Introduction to Object Oriented Programming Concepts

CHAPTER 1

Chapter Outline

1.1 INTRODUCTION

With the rapidly changing world and the highly competitive and versatile nature of industry, the operations are becoming more and more complex. In view of the increasing complexity of software systems, the software industry and software engineers continuously look for the new approaches to software design and development. The increased complexity had become the chief problem with computer programs in traditional languages. Large programs, because of this complexity, are more prone to errors, and software errors can be expensive and even life-threatening. The most adopted and popular programming approach, *structured programming approach*, failed to show the desired results in terms of bug-free, easy-to-maintain, and reusable programs. The latest programming approach, *Object-Oriented Programming (OOP)*, offers a new and powerful way to cope with this complexity. Its goal is : clearer, more reliable and, more easily maintained programs. This chapter introduces general OOP concepts that the traditional languages like C, Pascal, COBOL and BASIC lack in and the new generation languages (Object-Oriented Languages) support.

1.2 Evolution of Software

A program serves the purpose of commanding the computer. The efficiency and usefulness of a program depends not only on proper use of commands but also on the programming language it is written in. The two major types of programming languages : *Low Level* languages and *High Level* languages offer different features of programming.

Low Level languages (*i.e., machine language and assembly language*) are machine-oriented and require extensive knowledge of computer circuitry. *Machine language*, in which instructions are written in binary code (using 1's and 0's), is the only language the computer can execute directly. *Assembly language*, in which instructions are written using symbolic names for machine operations (*e.g.*, READ, ADD, STORE etc.) and operands, makes programming less tedious than machine language programming. However, assembly program is then converted into machine language using *assembler* software.

High Level languages, (HLLs), on the other hand, offer English like keywords, constructs for sequence, selection (decision) and iteration (looping) and use of variables and constants. Thus, it is very easy to program with such languages compared to low level languages. The programs written in HLLs are converted into machine language using *compiler* or *interpreter* software as a computer can execute machine language directly.

A programming language should serve *two* related purposes :

(*i*) it should provide a vehicle for the programmer to specify actions to be executed and

(*ii*) it should provide a set of concepts for the programmer to use when thinking about what can be done.

The first aspect ideally requires a language that is *"close to the computer machine"*, so that all important aspects of a machine are handled simply and efficiently. The second aspect ideally requires a language that is *"close to the problem to be solved"* so that the concepts of a solution can be expressed directly and concisely.

The *low level* languages serve only the first aspect *i.e.*, they are close to the machine and the *high level* languages serve only the second aspect *i.e.*, they are close to the programmer. However, the languages 'C' and 'C++' serve both the aspects, hence can be called as 'middle level languages'.

1.2.1 Programming Language Generations

Computer programming started with instructions being typed in binary format *i.e.*, with 0s and 1s and it evolved into today's sophisticated programming languages that offer many features to make our task much easier and effective. In the following lines, we are discussing various generations of programming languages that describe the evolution of programming languages.

1. First Generation Programming Language (1GL)

A **first-generation programming language** is a machine-level programming language or ***machine language*** in short. It consists of 1s and 0s. Originally, no translator software such as *compiler* or *interpreter* was used to compile or assemble the first-generation language.

The main benefit of programming in a first-generation programming language is that code a user writes can run very fast and efficiently since it is directly executed by the CPU, but machine language is somewhat more difficult to learn than higher generation programming languages, and it is somewhat more difficult to debug and edit if errors occur.

2. Second Generation Programming Language (2GL)

A **second-generation programming language** is a term usually used to refer to some form of assembly language. Unlike first-generation programming languages, it can actually be read and written fairly easily by a human as it uses mnemonics to specify instructions. It requires some translation (from assembly language to machine language) to make it useful to a computer. The translator that does so is known as **assembler**. However, the language is still specific to and dependent on a specific processor and environment and is difficult to use effectively to write large applications.

3. Third Generation Programming Language (3GL)

A **third generation language** (3GL) is a programming language designed to be easier for a human to understand, using normal language like words such as if, repeat, while etc.

