

C307 Data Representation

Unit-2, Part A Data Structures

By:



Dr. Hossein Hakimzadeh
Computer Science and Informatics
IU South Bend

Elementary Data Structures

- ☐ Memory
- ☐ Variables
- ☐ Arrays (1D, 2D, 3D, etc.)
- ☐ Classes
- ☐ Linked List
- ☐ Stacks
- ☐ Queues
- ☐ Doubly Linked List
- ☐ Binary Trees
- ☐ Hash Tables

Memory

- ❑ The main memory is an electronic solid-state random access device.



- ❑ Memory can be viewed as a series of sequential bytes, in which data and/or code can be stored.

Variables

- ❑ A variable is the name given to a memory location. (the memory location may hold data items such as numbers, characters, etc.)

- ❑ Example:

- ❑ `int x;`
- ❑ `float y;` `// 32 bit float`
- ❑ `double yy;` `// 64 bit double precision.`
- ❑ `char z;`

- ❑ `x = 5;`
- ❑ `y = 4.8f;` `// the "f" indicates that 4.8 is 32 bit float`
- ❑ `y = 4.8;`
- ❑ `z = 'A';`

Variables

- ❑ A variable is the name given to a memory location. (the memory location may hold data items such as numbers, characters, etc.)
- ❑ Example:
- ❑ `boolean aBoolean = true;`
- ❑ `String FirstName = "Jack"; // static allocation`
- ❑ `String LastName = new String("Jackson"); // dynamic allocation`
- ❑ `byte aByte = 'A'; // Displays 65`
- ❑ `char aChar = 'A'; // displays 'A'`

Other Data Types:

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Arrays

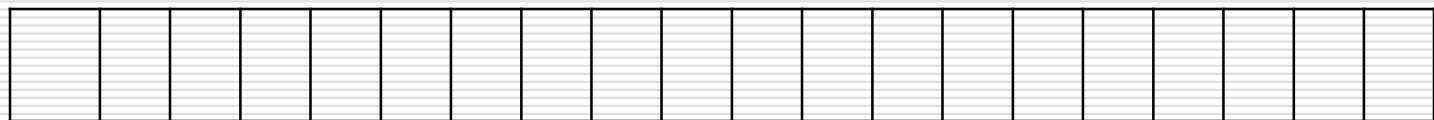
- ❑ A homogeneous aggregate of data elements.
- ❑ A collection of objects of the same type.
- ❑ A collection of similar variables which are identified under the same name.

- ❑ **Creating an Array of 20 integers:**

- `int test_scores = new int[20];`

or

- `int[] test_scores = new int[20];`



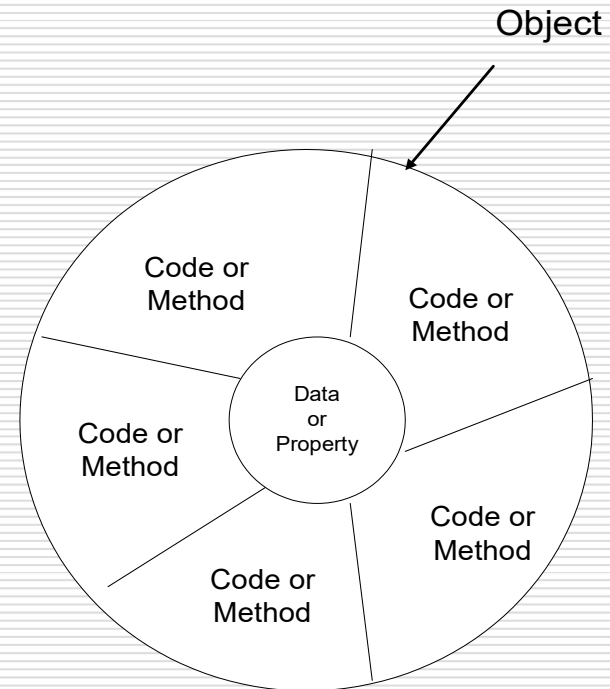
0 1 2 3 4 19

Array as an Object

- ❑ Encapsulating an array as an object allows the programmer to view the data and its operation as one cohesive component.

- ❑ Array Operations or Methods:

- Initialize()
- Insert() or Read()
- Print() or Display()
- Search()
- Sort()
- Mean()
- StdDev()



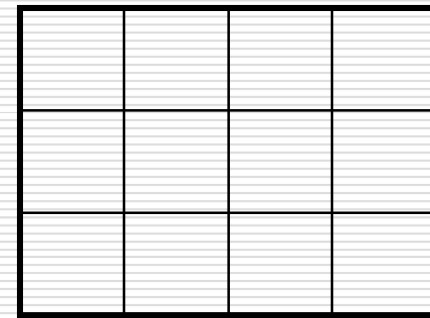
Multi-Dimensional Arrays:

❑ Arrays can have more than one dimension

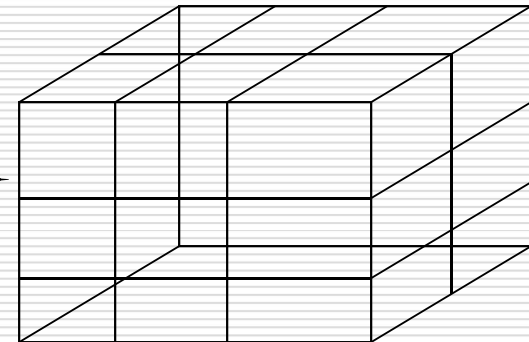
❑ Like a table of values

❑ Or a Cube of values.

2-D



3-D

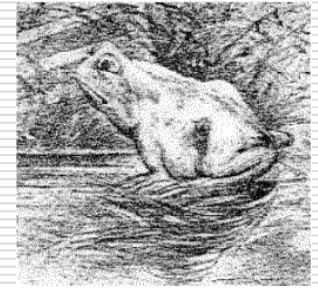


Tic-tac-toe game:

Make a 3x3 storage locations

	0	1	2
0	X	O	O
1	O	X	X
2	X	X	O

Storing images:



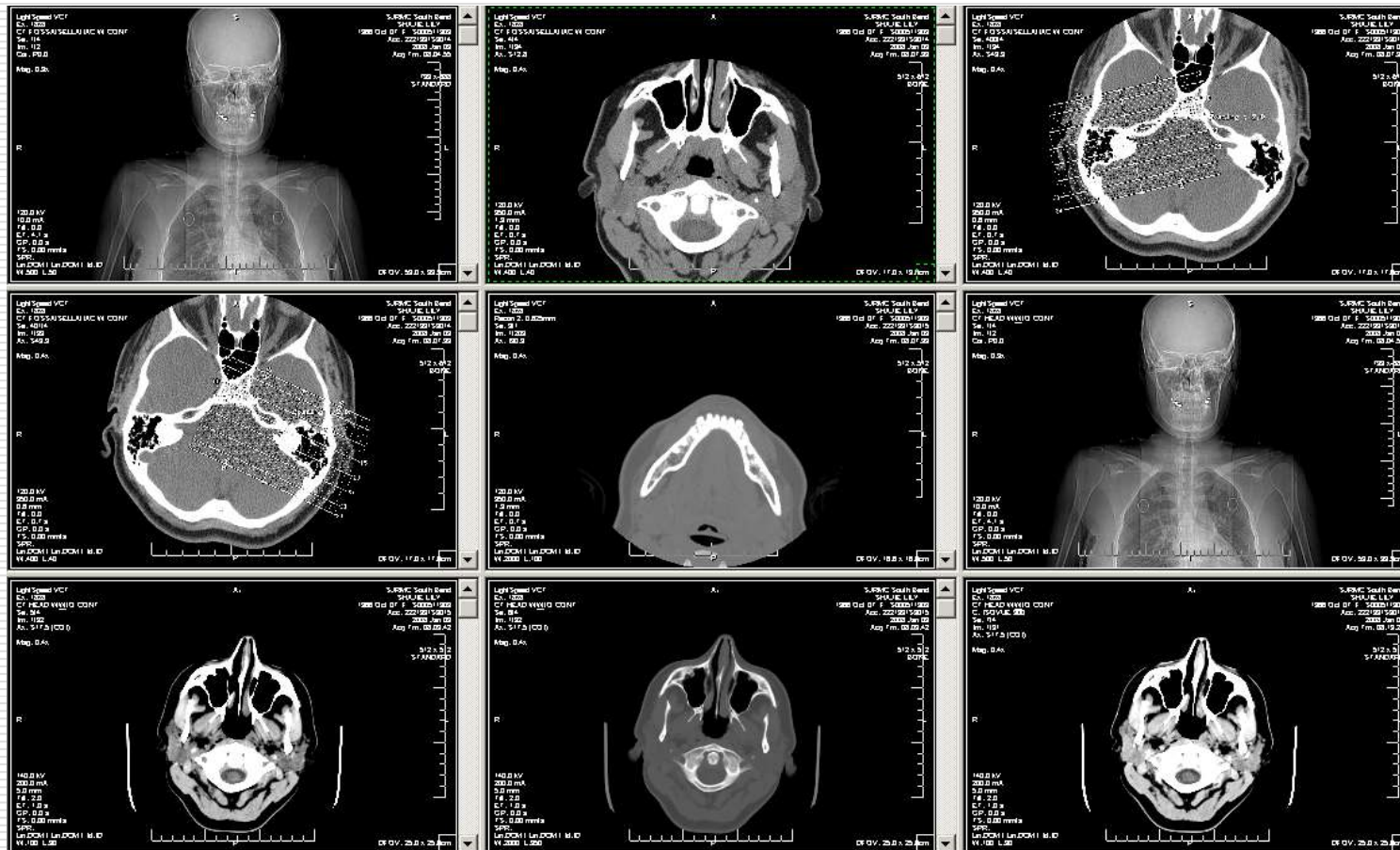
Storing videos:

A form
of 3-D
array



(c) Copyright 2007, H.
Hakimzadeh

Medical imaging:

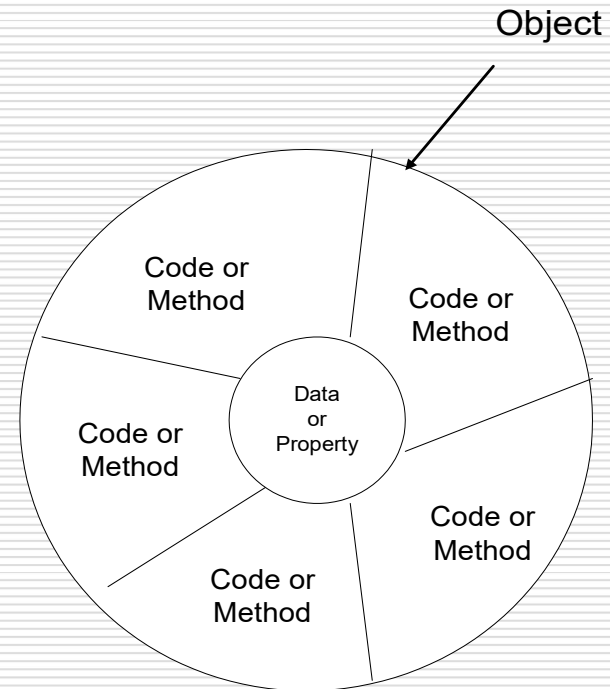


2-D Array as an Object

- ❑ Encapsulating an array as an object allows the programmer to view the data and its operation as one cohesive component.

- ❑ Array Operations or Methods:

- Initialize()
- Insert(x, y)
- Randomize()
- Display()
- Search(data)



Simple Data Structures

- ☐ Memory
- ☐ Variables
- ☐ Arrays (1D, 2D, 3D, etc.)
- ☐ Structs (not part of java, but a pre req for classes)
- ☐ Classes
- ☐ Linked List
- ☐ Stacks
- ☐ Queues
- ☐ Doubly Linked List
- ☐ Binary Trees
- ☐ Hash Tables

Structures

Note: Structures are **not part of the Java Language**, but they are like a java class with only **public variables** in it.

□ A **structure** provides the means to group together related data elements.

```
//-----  
// Defining a new structure called CD Account.  
// A CD account is a structure which holds  
// the following information:  
// Owner, CD balance, Interest Rate,  
// and the term or (maturity period) of the account.  
//
```

□ A **structure** is heterogeneous aggregate of data elements.

```
struct CDAccount {  
    string owner;  
    double balance;  
    double interest_rate;  
    int maturity_term;  
};
```

```
// Creating a structure  
CDAccount    my_account;
```

Simple Data Structures

- ☐ Memory
- ☐ Variables
- ☐ Arrays (1D, 2D, 3D, etc.)
- ☐ **Classes** (extend the notion of structure...)
- ☐ Linked List
- ☐ Stacks
- ☐ Queues
- ☐ Doubly Linked List
- ☐ Binary Trees
- ☐ Hash Tables

Classes

- A **Class** is very similar to a structure, however it also allows the programmer to encapsulate the methods that act upon the data.

Java Class:

```
public class Person {  
  
    private String lastName;  
    private String firstName;  
    private int age;  
  
    //-----  
    public Person(String last, String first, int a) { // constructor  
        lastName = last;  
        firstName = first;  
        age = a;  
    }  
    //-----  
  
    public void displayPerson() {  
        System.out.print("  Last name: " + lastName);  
        System.out.print(", First name: " + firstName);  
        System.out.println(", Age: " + age);  
    }  
    //-----  
  
    public String getLast() // get last name  
    {  
        return lastName;  
    }  
  
} // end class Person
```

C++ Class:

```
class student  
{  
    long ID;  
    char F_name[15+1];  
    char L_name[25+1];  
    char *E_mail;  
    int  Get_next_field(char *student_record_as_string,  
                        int curr_str_location,  
                        char *next_field);  
  
public:  
    student();  
    student(const student& a_student);  
    ~student();  
    void Init();  
    void Print();  
    void Set_student_data();  
    void Set_student_data(long id, char *fn,  
                           char *ln, char *email);  
    void String_to_student(char *student_record_as_string);  
    void Student_to_string(char *student_record_as_string);  
    student& operator= (student &RHS);  
    int operator== (student &RHS);  
    int operator!= (student &RHS);  
};
```