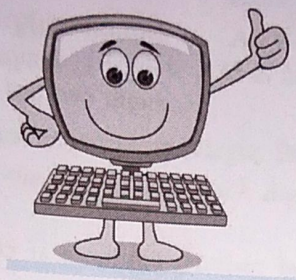


CHAPTER 2

CLASS AS THE BASIS OF ALL COMPUTATION

CHAPTER AT A GLANCE :

- Introduction
- Data (or Member data or Instance variables or Attributes)
- Methods (or Functions or Behaviour)
- Need of Class
- Primitive Data Types within the Class
- Composite Data Type
- Data Members within the Class
  - Static or Class Variable
  - Instance Variable
- Methods within the Class
  - Static or Class Method
  - Instance Method
- Scope of variable
- Objects as Instance of Class
- Class as an Object Factory
- Points to Remember

2.1 Introduction

From the revision tour you have understood that many objects can be created from a defined class which encapsulates or wraps a group of attributes (means data members or variables) and a number of methods or functions (which contains the required operation on data members). These methods actually control the behaviour of the objects being created. Many classes can create many objects, which can interact amongst themselves by passing messages depending upon the computational requirements. The data members (or variables), defined within a Class, are also called instant variables. The methods (or functions) contain set of instructions to manipulate those variables. Data and methods, present in a class, are collectively called members of that class.

The Java allows creation of class/classes within the project and to execute the member of class an object of that class must be created within the main( ) method. However, more than one classes can also be defined within one project. If a project consists of many classes, then in this case, only one class should contain the main( ) method. Of course, a project using a single class should contain the main( ) method within that class only. The class containing the main( ) method is supposed to control the entire computational activities with help of objects. You must remember that to avoid errors in the program the data members (or variables) must be declared with valid data types and methods (or functions) must contain valid set of executable statements. Although these concepts have already been discussed in the revision tour but let us summaries these concepts once again in this chapter.

2.2 Class

Definition : The *user defined data type* which contains *data* and its associated *functions* encapsulated or wrapped together is known as a class. In other words, **a class is a way of binding** or **combining data** and **its associated functions in a single unit**.

The class is also known as blueprint or prototype from which objects are created.

The internal data of a class is known as **member data** (or **instance variable**) and the procedures are known as member **methods** (or **functions**).

The member function(s) are used for manipulating data members of class. The class also acts as an independent program that logically organizes data to generate the required output.

2.3 Data (or Member Data or Instance Variables or Attributes)

Definition : The variables declared within the class on which some operations to be performed are known as **data** or **data members** or **instance variables** or **attributes** or **fields**.

The **data member** or **variables** must be declared along with its **data type**. Remember that the **data member** or **variable** or **attribute** define **state**.

Example :

```
int a, b;      float x, y;      // see that a, b are integer and x, y are floating data members
char ch;       long m;          // here ch is character and m is long type data members
```

In the above example **int**, **float**, **char** and **long** are primitive or fundamental data types.

2.4 Method (or Function or Behaviour)

Definition : The user defined compound block which contains set of instructions perform a task is known as **method** or **function**. The task may be addition, multiplication, division *etc.*

Remember that the **method** or **function** define **behaviour of class**.

Example :

```
void Display( )      // starting of the function, here 'void' is data type and 'Display( )' is the
{                    // name of the function
    a++;             // these are executable set of instructions
    b = a * 10;
    System.out.println("The results are = " + a + "  " + b );
}//end of the function Display( )
```

In the above example ***void*** is function's data type and ***Display( )*** is name of the function or method. This method or function **void Display( )** defines **behaviour** means some working on variables. See that inside the function some operation on data members or variables or attributes (**a** and **b**) are performed.

2.5 Need of the class

As Java is pure *Object Oriented Language* so all the required operations should be managed through the class itself. None of the operation are possible without the class. ***Thus it can be said that the class is needed for all kinds of operations or tasks***. The class (or members of the class) can handle so many operations such as Data Hiding, Abstraction, Polymorphism, Encapsulation, Inheritance *etc. i.e.* OOP's concepts.

Now one of the most important things is to **access/execute** members of the class. This is done by the variable that belongs to the *class* which is known as "**Object of the class** or **Instance of the class**" *i.e.* the objects are **instances** of the class.

To control the access of class members an object must be created. Thus it can be said that the **classes as the basis of all computation**.

The class is declared using the syntax or format given below :

Syntax :

```
[<access specifire>]  class      class_name
{
      data type      instance variable 1;
      data type      instance variable 2;
      data type      instance variable 3;
      data type      instance variable 3;
      - - - -
[<access specifire>]  data type   method/function name1 (argument or parameter list, if any)
      {
                  required statements and process within the function;
      }//end of the function 1
[<access specifire>]  data type   method/function name2 (argument or parameter list, if any)
      {
                  required statements and process within the function;
      }//end of the function 2

[<access specifire>]  data type   method/function name2 (argument or parameter list, if any)
      {
                  required statements and process within the function;
      }//end of the function 3
      - - - -
}// end of the class
```

POINTS TO REMEMBER

- Square brackets '[ ]' means optional value and the symbols '< >' means value within this is given by the user. Thus anything given within the square brackets '[ ]' is optional.
- The term **class** is a keyword and should be used as it is *i.e.* all **lower case letters**. The term '*class_name*' is user defined word / identifier / variable that should **not** be a **keyword**. A class represents a set of objects that share common characteristics and behaviour.
- Curly braces '{ }' shows beginning and end of the class. These curly braces '{ }' are also used to declare compound block as well as method(s).
- The variables given within the class to perform some tasks are known as **data members** or **instance variables**.

- Those specifiers that controls the execution of members of a class are known as **Access specifires / accessibility modes.** In Java following access specifiers are used:
  - (i) **public :** It is a keyword also known as *Access Specifier*. This mode refers that *public members are accessible* by the object of class, method(s) of class as well as by any subclass.
  - (ii) **private :** It is a keyword also known as *Access Specifier*. This mode refers that *private members are accessible only by the method(s) within a class* and **not** by any subclass or outside the class or its scope.
  - (iii) **protected :** It is a keyword also known as *Access Specifier*. This mode refers that *protected members are accessible within the class* member(s) and as well as other class(es) by the next inherited class.
- In the absence of **access specifiers**, all the declaration within the class are **default** type. Here **default** is "**not**" a keyword. In this case the declared members are accessible within the class.
- Those declarations which are done under **private** are **not** accessible to any outside function. So "*The activity which restrict the execution of its members outside its scope is known as* ***Data Hiding/Information Hiding***".
- Those declarations which are done under **public** are accessible to any function outside the class. "*The activity of extracting essential functions of an object in the form of public interface is known as* ***Abstraction***".

Example 1 (Using only One Class) : The following is declaration of a class with its name "**sample**"

```
public   class  sample
{
 // the following are the data members or  instance variables of the class
        int x, y;
        float m;
        char ch;
public   void  show( )        // here show( ) is  method or function  name
{
    y = x + 25;          // operation on data or variable
    m = 75;              // operation on data or variable
    ch++;                // operation on data or variable
}// end of the function show( )
}// end of the class sample
```

These variables of the class defines STATE

This method void show( ) define BEHAVIOUR means operation on data members or variables

In the above example variables **x**, **y**, **m** and **ch** are **data members** or **variables** or **attributes** of the class define **state** and method **void show( )** defines **behaviour** of data members (**x**, **y**, **m** and **ch**), means method(s) or function(s) of the class define **behaviour**. Thus, **a class contains state and behaviour**.

Example 1 (Using Two Classes) : See the declaration two classes "**sample**" and "**example**"

```
public   class  sample                  // beginning of class number 1
{
 // the following are the data members or  instance variables of the class
        int x, y;
        float m;
        char ch;
public   void  show( )        // here show( ) is  method or function  name  with data type 'void'
{
    y = x + 25;          // operation on data or variable
    m = 75;              // operation on data or variable
    ch++;                // operation on data or variable
}// end of the function show( )
}// end of the class sample
```

```
public   class   example                    // beginning of class number 2
{
 // the following are the data members or  instance variables of the class
          int m;
          double q;
public    double   Print( )      // here Print( ) is  method or function  name with data type 'double'
{
     m = 299;              // operation on data or variable
     q = m * 200;          // operation on data or variable
     return q;             // operation on data or variable
 }// end of the function Print( )
 }// end of the class example
```

In the above example variables **x**, **y**, **m** and **ch** are **data members** or **variables** or **attributes** of the class define **state** and method **void show( )** defines **behaviour** of data members (**x**, **y**, **m** and **ch**), and belongs to class sample, whereas, variables **m** and **q** are **data members** or **variables** or **attributes** of the class also define **state** and method **void Print( )** also defines **behaviour** of data members (**m** and **q**), and belongs to class example.