A sample fragment might be :

```
let c = c + 2 * d
```

Or

```
If sales > 20000 then
    Discount = 10%
Else
    Discount = 5%
```

Fortran, ALGOL and COBOL are early examples of this sort of language. Most "modern" languages (BASIC, C, C++, Java etc.) are also third generation languages.

Third Generation Language programs also need to be converted into machine language. **Compilers** and **interpreters** are two such software that covert a 3GL program into machine language program.

4. Fourth Generation Programming Language (4GL)

4GL or **fourth-generation language** is designed to be closer to natural language than a 3GL language. 4GLS are *non-procedural in nature*. That is, programmers have just to specify WHAT is required rather than HOW it is to be done, which is done in case of 3GLs. Languages for accessing databases (such as SQL) are often described as 4GLs. A 4GL language statement might look like this :

```
EXTRACT ALL CUSTOMERS
WHERE "PREVIOUS PURCHASES" TOTAL MORE THAN ' 10000'
```

All 4GLs are designed to reduce :

- programming effort.
- the cost of software development.
- the time it takes to develop software.

One most popular 4GL is SQL, which is a query language used in databases. Other 4GLs include many report generators, screen painters, GUI creators etc.

5. Fifth Generation Programming Language (5GL)

A **fifth-generation programming language** is a programming language based around solving problems using constraints entered by the programmer, which the computer uses to solve the given problem.

The main difference between *fourth-generation programming languages* and *fifth-generation languages* is fourth-generation languages are designed to build specific programs while fifth generation languages are designed to make the computer solve the problem for you. This way the programmer only needs to worry about what problems needs to be solved and what conditions need to be met without worrying about how to implement a routine or algorithm to solve them.

Fifth-generation languages are used mainly in Artificial intelligence research. *Prolog, OPS5,* and *Mercury* are the best known fifth-generation languages.

After learning about various categories of programming languages, let us now talk about different ways of developing programs.

1.3 Programming Paradigms

By *paradigm* one means **a way of thinking or doing things.**

Since the invention of the computer, many programming approaches have been tried such as *procedural programming, modular programming, structural programming* etc. The primary motivation in each case has been the concern to handle the increasing complexity of programs that are reliable and maintainable.

Let us discuss these different programming paradigms and key language mechanisms necessary for supporting them.

Paradigm

Paradigm means organizing principle of a program. It is an approach to programming.

1.3.1 Procedural or Modular Programming

A program in a procedural language is a list of instructions where each statement tells the computer to do something. The focus is on the processing, *i.e.,* the algorithm needed to perform the desired computation. This *paradigm*[1] is :

Decide which procedures you want ;
use the best algorithms you can find.

Languages support this *paradigm by providing facilities for passing data values to functions* (sub-programs or sub-routines) and *returning values from functions.*

In procedural paradigm, the emphasis is on doing things and not on data. What happens to the data ? Data is, after all, the reason for a program's existence. The important part of an inventory program isn't a function that displays or checks data ; it is the inventory data itself. Yet data is given second-class status while programming.

1. A programming paradigm refers to a way/style of programming.

Also, in procedural programming, data types are used and worked upon by many functions. If a function makes any change to a data type, then it must be reflected to all the locations (functions), within the program that process this data type. This is very time-consuming (and of course, frustrating) for large-sized programs.

Thirdly, procedural programming does not model real world very well. For instance, a vehicle is an object, which is capable of moving in real world. However, the procedural programming paradigm would just be concerned about the procedure (or doing things) *i.e.*, the procedure programming paradigm would just think of moving part and not the vehicle. See the difference between real world interpretation of vehicle and procedural programming's interpretation of the same vehicle.

Modular Programming is an extension of procedural programming. In modular programming, a big program is divided into smaller parts known as *modules*.

Modules are identifiable pieces of code with a defined *interface*[2] within a program. They can be called from any part of a program and allow large programs to be split into more manageable tasks, each of which can be independently tested.

A module can be thought of as a 'black box' with a set of inputs and outputs. It has a sole purpose, and processes its inputs in a way dictated by its goal and provides some output. In most cases the actual operation of the 'black box' should be invisible to the rest of the program. Just like an educational program, where the course is broken down into a series of modules, which the student must undertake. For each module there is an *initial specification* (the input to the module, along with the students enrolled on the module), and the *output* are graduated students, with their results. As much as possible, these modules should be taken independently from the other modules on the program, as the teaching and operation of one module should not really have much effect on other modules. The amount that they are interrelated is known as *cross coupling*. For example, a student doing a *Cyber Law* course may also take a *Java module*. The cross coupling between this module, and the *Cyber Law* module is likely to be zero, as the teaching of one does not effect the other. However, some modules do have a strong coupling, such as the *Object-Oriented Design Methods* (OODM) and the *Java module*. This is because Java is an object-oriented language and implements object-oriented design methods. Even though two or more modules may have strong coupling, but it is tried at the level best to reduce the amount of coupling, so that the modules can run without effecting other modules.

Module

A Module is an identifiable piece of code within a program, with a set of inputs and outputs. It has a sole purpose, and processes its inputs in a way dictated by its goal and provides some output.

In modular programming, a program consists of a number of modules, which, in most cases, work independently of all others. Each module can work with its own data as well as with the data passed to it.

Figure 1.1 illustrates a function represented by an ideal 'black box' with inputs and outputs, and Fig. 1.2 shows a main function calling several *subfunctions* (or *modules*).

2. An interface refers to a way of interaction.

New techniques in software development use *components*, which are large general purpose modules, which can be used in a number of ways. For example, a networking component could be used in a program. If the component is well designed, it could be used in a number of ways, such as supporting different network protocols, or different network types.

In **Modular Programming**, a program consists of a number of *modules*, which, in most cases, work independently of all others. Each module can work with its own data as well as with the data passed to it.

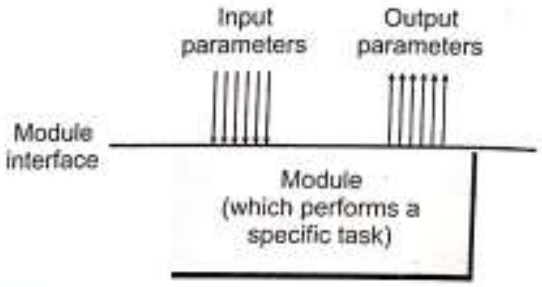


Figure 1.1 An ideal 'black-box' representation of a module.

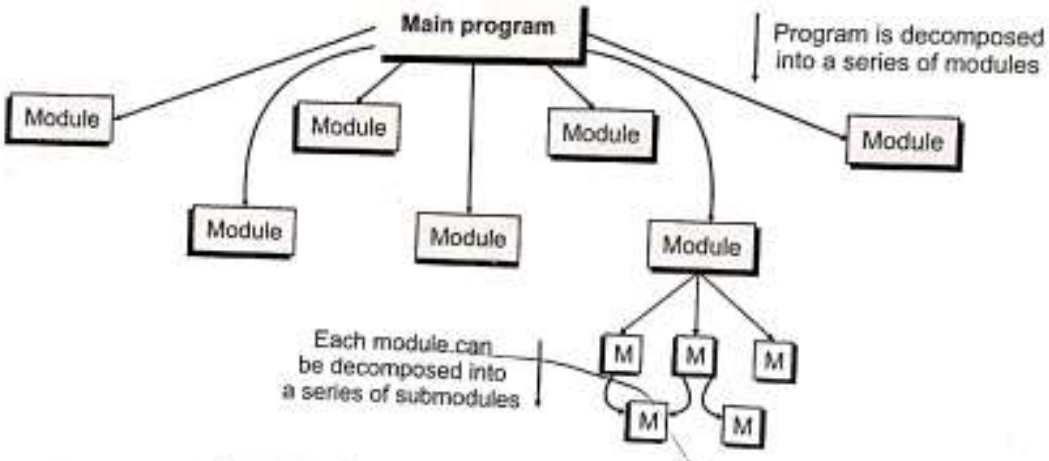


Figure 1.2 Hierarchical decomposition of a program.

Limitations of Procedural Programming Approach

(*i*) Emphasis on algorithm (or procedure) rather than data. Data takes the back seat with this programming paradigm.

(*ii*) Change in a datatype being processed needs to be propagated to all the functions that use the same datatype. This is a frustrating and time-consuming process.

(*iii*) The procedural programming paradigm does not model real world very well.

1.3.2 The Object Oriented Programming

After procedural programming, let us talk about *object-oriented* programming. The *object-oriented* approach views a problem in terms of *objects* involved rather than *procedure* for doing it.

Do you know what an object is ? Well, an *object* is an identifiable entity with some characteristics and behaviour.

Object

Object is an identifiable entity with some characteristics and behaviour.

For instance, we can say 'Orange' is an object. Its characteristics are : *it is spherical shaped, its colour is orange* etc. Its behaviour is : *it is juicy* and *it tastes sweet-sour*. While programming using OOP approach, the characteristics of an object are represented by its data, and its behaviour is represented by its associated functions. Therefore, *in OOP programming, object represents an entity that can store data and has its interface through functions.*

To understand this concept more clearly, let us consider an example. Let us simulate traffic flow at a red light crossing.

As you know, procedural programming paradigm focuses on the procedures or the working action. Using procedural programming paradigm, the above said problem will be viewed in terms of working happening in the traffic-flow *i.e.*, moving, halting, turning etc. The OOP paradigm, however, aims at the objects and their interface. Thus, in OOP approach, the traffic-flow problem will be viewed in terms of the objects involved. The objects involved are : cars, trucks, buses, scooters, auto-rickshaws, taxis etc.

With this elaboration, the concept must be clear enough to proceed for the OOP paradigm definition. But before that let us understand another very important term *class*.?

In the above example, cars have been identified as objects. They have characteristics like : steering wheel, seats, a motor, brakes etc. and their behaviour is their mobility. Car, however, is not an object, it is a class (compare with the definition). *'Optra'*[3] is an object of class type *'car'*. *'Car'* is a subclass of another class *'automobiles'* which again is a subclass of *'vehicles'*. *'Object'* is an instance of *'class'*. For example, we can say *'bird'* is a class but *'parrot'* is an object.

Now, after understanding the basic terminology, we can define Object-Oriented Programming (OOP) paradigm as :

> Decide which classes and objects are needed ;
> provide a full set of operations for each class.

Class

A *Class* is a template / blueprint representing a group of objects that share common properties and relationships.

Object

While class is a conceptual blue-print, an object is an actual entity which is an instance of a class.

That means, *object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationship.*

How OOP Overcomes Procedural Paradigm's Problems ?

As you already are aware of the shortcomings of procedural paradigm *i.e.*,

(*i*) its emphasis on procedures rather than data,

(*ii*) need for propagating updates and

(*iii*) not able to model real world well, let us see how these shortcomings are overcome by OOP.

The object-oriented approach overcomes these shortcomings by the following things :

(*i*) OOP approach gives data the prime consideration and by providing an interface through the functions associated with it.

3. To be precise, a specific Chevy Optra Car with a registration number say *XYZ 2197* is an object of 'Car' class.

(ii) An object is a complete entity *i.e.*, it has all the data and associated functions within it. Whenever something is to be changed for an object, only its class gets changed because it is complete in itself. All the functions that are working on this data or using it are defined within the class, they get to see the change immediately. And nowhere else change is required.

It is just like the way we replace a defective part, for instance a transistor, in an equipment. Since a transistor is a complete part, complete with its components and interface, a defective transistor can be replaced with a new transistor without altering the rest of the machinery. Similarly, if we make a software complete in its components and its interface, such a software can easily be replaced with a new one without altering the rest of the software.

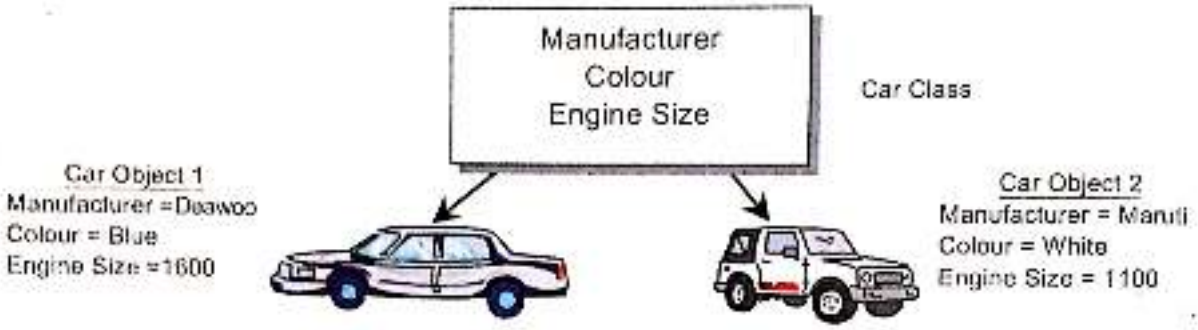


Figure 1.3 Car class and object.

Object-oriented programming is an excellent programming technique as it involves creating a program around the data, and not, as in modular programming, around the modules. This allows for a much better definition of the problem.

Figure 1.4 illustrates the basic conceptual difference between traditional procedural paradigm and OO paradigm.

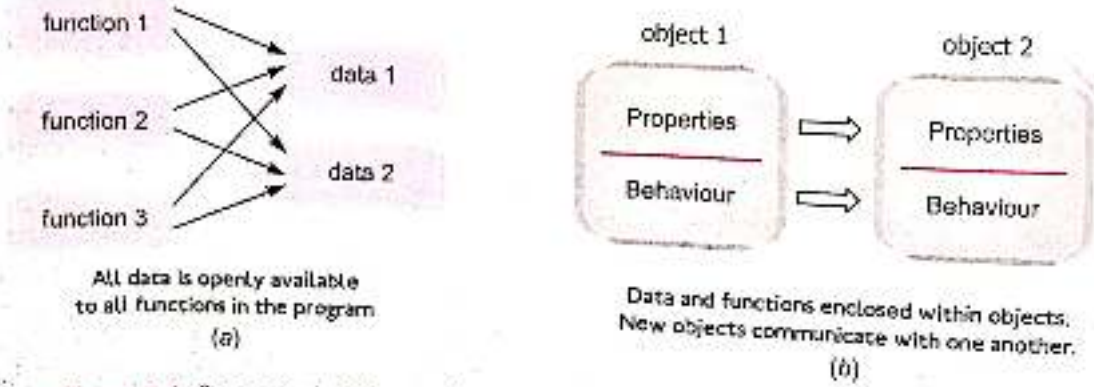


Figure 1.4 Conceptual difference between procedural paradigm and OO paradigm.

1.4 Basic Concepts of OOP

The object oriented programming has been developed with a view to overcome the drawbacks of conventional programming approaches. The OOP approach is based on certain concepts that help it attain its goal of overcoming the drawbacks or shortcomings of conventional programming approaches.

1.4.1 Data Abstraction

Abstraction is the concept of simplifying a real world concept into its essential elements. To understand abstraction, let us take an example. You are driving a car. You only know the essential features to drive a car *e.g.*, gear handling, steering handling, use of clutch, accelerator, brakes etc. But while driving do you get into internal details of car like wiring, motor working etc. ? You just change the gears or apply the brakes etc. What is happening inside is hidden from you. This is abstraction where you only know the essential things to drive a car without including the background details or explanations. Take another example of 'switchboard'. You only press certain switches according to your requirement. What is happening inside, how it is happening etc. you needn't know. Again this is abstraction – you know only the essential things to operate on switchboard without knowing the background details of switchboard.

Data Abstraction

Abstraction or Data Abstraction refers to the act of representing essential features without including the background details or explanations.

Data Hiding

Data hiding is a related concept. When abstraction exposes the necessary or required details, it hides the rest of the details, which is data hiding. *For example,* in a Switchboard, only switches and a board is exposed ; other details like circuitry and wiring etc. are hidden.

1.4.2 Encapsulation

Recall that the **characteristics** and **behaviour** together make an object. In a programming language, *characteristics* are represented through ***data*** and *behaviour* through ***functions***. **Encapsulation** is the way of combining both data and the functions that operate on that data under a single unit (called **member functions** or **methods**).

Encapsulation

The wrapping up of data and operations/functions (that operate on the data) into a single unit (called class) is known as *Encapsulation*.

The data cannot be accessed directly. If you want to read a data item in an object (an instance of the class), you call a **method** or **member function** in the object to do it. The data is hidden, so it is safe from accidental alteration. Data and its functions are said to be encapsulated into a single entity.

Let us now consider an analogy to encapsulation.

In a big company, there are so many departments, Sales, Accounts, Payroll, Purchase, Production etc. Each department has its own personnel that maintain its data. Suppose an employee in the production department wants to know how much raw material has been purchased for the next month. The production dept employee would not be allowed to himself go through the purchase department's data files. Rather he'll have to issue a memo to the 'purchase' requesting for the required information.

Then some employee of the 'purchase' department will go through the purchase data files and send the reply with the asked information. This practice ensures that the data is accessed accurately and that it is not corrupted by inept outsiders. Therefore, we can say here 'Department data and department employees are encapsulated into a single entity, the department'.

In the same way, objects provide an approach to program organization while helping to maintain the integrity of the program data (see Fig. 1.5).

Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT). 'Data types' because these can be used to create objects of its own type.

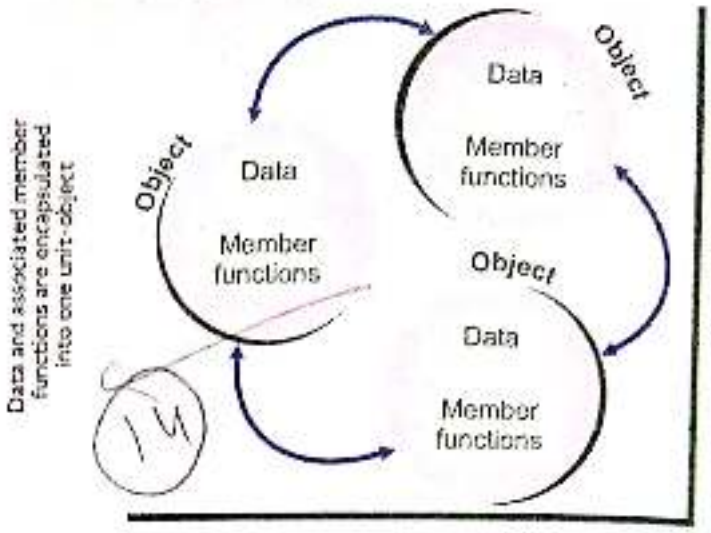


Figure 1.5 The OOP approach (Data Abstraction & Encapsulation).

1.4.3 Inheritance

Understanding inheritance is critical to understanding the whole point behind object oriented programming. For instance, we are humans. We inherit from the class 'Humans' certain properties, such as ability to speak, breathe, eat, drink etc. But these properties are not unique to humans.

The class 'Human' inherits these properties from the class 'Mammal' which again inherits some of its properties from another class 'Animal'. We take our same old example of cars. The class 'Car' inherits some of its properties from the class 'Automobiles' which inherits some of its properties from another class 'Vehicles'. The capability to pass down properties is a powerful one. It allows us to describe things in an economical way.

The object oriented languages express this inheritance relationship by allowing one class to inherit from another. Thus a model developed by languages is much closer to the real world. The principle behind this sort of division is that each subclass shares common characteristics with the class from which it is derived (Fig. 1.6).

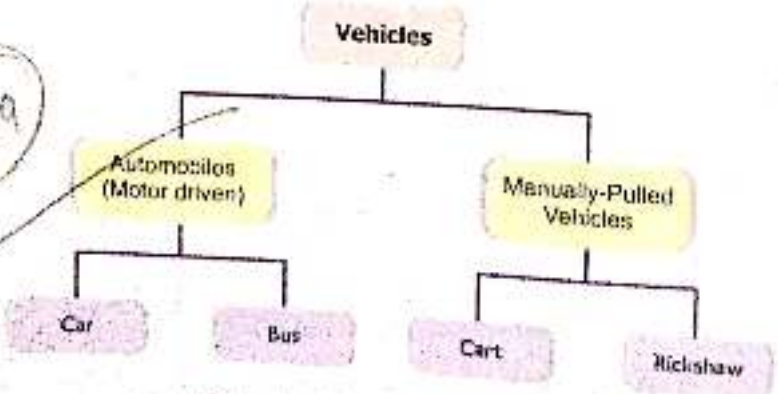


Figure 1.6 Property inheritance.

'Automobiles' and 'Pulled Vehicles' are subclasses of 'Vehicles'. 'Vehicles' is base class of 'Automobiles' and 'Pulled Vehicles'. 'Car' and 'Bus' are sub-classes of 'Automobiles'. 'Automobiles' is the base-class of 'Car' and 'Bus'.

Inheritance

Inheritance is the capability of one class of things to inherit capabilities or properties from another class.

Some Facts and Terms

When we say that the class 'student' inherits from the class 'Person', then 'Person' is a base class of 'Student' and 'Student' is a **subclass** (or derived class) of 'Person'.

Although 'Student' is a 'Person' yet the reverse is not true. A 'Person' need not be 'Student'. (All members of 'Student' are not members of class 'Person'.) The class 'Student' has properties it does not share with class 'Person'. For instance, 'Student' has a 'marks-percentage', but 'Person' does not have any marks. A subclass defines only those features that are unique to it. So, a subclass has features of its *baseclass* and some additional features that are unique to it.

Base Class & Subclass

A **base class** is a super class from which another class inherits properties. Inheriting class is called **subclass (derived class)**.

1.4.4 Polymorphism

Polymorphism is key to the power of object oriented programming. It is so important that languages that don't support polymorphism cannot advertise themselves as OO languages. Languages that support classes but not polymorphism, are called object-based languages. Ada is an object-based language.

Polymorphism is the concept that supports the capability of an object of a class to behave differently in response to a message or action. For instance, 'Human' is a subclass of 'Mammal'. Similarly, 'Dog', 'Cat', are also subclasses of 'Mammal'. Mammals can see through day-light. So if a message *'see through daylight'* is passed to all mammals, they all will behave alike. Now if a message *'see through darkness'* is passed to all mammals, then humans and dogs will not be able to view at night whereas cats will be able to view during night also. Here cats (mammals) can behave differently than other mammals in response to a message or action. This is *polymorphism*.

Polymorphism

Polymorphism is the ability for a message or data to be processed in more than one form.

Take another example. If you give 5 + 7, it results into 12, the sum of 5 and 7. And if you give *'A'* + *'B'*, it results into *'AB'*, the concatenated strings.

The same operation symbol '+' is able to distinguish between the two operations (summation and concatenation) depending upon the data type it is working on.

Figure 1.7 illustrates that a single function name can be used to handle different number and different types of arguments. This is something similar to a particular word having several different meanings depending on the context.

Polymorphism is a property by which the same message can be sent to objects of several different classes, and each object can respond in a different way depending on its class.

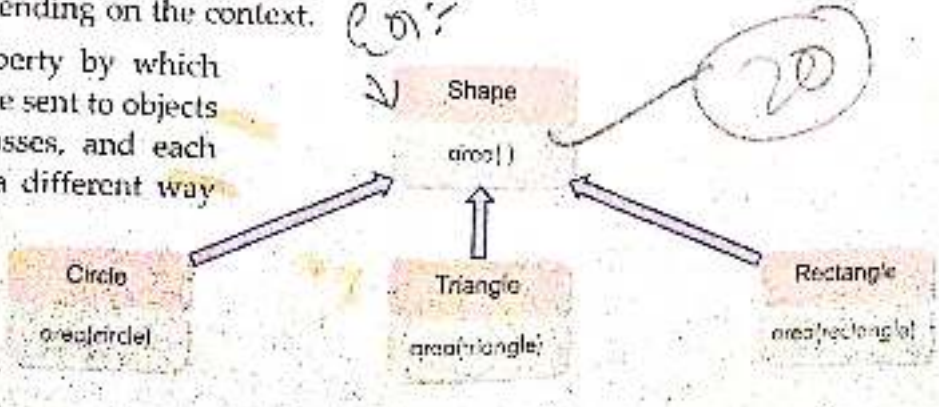


Figure 1.7 Polymorphism.

1.5 Advantages and Disadvantages of OOP

Finally, before winding up this chapter, let us quickly highlight the advantages of Object Oriented style of coding over other programming methodologies.

The **advantages** offered by OOP are :-

1. *Re-use of code.* Linking of code *i.e.*, functions to objects allows related objects to share code. Encapsulation allows class definitions to be re-used in other applications. The availability of a consistent interface to objects lessens code duplication and thereby improves code re-usability.

2. *Ease of comprehension.* The classes can be set up to closely represent the generic application concepts and processes. OOP codes are more near to real-world models than other programming methodologies' codes.

3. *Ease of fabrication and maintenance.* The concepts such as encapsulation, data abstraction allow for very clean designs. When an object is going into disallowed states, which are not permitted, only its methods need be investigated *e.g.*, if student is getting more than maximum marks, only the functions are to be retested. This narrows down search for problems.

4. *Easy redesign and extension.* The same concepts facilitate easy redesign and extension.

Although OOP has proved revolutionary in software development yet it has some **disadvantages** too. These are :

(a) With OOP, classes tend be overly generalised.

(b) The relations among classes become artificial at times.

(c) The OOP programs' design is tricky.

(d) Also one needs to do proper planning and proper design for OOP programming.

(e) To program with OOP, programmer need proper skills such as design skills, programming skills, thinking in terms of objects etc.

LET US REVISE

- *Low Level Languages (LLLs) i.e., machine language and assembly language are more close to the machine as these are machine oriented and require extensive knowledge of computer circuitry.*
- *High Level Languages (HLLs) are more close to the programmer as these offer English like keywords and programming constructs (sequence, selection, iteration).*
- *Procedural Programming aims more at procedures. The emphasis is on doing things. Data takes the back seat here.*
- *Modular Programming combines related procedures in a module and hides data under modules. The data are dependent on the functions. The arrangement of data can't be changed without modifying all the functions that access it.*
- *Object oriented programming is based on the principles of data hiding, abstraction, encapsulation, modularity, inheritance, and polymorphism. It implements programs using the objects in an object oriented language.*
- *Object represents data and its associated functions under single unit.*
- *Class represents a group of similar objects.*
- *Abstraction is the way of representing essential features with background details.*

- *Encapsulation is the way of combining data and its associated functions under single unit. Encapsulation implements abstraction.*
- *Modularity is the property of decomposing a system into a set of cohesive and loosely coupled modules.*
- *Inheritance is the capability of a class to inherit properties from another class. That class that inherits from other class is subclass or derived class and the other class is base class.*
- *Inheritance supports reusability and is transitive in nature.*
- *A subclass defines only those features that are unique to it, rest it inherits from its base class.*
- *Polymorphism is the ability for a message or data to be processed in more than one form. The same operation is performed differently depending upon the data type it is working upon.*

Conceptual Questions

Section A : Objective Type Questions

1. An object is a/an _____ of a class.
   (*a*) member (*b*) instance
   (*c*) interface (*d*) alternate name
2. A blueprint that represents characteristics and behaviour of a group of entities is called _____.
   (*a*) object (*b*) abstraction
   (*c*) class (*d*) polymorphism
3. Exposing only essential features while hiding unnecessary details is _____.
   (*a*) data hiding (*b*) encapsulation
   (*c*) modularity (*d*) abstraction
4. Shubhra is writing an *OOP* code for items like *car, truck, bus* etc. These items under the class *vehicle* are called _____ .
   (*a*) methods (*b*) items
   (*c*) objects (*d*) attributes
5. Wrapping up of characteristics and behaviour in one single unit is _____.
   (*a*) Encapsulation (*b*) