Java Programming (Java Fundamental)

3rd Class Department of Network
College of Information Technology

OUTLINE

- **Beginning in Java**
  - 1.1.Beginning in Java
    - 1.1.1.What is Java
    - 1.1.2.Characteristics of Java
    - 1.1.3. Why we learn Java?
  - 1.2.Components of Java Programming
  - 1.3.Java Variable and Data Type
    - 1.3.1. Variables and key words in Java

- **Components and Control Tools in Java**
  - 2.1. Data input
  - 2.2. Data output
  - 2.3. Condition statements
  - 2.4. Loop statements

OUTLINE

Arrays and Methods in java

- 3.1 Java array
  - 3.1.1.Declaration of an array
- 3.2. Java Methods
  - 3.2.1.Method Definition:
  - 3.2.2.Methods Overloading

OOP and Java Structure

- 4.2. Java Program Structure
  - 4.2.1. What is a Class?
  - 4.2.2. Defining of Classes
- 4.3. Constructors:
- 4.4. Objects

INTRODUCTION IN JAVA

This case study consists of four sections which represent the mainly introduction in java programming.

The first section deals ***Java primitive definition* and *variables***. we mentioned beginning and characteristic of Java . An important ideas such as why we learn Java? Reserved Java keywords and variable definition, are included also in this section.

The second section presents the main ***input/ output operations*** such as *scanner* and ***showinputmessage* …etc.** furthermore, this section included also the definition of the main loops structures such ***for, while (condition) and switch multiple case***. We explained in details the main structure for each loop. This section will enable the user to write down the first simple project in Java.

The third section present the two important thinks. The first one is the **array structure** and how represents it in memory? The second introduces **the introductory of methods**. Also, it mentions *methods definition and methods overlapping*.

INTRODUCTION IN JAVA

Finally, The four section shows more robust programming. It deals with ***object oriented programming (OOP)*** by covering the ***class definition*** in Java**.** What are ***the fields and methods*** in this class**,** what is the constructor***? What is the different between constructor and method.*** The main important think here is the ***objects*** that call the class and how pass the data to this class? All of these thinks are included by many examples in this section. After finishing this section, the user will deal with robust programming in Java such as ***classes, Constructor and methods.***

1.1. Beginning in Java

Sun Microsystem (by James Gosling) began developing Java behind closed door in 1991. It wasn't reveal to the public until 1995. Java is a relatively new and exciting technology.

The original name of Java was intended to be "OAK", but they couldn’t use that name because it was already taken (by Oak Technologies). Other names floating around were "Silk" and "DNA". Apparently, the name "Java" was ultimately picked because it gave the web a "jolt" and Sun wanted to avoid names that sounded nerdy.

1.1.1 What is Java

DEFENTION - ***Java*** is a programming language expressly designed for use in the **distributed** environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an **object-oriented**. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build a small application module or **applet** for use as part of a Web page.

1.1.2.Characteristics of Java

- ***Platform independent***

  Java program do in general not access directly system resources but use the Java ***virtual machine*** as abstraction. This makes Java programs **highly portable**. A Java program which is standard complaint and follows certain rules can run unmodified all several platforms**, e.g. Windows or Linux.**

- ***Object-orientated programming language***

  Except the primitive data types, all elements in Java are objects.

- ***Interpreted and compiled language***

  Java source code is transferred into byte-code which does not depend on the target platform. This byte-code will be interpreted by the Java Virtual machine (JVM).

- ***Automatic memory management***

  Java manages the memory allocation and de-allocation for creating new objects. The program does not have direct access to the memory. The so-called *garbage collector* deletes automatically object to which no active pointer exists.

Note* :** The Java syntax is similar to C++. Java is case ***sensitive, e.g. the variables *myValue* and *myvalue* will be treated as different variables.

1.1.3. Why we learn Java?

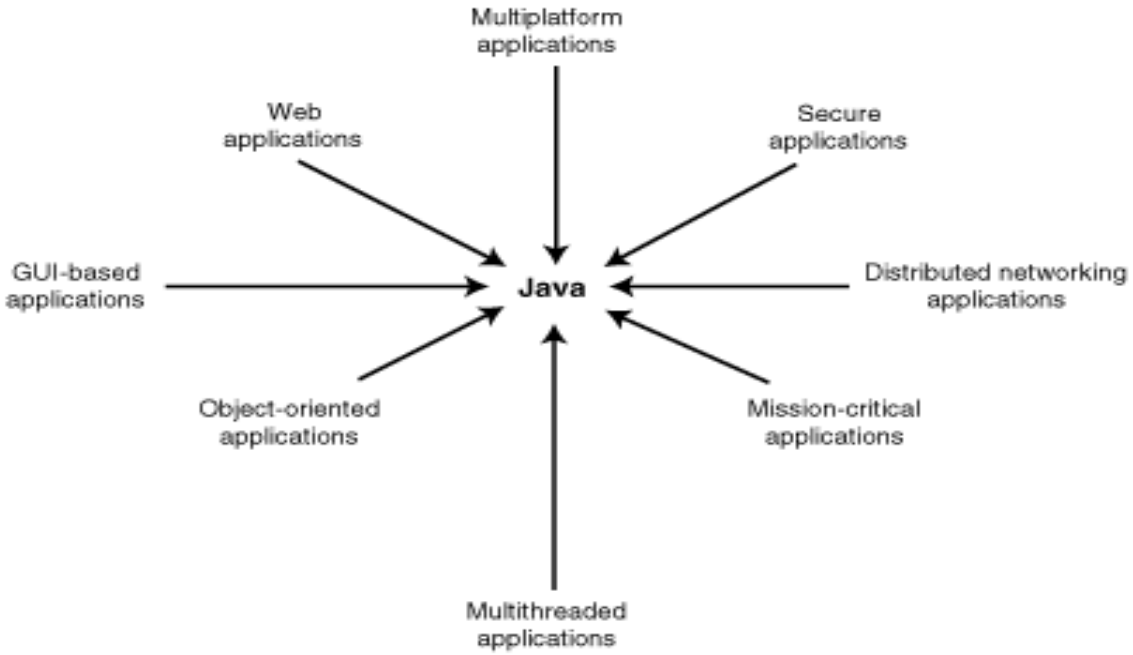

Fig 1.1 Where Java excels

1.2. Components of Java Programming

- Comments.(to illustrate the programs steps and comments)
- Package. (it using to include one or more class in one package)
- Class.(at least one in one project)
- Main Method.(at least one in one project and its use for project execution)
- Other methods. (the main class might be included many methods)
- Blocks.(codes must be organize)
- Reserved Keyword. (such as ***private***, ***String***, ***extends, class***, and so on)
- Instructions.(language tools)
- Access specify. (like ***public***, ***private***, ***protected***)

1.3. Java Variable and Data Type

The Java programming language is a ***strongly typed*** language, which means that every variable and expression has a type that is known at compile time. ***In the other word*** Variables in Java are ***strongly typed***, which means that the compiler checks the type of a variable with the data associated with that variable. For example, if you declare a variable to hold a **String**, you cannot assign an **integer** value to that variable.

There are three basic kinds of scope for variables in Java:

- ***Local variable.*** declared within a method in a class, valid for (and occupying storage only for) the time that method is executing.
- ***Instance variable.*** declared within a class but outside any method. It is valid for and occupies storage for as long as the corresponding object is in memory; a program can instantiate multiple objects of the class, and each one gets its own copy of all instance variables.
- ***Static variable.*** declared within a class as *static*, outside any method. There is only one copy of such a variable no matter how many objects are instantiated from that class.

1.3.1 Variables and key words in Java

The name of a variable can be made of one letter (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z,A ,B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, or Z) .

- The name of a variable can start with a letter, an underscore "_", or the dollar sign $.
- After the first character, the name of the variable can include letters, digits (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9), or underscores in any combination
- The name of a variable cannot be one of the words that the Java languages has reserved for its own use. See Table in next slide.

The next slide show us the main key words in Java programming language.