2.6 Primitive Data Types within the Class

A class can contain many type of data or variables as its members. The variables declared using *short*, *int*, *long*, *float*, *double*, *char*, *byte* and *boolean* are called *primitive data types* members of class.

Consider the following example with primitive data type and variables inside the class:

Example : Declaration of a class with primitive data types and variables:

```
public   class   sample
{
 // the following variables are members of class
      int x, y;
      float m;   boolean  flg;
      char  ch;
}
```

These are primitive type members within the class. Here int, float, boolean and char are primitive data types and x, y, flg and ch are variables

2.7 Composite Data Types

The block which contains primitive data types as its members is known as primitive data type. Thus, A class is known as composite data type because a class contains many primitive data types as its members.

Consider the following example with primitive data type and variables inside the class:

Example : Consider the following class declaration:

```
public   class   sample
{
 // data members of class
          int x, y;
          float m;   boolean  flg;
          char  ch;

}
```

Here class sample is a composite data type because this class contains primitive data types int, float, boolean and char type variables

2.8 Types of Data Member (or Variable or Attribute) within the class

Definition : The **variables** declared within a class along with the **data type** (like *int*, *float*, *char* etc.) on which some operations to be performed are known as *data members* or *variables* or *attributes of the class*.

The variable is commonly divided into following categories :

1. **Static Variable or Class Variable**
2. **Instance Variable**

2.8.1 Static Variable (or Class Variable)

Definition : The *data member or variable declared* ***once*** *for a class is known as static / class variable.* All the objects of that class shares these variables because only a single copy of these variable(s) is available in the memory. The keyword **static** is used to make a variable as **class variable** or **static variable**.

Examples :
```
static int x, y;        // variables 'x' and 'y' are integer type class or static variables
static double mark;     // variable 'mark' is double type class or static variable
```

2.8.2 Instance Variable

Definition : The d*ata member or variable declared within the class is known as instance variable.* These variables are available for every object of the class. It means as many objects of a class are available as many copies of instance variables will be available in the memory.

Examples :
```
int m, n;     // variables 'm' and 'n' are integer type instance variables
double x;     // variable 'x' is double type instance variable
```

The following example of class describes the above two types of variables :

```
class  sample
{
int x;                  ——→ This is instance variable because declared without 'static' keyword
static char ch;         ——→ This is class or static variable, see the 'static' keyword .
double mark;            ——→ This is another instance variable because declared without 'static' keyword
static double roll;     ——→ This is another class or static variable see the 'static' keyword
}// end of the class
```

2.9 Method (or Function) within the class

Definition : The method / function / procedure / subroutine defined within the class to execute data members of class (*instance variables* or *static variables*) is known as function or method within the class.

The method (or function) is divided in following two categories :

1. **Static Method or Static Function**
2. **Instance Method or Instance Function**

2.9.1 Static Method (or Static Function)

Definition : The method (or function) which begins with **static** keyword is known as static method. These functions uses only **static** variable(s).

Example :
```
public static void show( )   // see the static keyword makes it static method
          OR
static void show( )          // use of public is optional
```

2.9.2 Instance Method (or Instance Function)

Definition : The method or function which uses both static (or class) variable(s) and instance variable(s) is known as instance method.

Example :
```
public void show( )     // this is instance function
       OR
void show( )            // use of public is optional
```

Example of class with above two types of variables and functions :

```
class   sample
{
static double m;  ------>   This is static data member or variable of the class
int x;                      // this is instance variable
public static void show( )  ------>   This is static or class method or function
{
x = 76;        // ERROR- instance variables can not be used within static methods
m = 3.25;                   // VALID- because 'm' is a class or static variable
System.out.println(x + "\t" + m);
}// end of the function show( )
public void display( )  ------>   This is instance method or function
{
x = 81;        // VALID- instance variables can be used within instance methods
m = 46·38;     // VALID- static variables can also be used within instance methods
System.out.println(x + "\t" + m);
}// end of the function display( )
}// the class closes here
```

2.10 Scope of Variable (or Scope of Identifier)

Definition : The access range of variable(s) within the class is known as scope of variable.
The scope means the area within the ***open and close curly braces*** '{ }'. It covers the following rules:

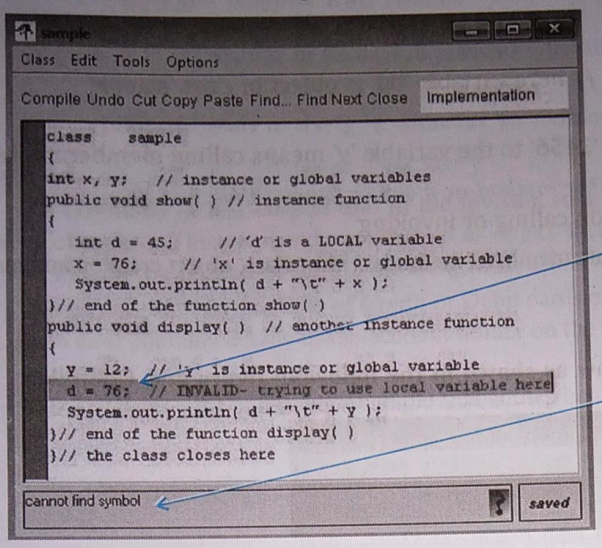
1. The ***instance variable****(s)* is accessible within the class *i.e.* inside all ***instance*** functions (or methods).
2. The *static* (*or class*) *variable(s)* is accessible in all the functions (or methods) within the class. The functions may be **instance** or **static**.
3. The variable declared within a function is accessible within that function only *i.e.* within the *open* and *close curly braces* of that function. *Such type of variable is known as* ***local variable***.

Example 1 : The following class shows the concept of scope of variables :

```
class   sample
{
        int x, y;          // these are instance variables (also known as Global variables)
public void show( )                // this is an instance function
{
int d = 45;     // variable 'd' is declared and initialized within function show( )
                // So its working scope will be within THIS function only and
        // this variable 'd' is also known as LOCAL variable
x = 76; // VALID- instance or global variables can be used within any instance methods
}// end of the function show( )
public void display( )             // another instance function
{
y = 12; // VALID- instance or global variables can be used within any instance methods
d = 76; // INVALID- local variables can NOT be used out of the scope of its own
// function and see that the variable 'd' is declared within the function show( )
}// end of the function display( )
}// the class closes here
```

The cross (×) indicates that this task is not allowed.

Example 2 : The following figure also shows the concept of scope of variables :



The highlighted area is an **error** that appears during compilation of program because variable **'d'** is declared inside function **Show( )** which is a **local variable** to the function **Show( )** and *local variable **can not** be used outside its scope* and here variable **'d'** is used by function **display( )** which is **not valid**. On the other hand variables **'x'** and **'y'** are used within **both** the functions because they are **global** or **instance variables** and can be used by any function of the class.

This is the error message that appears because local variable **'d'** is declared inside the function **Show( )** and used by function **display( )**. *As per the rules of **Scope of variables** this is not allowed.*

2.11 Objects as Instance of class

As a class is a way of binding data and associated function(s) together in the form one unit and to get the desired output accessing (or executing) the class members is required. The task of accessing (or executing) the class members is achieved by a *special variable* that belongs to the *class* and known as **Object of the Class**.

Definition : An object is an identifiable entity with some characteristics and behaviour.

Let us consider the example of ATM of a bank. To access the services of the bank (*such as withdraw of money from ATM etc.*), we need to have an ATM card and its password / key / pin number because without these things nobody can access ATM machine. Similarly, if we treat ATM machine as a **class** and service provided by the ATM as the **members of the class**. To access or execute **the members of class an object** (or **instance**) is needed. Similarly, to **use** or **access** the *ATM services* the **ATM card** and **pin number** acts as an **object** or **instance**.

Thus, it can be said that **without an object** none of the members of class can be accessed. *Remember that the object of the class should **not** be a **keyword**.*

The following is the format to create an object of the class and to access the members :

Syntax 1 : **To create *object* of class :**

class_name object_name/variable = **new** class_name( );

Syntax 2 : **To *access* or *execute* or *run* data member(s) of class :**

object_name . data_member_name/variable;

Syntax 3 : **To *access method* or *member function* of class;** **[also known as function call / invoke]**

object_name . method/function_name(parameter/argument list);

Consider the following class along with data members and method:

Example :

```
class   sample
{
        int x, y;              // these are data members of class
public void display( )      // this is an instance function
{
x = 45; // assign 45 to variable 'x'
System.out.println("Result = " + x );
}// end of the function display( )
}// the class closes here
```

Consider the following examples to create an object of the above class and to access or execute the members class:

Example 1 : To create *object* of class use the following process described in syntax 1:

```
sample std = new sample( );        // here variable 'std' is object of class 'sample'
```

Example 2 : To *access* or *execute* or *run* data member or variable 'y' given in class 'sample' :

```
std . y = 3456;        // assign value '3456' to the variable 'y' means calling member of class
```

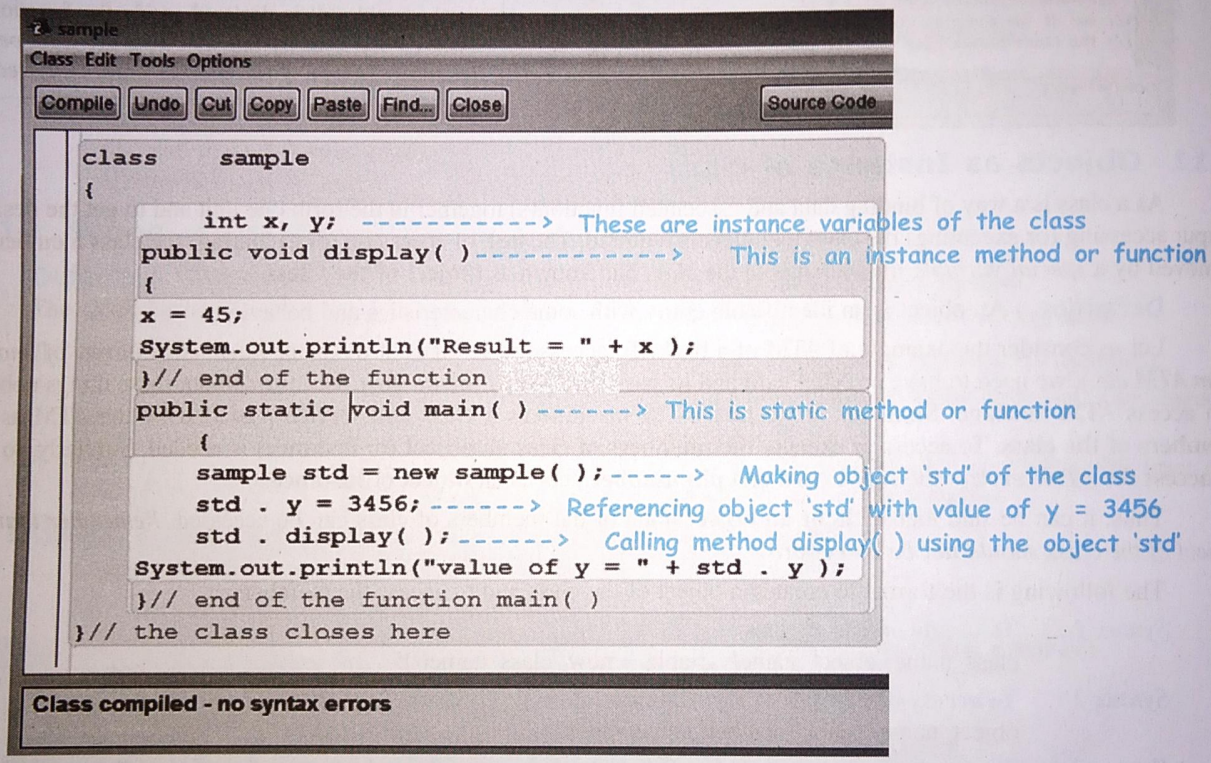
Example 3 : To *access or call or invoke the member method* or *member function* of class :

```
std . display( );        // this is function calling or invoking
```

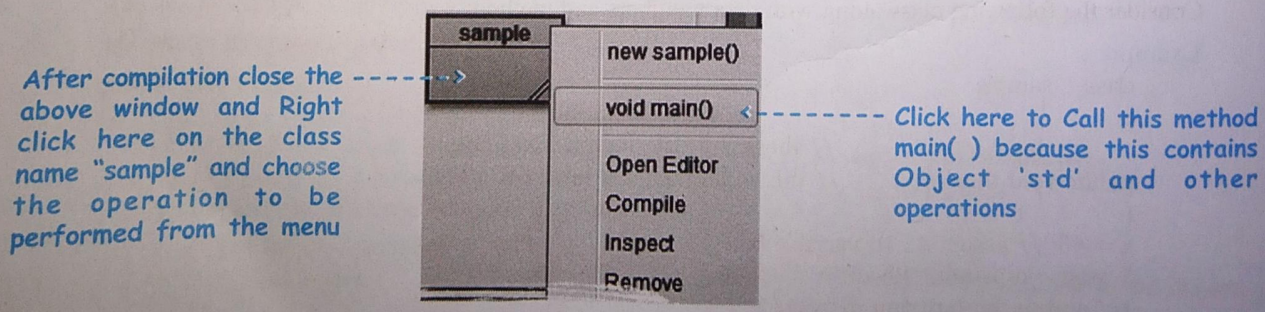
Consider one more example program with a class, data members, methods within class, object creation and method calling:

Example :

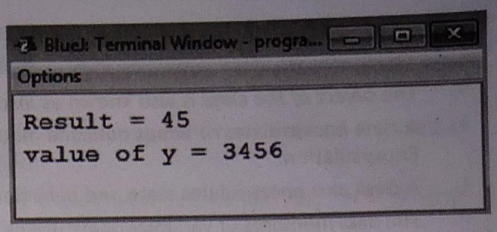
Step 1: Type a program in the BlueJ edit window as shown in the following figure and Compile:



Step 2: Call the method or function in which Object is created. Here it is main( ) method:



Step 3: The following figure shows the output terminal window with results:



2.12 Class as an Object Factory

As discussed in this chapter and in the revision tour that object plays an important role to manage the entire activities of a class. Thus, a class is considered as object factory because the data members and member methods of a class automatically captured by the object along with all its associated operations.

Let us consider the example of Credit or Debit cards etc., these cards can be considered as object factory of class bank. Each card contains details of the account holder on the chip or magnetic strip and account holder belongs to some bank (or a class).

Let us observe the following example that shows behaviour of object of class step by step:

Example :

```
class   sample
{
        int x, y;              // these are data members of class
public void display( )         // this is an instance function no. 1
{
        x = 45;
}// end of the function display( )
public void process( )         // this is an instance function no. 2
{
        x = x * 10;
}// end of the function process( )
public static void main( )   // this is static function
{
sample std = new sample( );  --> Creating object 'std' which contains -->  (object) std  x = 0  y = 0

std . y = 3456;  --> Object 'std' is updated as -->  x = 0  y = 3456  (object) std

std . display( );  --> After function call, Object 'std' is updated as -->  x = 45  y = 3456  (object) std

std . process( );  --> After function call, Object 'std' is updated as -->  x = 950  y = 3456  (object) std

System.out.println("Value of x = " + x );  ---->  It will print : Value of x = 950

System.out.println("Value of y = " + std . y );  ---->  It will print : Value of y = 3456
}// end of the function main( )
}// the class closes here
```

From the above example, it is clear that the class is an object factory because the object contains all the data members (x and y) and keeps on updating the object with the new values of data members at every stage. The object is used to call the data member or variable either directly (**std. y = 3456**) or by calling or invoking or executing the methods or functions (**std . display( )** and **std . process( )**).

Remember that to see the output of the above program, you will have to call the method or function **main( )** because this contains all the required operations.

Now, you may easily understand that class and object is an important factors of Java programming because anything you want to operate must be within the class and its method(s).

POINTS TO REMEMBER

- The *object of the class* is also known as *instance of the class.*
- A class encapsulates or wraps data and methods together in the form of single unit or single entity. So, a class implements Encapsulation.
- A class also encapsulates state and behaviour.
- The data members or variables or attributes can be referred as state and methods or functions can be referred as behaviour.
- Every class must be supported by its object.
- One or more than one objects for a class can be created as needed.
- The operator **new** (*refer to the syntax 1 and example 1*) is a keyword that allocates the memory space to an object.
- The **dot (.)** operator used after the **object_name** is known as **member operator** and helps in calling data member (or variable) and methods (or functions) of the class *i.e.* members of the class.
- The object plays an important role because its stores the data and its values.
- The object keeps on updating itself on the basis of the operations used within the methods of the class.
- The class is also known as **object factory** because the object of class contains (or stores) actual value of all the *instance, static* and *local variables.*
- A class can contain eight (8) primitive data types and all reference data types (String, Interface and Array).
- The data members of a class can also be called as attributes.
- Composite data type refers to the combination of more than one type of data. In other words composite data type comprise of various primitive data types or various reference data types or both the types.
- The class is known as composite data type because it may contains primitive data types as well as composite data types.
- The data members or variables starts with keyword **static** are known as **static variable** or **class variable** (e.g. **static int x;**).
- The methods or functions with keyword **static** are known as **static methods** or **class methods** (e.g. **static void show( )**).

Solved Questions

Q. 1 : What is the difference between an *object* and a *class* ? **ICSE 2011**

Ans. : **Object :** An object is an identifiable entity with some characteristics and behaviour.

Class : It represents a set of objects that share common characteristic and behaviour.

Q. 2 : Why are the variables declared within the class known as data members?

Ans. : These variables are declared within the scope of class so becomes its members and known as data members or instance variables of class.

Q. 3 : How you justify state and behaviour? Give one example.

Ans. : The variables declared within the class represents state and working on these variables within the method represents behaviour.

Example:

```
class    Demo
{
        int x;    char ch;// these are data members represents state
public void show( )
{
        x = 39;          x = x *30;      // working on state or variable represents behaviour
ch += 2;                                 // working on state or variable represents behaviour
}
}
```

Q. 4 : Explain *local variables*, *instance variables* and *class variables*.

Ans. : Brief details of these variables are as follows:

Local variables : are used inside blocks as counters or in methods as temporary variables and are used to store information needed by a single method.

Instance variables : are used to define attributes or the state of a particular object and are used to store information needed by multiple methods in the objects.

Class variables : are global to a class and to all the instances of the class and are useful for communicating between different objects of all the same classes or keeping track of global states. They are declared using **"static"** keyword.

Q. 5 : Name the primitive data types.

Ans. : Eight primitive data types provided by Java are : *byte*, *short*, *int*, *long*, *float*, *double*, *boolean*, *char*.

Q. 6 : Name the reference data types.

Ans. : The reference data types are : *class*, *interface* and *array*.

Q. 7 : What is a *static* variable?

Ans. : The *static* variables are shared by all instances or objects of a class. The static variable is also known as class variable and declared using keyword **"static"**.

Q. 8 : Which of the following are primitive types?

(a) byte (b) String (c) integer (d) Float

Ans. : byte

Q. 9 : Give the difference between *implicit type conversion* and *explicit type conversion*. ICSE 2010

Ans. :

Implicit type conversion	Explicit type conversion
1. The conversion performed by Java compiler automatically from lower data type to higher data type is called implicit type conversion. **2.** This can be done using various data types and there is no keyword to do this. **Example :** int x; float y, s; s = x * y; // **int x** is promoted **float**	**1.** To convert one data type to another data forced by user by giving the desired data type is known as explicit type conversion. **2.** The data type (int, float etc.) is used as keyword. **Example :** double m, n; float s = **(float)** m / n;

Q. 10 : Give the difference between *primitive data type* and *reference data type* with one example.

Ans. :

Primitive data type	Reference data type
1. The fundamental data types available with Java is known as primitive data type. **2.** The primitive data types are used to declare various variables. **Example:** int, float, char, double, long, short, byte, boolean.	**1.** The data types constructed by combining primitive data types is known as reference data type. **2.** The reference data type stores the address or the reference of the memory location of the actual data stored. **Example :** class, arrays, reference.

Q. 11 : Which operator is used to access instance variable of a class in the main( ) function ?

Ans. : The **dot (.)** operator.

Q. 12 : What is the purpose of the *new* operator?

Ans. : The **new** operator/keyword creates a single *instance* of a class and returns a reference to that *object*.

Q. 13 : What are methods and how are they defined? Give two examples.

Ans. : A method is a compound block that contains set of statements to be executed to perform a task.

The method definition has four parts : The **data type** (*like int, float, char, boolean, void etc.*), The **method name** (*any variable or user defined name*) and **one** or **more parameters** within the parenthesis, if any followed by *set of instructions* within the *curly braces* '{ }' (also known as body of the method).

Example 1:

```
void Display( )
{ int y = 3;
int x = y * 4;
System.out.println( x );
}
```

Example 2:

```
void Display( int y )
{
int x = y * 4;
System.out.println( x );
}
```

Note: In the **example 1** and **example 2** the keyword '**void**' is function's **data type** and **Display( )** is the name of the function (or method). In the **example 2** (**int y**) is the **parameter/argument** and within the **curly braces** '{ }' *executable set of statements are given as body of the function* (or *method*).

Q. 14 : A class Demo contains int n its member. Create an object of the class and assign 49 to n.

Ans. : The statement to create object is as: **Demo obj = new Demo( );**.
The statement to assign 49 to n is as: **obj . n = 49;**

Q. 15 : What is method invoking (or calling) ?

Ans. : To execute set of instructions given within the function is known as function calling (or invoking).

Q. 16 : What is the difference between a class variable and instance variable ? **ICSE 2011**

Ans. :

Class variable	Instance variable

1. The data member which is declared once for a class and all the objects of this class shares this data member is known as class variable.
2. A single copy of the variable accessible to all objects is available in the memory for operations.
3. The keyword **static** is used make a variable as class variable.
Example :
`class data`
`{`
`static int a;`
`}`
Here variable **'a'** is class variable.
	1. The data member which is created for every object of the class and is not shared is known as instance variable.
	2. Many copies of the variable are available in the memory as many number of objects.
	3. Only primitive data types are used to make a variable as instance variable.
	Example :
	`class data`
	`{`
	`int h, t;`
	`}`
	Here variable **'h', 't'** are instance variables.

Q. 17 : In the program given below, state the name and the value of the : **ICSE 2012**

(i) method argument or argument variable **(ii) class variable**
(iii) local variable **(iv) instance variable**

```
classmyClass
{
staticint x = 7; int y = 2;
public static void main( String args[ ] )
{
myClassobj = new myClass( );
System.out.println( x );
obj .sampleMethod( 5 );   int a = 6;
System.out.println( a );
}
```

```
voidsampleMethod( int n )
{
System.out.println( n );
System.out.println( y );
} }
```

Ans. : (i) Method argument or argument variable is: ***n***. (ii) The class variable is: ***x***.
(iii) The local variable is: ***a***. (iv) The instance variable is: ***y***.

Q. 18 : Correct the following statement for a class name *hockey* : obj hockey = new( ) hockey;

Ans. : The correct form is : **hockey obj = new hockey( );**

Q. 19 : A class CRICKET contains a function *void print( )*. Write a valid statement to create object and invoke the function.

Ans. :
```
CRICKET b1 = new CRICKET( );          // This statement creates object and its name is 'b1'
b1.print( );                          // This is function calling / invoking
```

Q. 20 : Identify the statements listed below as assignment, increment, method invocation or object creationstatements. **ICSE 2014**

(i) System.out.println("Java"); **(ii) costPrice = 457.50;**
(iii) Car hybrid = new Car(); **(iv) petrolPrice++;**

Ans. :
(i) System.out.println("Java"); ---> is method invocation statement.
(ii) costPrice = 457.50; ---> is assignment statement or variable initialization.
(iii) Car hybrid = new Car( ); ---> is objet creation statement where 'hybrid' is object of class.
(iv) petrolPrice++; ---> is an increment statement.

Q. 21 : Consider the following class: **ICSE 2014**

```
public class  myClass
{
public static int x = 3, y = 4;
public int a = 2, b = 3;
}
```

(i) Name the variables for which each object of the class will have its own distinct copy.
(ii) Name the variables that are common to all objects of the class.

Ans. :
(i) The variables : '**a**' and '**b**' (because these are *non-static or instance variables*)
(ii) The variables : '**x**' and '**y**' (because these are *static or class variables.*)

Q. 22 : Arrange the following primitive data types in Ascending order of their sizes. **ICSE 2015**
(i) char (ii) byte (iii) double (iv) int

Ans. : The ascending order list is as: **byte**, **char**, **int**, **double**.

Q. 23 : What is composite data type?

Ans. : Group of more than one primitive data type or reference data type as single unit (class) is known as composite data type. A class is known as composite data type.

Q. 24 : Give one example of composite data type.

Ans. : The example of composite data type is as:

```
class Demo
{
int x; float m;    char ch;
}
```

Q. 25 : Why is the class declared in question 24 called composite data type? Give reason.

Ans. : The class **Demo** in question 24 is known as composite data type because the class contains primitive data types (**int x**, **float m** and **char ch**) in one block named as **Demo** *i.e.* single entity, so this class is an example of composite data type.