %d\n", i + 1);
        a[i] = new Book();
        a[i].inputBook();
        System.out.println("-------------------------");
    }
    System.out.println();
}
public void displayBookList() {
    for (int i = 0; i < n; i++)
        a[i].displayBook();
}
4. Array of Objects

class BookArrayApp {
    public static void main(String[] args) {
        int m;
        Scanner in = new Scanner(System.in);
        BookList list = null; // reference to array
        System.out.print(
            "Enter the number of books: ");
        m = in.nextInt();
        in.nextLine(); // flush buffer
4. Array of Objects

```java
list = new BookList(m); // create the array
list.inputBookList();
list.displayBookList();
}
} // end class BookArrayApp
```
class PRNGApp {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = 10, min = 5, Max = 10;
        // generate n double numbers in [0.0, 1.0)
        for (int i = 1; i <= n; i++) {
            double x = Math.random();
            System.out.printf("%.2f ", x);
        }
        System.out.println();
    }
}
5. Pseudo Random Number Generator

// generate n integer numbers in [0, n)
for (int i = 1; i <= n; i++) {
    int x = (int)(Math.random() * n);
    System.out.printf("%d ", x);
}
System.out.println();

// generate n integer numbers in [0, n]
for (int i = 1; i <= n; i++) {
    int x = (int)(Math.random() * (n + 1));
    System.out.printf("%d ", x);
}
System.out.println();
5. Pseudo Random Number Generator

// generate n integer numbers in [min, Max]
for (int i = 1; i <= n; i++) {
    int x = min + (int)( Math.random() *
                       ((Max - min) + 1) );
    System.out.printf("%d ", x);
}
System.out.println();
} // end main()
} // end class PRNGApp
1. The selection problem: Write a program to find the $k$th smallest element of an $n$-element array of integers.
2. Implement two methods of searching and five methods of sorting mentioned in the lecture.
3. Use theoretical topics taught in the class to write a simple program of student management system using an array of student objects. Required functionalities are as follows.
Exercises

1. Input student list
2. Output student list
3. Add a new student
4. Search a student
   a. Search on given last name
   b. Search on given student ID
5. Delete a student with a given student ID
6. Update a student with a given student ID
7. Sort on last name or average mark
   c. Sort on average mark
   d. Sort on last name
8. Exit
The student information is as follows.

class Student {
    private String ID;
    // student ID
    private String fName;
    // student first name
    private String lName;
    // student last name
Exercises

The student information is as follows.

```java
private int[] cwc = new int[5];
// coursework components
// cwc[0] = assignment 1 (7%),
// cwc[1] = assignment 2 (7%),
// cwc[2] = assignment 3 (6%),
// cwc[3] = CW exam 1 (10%)
// cwc[4] = CW exam 2 (10%)
// marks are given in the range
// [0..100]
```
Exercises

The student information is as follows.

```java
private int cwm; // CW marks fall in [0..40]
// cwm = cwc[0]*7% + cwc[1]*7%
// + cwc[2]*6% + cwc[3] * 10%
// + cwc[4] * 10%
priivate int fem; // final exam
// marks fall in [0..60]
private int fm; // final marks
// = cwm + fem
```
Exercises

The student information is as follows.

// constructors
// methods

}
References


References

http://www.pearsonhighered.com/savitch