TABLE 1.1. : Main Keyword in Java Programming

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	Import
intanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

INTRODUCTION IN JAVA (BEGINNING IN JAVA)

A data type in Java can store into two kinds:

- **A reference to a Java object** (using ***new*** keyword when assign it in memory)
- **Java primitive type values**
  - **integers** (whole numbers, declared as **byte**, **short**, **int**, and **long**) ex) **int** a = 77;
  - **floating-point** (decimal numbers, declared as **float** and **double**) . ex) **float** a = 2.011;
  - **characters** (declared as **char**, representing one character like 'A' or ','). ex) **char** a = 'a';
  - **boolean** (holding only ***true*** or ***false*** as values).

    ex) **boolean** a = **true**;

See table in the next slide

Table 1.2. Instance variables access specification

Visibility	Default (friendly)	public	protected	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	No	No
From any class outside the package	No	Yes	No	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	No	Yes	Yes	No

Note: this table is same when instance methods and inner class access specify.

2.1. Data Input

2.1.1.Data Input by Console Scanner

One of the classes you can use is called ***Scanner***. Before using the Scanner class, you must import the **java.util.Scanner** package into your program. as in below

```
Scanner scnr = new Scanner(System.in);
```

To implement that , see the whole example below:

```
// This Example enters the data from console by Scanner class:
import  java.util.Scanner
public class Read_Scanner {
 public static void main( String [ ] args ) {
Scanner in = new Scanner( System.in );
 System.out.print( in.nextline() );
 }// end of main method
}// end of main class
```

2.1.2 Data Input by Dialog Box

you must import the **javax.swing.JOptionPane** package into your program.

String *variable_Name*
= JOptionPane.showInputDialog(null, "X", "Y", JOptionPane.Formal);

null : used for the opening dialog box in the middle of execution window.
X : used for the box title Such as " Input your value ".
Y : used for the dialog title. Such as "Dialog Input".
Formal : used for information message in picture box.

```
// This example enters the data from dialog box :
import javax.swing.JOptionPane
public class Read_Dialog {
 public static void main( String [ ] args ) {
 String st = JOptionPane.showInputdialog(null,"input dailog", "show
    this", JOptionPane.QUESTION_MESSAGE);
    System.out.print( st );   // print the entered data
 }// end of main method
}// end of class
```

The output of example above as in figure (2.1) below:-

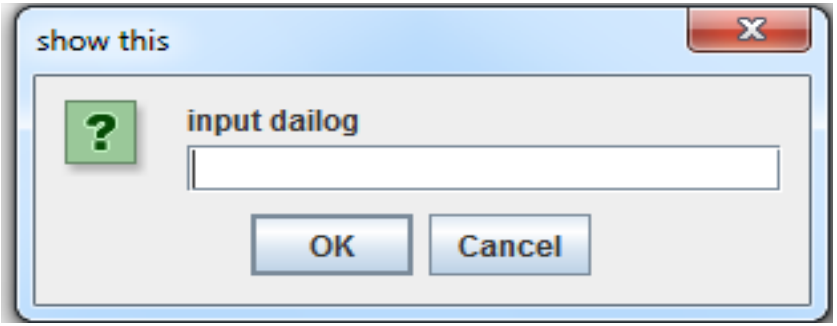

Fig.2.1 The output of Example (2)

2.2. Data output

Data output in java programming implements in several way

2.2.1. System.out.println()

System: It is a standard java class. It comes from the java.lang package. By default, It is a package.

out : The class system contains static field named "out". So, System.out refers to the value stored in that static filed. The value System.out is an object of the class *printStream* from the standard java package *java.io.*

Println() : It is the instance method come from the *PrintStream* class.

Example:

```
System.out.println(" Hello My class ");
```

2.2.2. Data Output by Dialog Box

you must import the **javax.swing.JOptionPane** package into your program. So, when you need to print, you can use this instruction.

```
JOption.showMessageDialog("+(null, "X", "Y",JOptionPane.Formal);
```

- null: used for the position.
- X : used for the output text for parameter output (such as " The sum is "+sum).
- Y : used for the dialog title such as "Dialog Output".
- Formal: used for information message in picture box.

2.3. Condition Statements

1. The ***if/else*** selection structure (single-selection structure)
2. ***Conditional operator*** ?: (ternary conditional)
3. The ***switch*** selection structure (multiple-selection structure)

2.3.1. if/else structure

The simple structure for ***if/else*** statement as figure (2.2) below:-

```
Syntax: if (condition(s)) {  statement or block of statements }
        Or : if (condition(s)) {  statement or block of statements }
else {  statement or block of statements }
```

Fig. 2.2 Syntax of *if/else* statement

The condition is a Boolean comparison that can use one of the binary operators:

== , != , > , < , >= , <=

In addition to the Boolean operators && (AND), || (OR), and ! (NOT). Of course combinations can occur using brackets ( )

Example:

```
if (studentGrade >= 60)
   System.out.println (“Passed”);
else
   System.out.println (“Failed”);
```

Example:

```
if (totalSum<50){
   // no discount
   System.out.println (“No discount”);
   System.out.println (“Payment is ”+totalSum);
   }
else {
   //10% discount
   totalSum=totalSum-(totalSum*0.1);
   System.out.println (“Payment with discount is ”+totalSum);
   }
```

Example:

```
 if (x !== y);
   System.out.println (“Not Equal”);
```

Example:

```
if  (x>5) {
    if (y>5)
        System.out.println(“x and y are > 5”);
}
else
    System.out.println(“x is <=5”);
}
```

2.3.2 Conditional operator

Syntax: (*condition* (s) **?** *statement1* **:** *statement2*)

Where : **?** means **then** and **:** means **else**.

Example:

System.**out**.println (studentGrade >= 60 ? “Passed” : “Failed”);

2.3.3. switch selection structure

Java provides the ***switch*** multiple-selection structure to handle decision making for the values of types byte, short, int, long, char, Boolean and string . figure (2.4) show this structure for switch selection.

Syntax: switch (variable)

{ case value_1 : { statement or block of statements; break; }
case value_2 : { statement or block of statements; break; }
case value_3 : { statement or block of statements; break; }
.....
.....
case value_n : { statement or block of statements; break; }
default : { statement or block of statements; } }

Example

```
String st = JOptionPane.showInputDialog("Enter an integer: ");
int i=integer.parseInt(st);
switch (i) {
        case 1: { System.out.println ("Read"); break;}
        case 2: {System.out.println ("Blue"); break; }
        case 3: { System.out.println("Black"); }
        default:        System.out.println("Unknown");
        }      //end of switch
```

2.4. Loop Statements

1. **for** loops
2. **while** loops
3. **do/while** loops

2.4.1. for structure

The ***for*** Repetition Structure is similar to the C++ style

```
Syntax:  for  ( initializations; condition(s); increment/ decrement ) {
                                statement or block of statements ;
                            }
```

Fig.2.5 Syntax *For* loops structure

Example

```
for (int i=0 ; i<10 ; i++) {
   System.out.println ("the value I is" + i);
  }
```

2.4.2. while structure

The ***while*** Repetition Structure as shown in figure (2.6)

```
Syntax:   while ( condition(s) ) {
             statement or block of statements ;
             }
```

Fig2.6. Syntax *While* loop structure

Example:

```
int numberCount = 1;          //   this step very important
while (numberCount<=10) {
   System.out.println (numberCount);
   numberCount++;
 }
```

2.4.3. do / while structure

The ***do/while*** Repetition Structure illustrate in figure (2.7) as below;

```
Syntax:   do {
                statement or block of statements ;
                } while ( condition(s) ) ;
```

Fig. 2.7 Syntax *Do/ While* loop structure

Example:

```
int numberCount = 10;
do {
    System.out.println (numberCount);
    numberCount--;
} while (numberCount>0);
```

3.1. Java Arrays

Array is the most important thing in any programming language. By definition, array is the static memory allocation. It allocates the memory for the same data type in sequence.

In **Java array is a container object that holds a fixed number of values of a single type**. see figure (3.1).

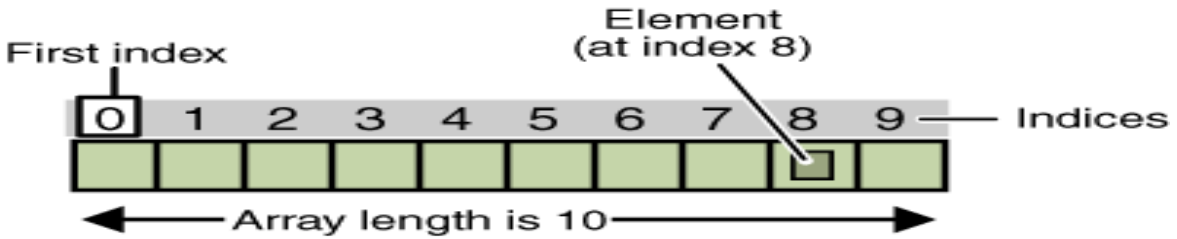

Fig. 3.1 Array definition

3.2.1. Declaration of an array

We can declaration such as :

```
int [ ] num;
num  =  new int [6];
```

or we can also :

```
int [ ]num = new int [6];
```

and also we can write

```
int num[ ] = new int [6];
```

Some times user declares an array and it's size simultaneously. You may or may not be define the size in the declaration time. such as:

```
int num[]  =  {50,20,45,82,25,63};
```

When we declare the definition of array like that **int** [ ] arraytest; this will be in memory like below :-

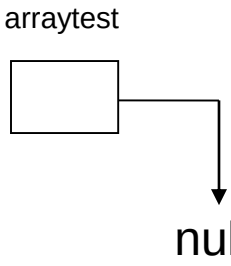

Fig 3.2 Declaration array in java

After that when implement this, arraytest = **new int** [6]; that's mean

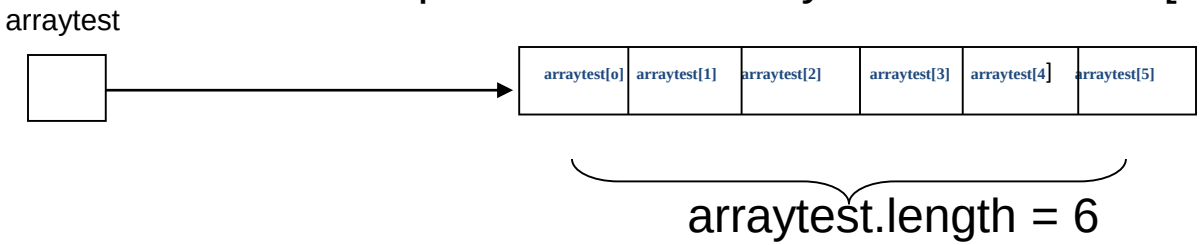

Fig 3.3 Specification array in java

```
package TextArray1
import javax.swing.JOptionPane;
public class TestArray {
public static void main(String[ ] args) {
final int num=6;                          // final means const in another language
int[ ] numbers=new int [num];
for(int i=0;i<numbers.length;i++) {
String numstring=JOptionPane.showInputDialog(null,"enter a number: ","example array",
                                                        JOptionPane.QUESTION_MESSAGE);
   numbers[i]=Integer.parseInt(numstring); // convert an input string to integer
 }
        int max=numbers[0];
                        for (int i=0;i<numbers.length-1;i++) {
                            if (max<numbers[i])
                                max=numbers[i];
                        }
        int count=0;
                        for (int i=0;i<numbers.length-1;i++) {
                             if (numbers[i]=max)
                            count++;
                         }
      for(int i=0;i<numbers.length-1;i++)
      String output+ = numbers[i] + " ";
System.out.println("\n the largest number is " +max);
System.out.println("the occurrence count of the largest number is " +count);
  }
}
```

3.2. Java Methods

In programming, experience has shown that the best way to develop and maintain a large program is to construct it from small, simple pieces or modules. This technique is called *divide* and *conquer*. Modules in Java are called *methods* and *classes*. Many methods and classes are already available “prepackaged” in the Java API (Java Class Library).

One of the important classes in Java is the Math class that has many methods allowing the programmer to perform certain common mathematical calculations. The Math class is part of the **java.lang** package, which is automatically imported by the compiler.

***Example*:**

```
System.out.println (Math.sqrt(900.0));
System.out.println (Math.sin (0.0));
```

3.2.1.Method Definition:

Any method in java contain two part: method header and method body as in figure (3.4)

```
[< method_specify>] <return_type> <method_name> ([< parameters >])
 {
 // method body (declarations and statements)
    }
```

Fig 3.4 Method definition

***Note* :** the bracket [ ], means inside it optional.

Example:

```
public double maximum (double x, double y, double z) {
    return Math.max ( x, Math.max ( y, z));
 }
```

Important notes:

- If the return type is void then the return statements inside the method should return nothing.
- Don’t put a semicolon after the first line of the method definition.
- Each variable in the parameter list should have a type.
- A method cannot be defined inside another method.

3.2.2.Methods Overloading

In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be overloaded, and the process is referred to as method overloading. Method overloading is one of the ways that Java implements polymorphism.

When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call. Thus, overloaded methods must differ in the type and/or number of their parameters. While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method. When Java encounters a call to an overloaded method, it simply executes the version of the method whose **parameters match the arguments** used in the call. Here is a simple example that illustrates method overloading:

Example:

```
public int sum(int  firstnumber , int seconnumber) {
    return firstnumber + secondnumber;
}
public float sum(int  firstnumber , float seconnumber ) {
    return firstnumber + secondnumber;
}
protected float sum(float  firstnumber , float  seconnumber) {
    return firstnumber + secondnumber;
}
public int sum(int  firstnumber , int seconnumber , int thirdnumber) {
    return thirdnumber+sum(firstnumber , secondnumber);
}
```

Now we need to call *System.out.print("the sum is " + sum(3.5,6.9));*

What will happen (i.e. which method will be called here?)

4.1. Object Oriented Programming with Java (OOP)

Java is a object oriented programming and to understand the functionality of OOP in Java, we first need to understand several fundamentals related to objects. These include ***class, method, inheritance, encapsulation, abstraction, polymorphism .***

Class - It is the central point of OOP and that contains data and codes with behavior. In Java everything happens within class and it describes a set of objects with common behavior.

Object - Objects are the basic unit of object orientation with behavior, identity. As we mentioned above, these are part of a class but are not the same. An object is expressed by the variable and methods within the objects.

Methods - We know that a class can define both attributes and behaviors. Again attributes are defined by variables and behaviors are represented by methods.

Inheritance - This is the mechanism of organizing and structuring software program. Though objects are distinguished from each other by some additional features but there are objects that share certain things common.

This saves work as the special class inherits all the properties of the old general class and as a programmer you only require the new features. ***This helps in a better data analysis, accurate coding and reduces development time.***

Abstraction - The process of abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface).

Encapsulation - This is an important programming concept that assists in separating an object's state from its behavior. This helps in hiding an object's data describing its state from any further modification by external component.

Polymorphism - It describes the ability of the object in belonging to different types with specific behavior of each type. So by using this, one object can be treated like another and in this way it can create and define multiple level of interface.

4.2. Java Program Structure

Let's summarize the general nature of how a Java program is structured

- A Java program always consists of one or more classes.
- You typically put the program code for each class in a separate file, and you must give each file the same name as that of the class that is defined within it.
- A Java source file must also have the extension .java.

4.2.1. What is a Class?

A class is a prescription for a particular kind of object - it defines a new type. We can use the class definition to create objects of that class type, that is, to create objects that incorporate all the components specified as belonging to that class.

There are just two kinds of things that you can include in a class definition:

- **Fields**

These are variables that store data items that typically differentiate one object of the class from another. They are also referred to as ***data members*** of a class.

- **Methods**

These define the operations you can perform for the class - so they determine what you can do to, or with, objects of the class.

4.2.2. Defining of Classes

To define a class you use the keyword class followed by the name of the class, followed by a pair of braces enclosing the details of the definition. Let's consider a concrete example to see how this works in practice.Class declares by using the following formula in figure (4.1) :

```
class_specify class class_Name {
   class_methods and instance_variable_definition
}
```

Fig 4.1 Class definition

Example:

```
class circle {
      double redius=1.0;
       circle ( ) {   }
      circle (double newradius) {
          radius=newradius;
          }
        public double findarea( ) {
             return radius*radius*3.14;
          }
  }
```

Notes:-

1- If we don't put any access control before the class word, then by default that class is friendly. (like the example above)

2- In the same application cannot be more than one class are public

Example:

```
package Test
public class Main_Class {
public static void main( String []args){
……..
……
}
}
public class AnotherClass {
……
}
```

That's wrong the another class must be default (default means don't put anything before **class** word)

3- Inner classes also the same of (instance method or variable access control).

4- There is a special method in any class called constructor.

4.3. Constructors:

When you create a new instance (a new object) of a class using the new keyword, a constructor for that class is called. Constructors are used to initialize the instance variables (fields) of an object. Constructors are similar to methods, but with some important differences.

- ***Constructor name*** is class name. A constructors must have the same name as the class's name.
- ***Default constructor***. If you don't define a constructor for a class, a default parameter less constructor is automatically created by the compiler. The default constructor calls the default parent constructor (super()) and initializes all instance variables to default value (zero for numeric types, null for object references, and false for booleans).
- ***Default constructor*** is created only if there are no constructors. If you define any constructor for your class, no default constructor is automatically created.

Differences between methods and constructors.

- There is no return type given in a constructor signature (header). The value is this object itself so there is no need to indicate a return value.

- There is *no return statement* in the body of the constructor.
- The *first line* of a constructor must either be a call on another constructor in the same class (using this), or a call on the super class constructor (using super). If the first line is either of these two ones, the compiler automatically inserts a call to the parameter super class constructor.

These differences in syntax between a constructor and method are sometimes hard to see when looking at the source. It would have been better to have had a keyword to clearly mark constructors as some languages do.

- ***this(...) - Calls another constructor in same class.*** Often a constructor with few parameters will call a constructor with more parameters, giving default values for the missing parameters. Use *this to call other constructors* in the same class.
- ***super(...).*** Use *super* to call a constructor in a parent class. Calling the constructor for the superclass must be the *first statement* in the body of a constructor. If you are satisfied with the default constructor in the superclass, there is no need to make a call to it because it will be supplied automatically.

Example:

```
public class Test {
  private  int  a,b;
Test ( ) {          a = 9;        // default constructor
                     b = 6;       }
Test ( int k ) {                  //  constructor have one parameter
                this ( k ,1);   // call another constructor in the same class
                                  }
Test ( int k, int y ) {         // constructor have tow parameter
                              a =  k;
                              b =  y;     }
public int sum( ) {
  return a + b;         }
}
```

What will be happened if we call these object in below?

```
Test  testObject1 = new Test ;
Test  testObject2 = new Test ( );
Test  testObject3 = new Test (8);
Test  testObject4 = new Test (2,3);
```

4.4. Objects

An instance is an executable copy of a class. Another name for instance is object. There can be any number of objects of a given class in memory at any one time. All objects in java programs are created on heap memory. An object is created based on its class. You can consider a class as a blueprint, template, or a description how to create an object. When an object is created, memory is allocated to hold the object properties. An object reference pointing to that memory location is also created. To use the object in the future, that object reference has to be stored as a local variable or as an object member variable.

```
//Create an 'MyObject' for the first time the application started
{
   MyObject obj = new MyObject();
}
```

Fig 4.2 Object formula

Now, ***how can we access to the data and methods object?***
We can access to the data object and call methods through the following formula: -

Call data object: ObjectRefVal . Data;

Call the methods object: ObjectRefVal . MethodName(parmaters);

Note : the access control from the data and methods here must be public or by default.

Now consider the following example that shows you these issues

```
// First class contain main method
package Test_Class;
import javax.swing.JOptionPane;

public class TestCircle {
public static void main(String[ ] args) {
        Circle myCircle = new Circle(5.0);
        System.out.println(" the area of the circle of radius " +  myCircle.getRadius( ) + "is "
                                + myCircle.findArea( ));
         myCircle.setRadius(myCircle.getRadius( )*1.1);
         System.out.println(" the area of the circle of radius " + myCircle.getRadius( ) + "is "
                                 +  myCircle.findArea( ));   }
 }    //end of class1
//  The circle class with accessor  methods
package Test_Class;
public class Circle {
private double radius;
public Circle( ) {    radius = 1.0;     }  // default constructor
public Circle(double newRadius) {   radius=newRadius;  }     // copy constructor
public double getRadius( ) {       // we need this method because Radius is private
                    return radius;     }
public void setRadius(double newRadius) {      // update Radius
    radius = (newRadius >= 0 ? newRadius : 0);
}
public double findArea( ) {
    return radius * radius * 3.14;   }
 }     //end of class2
```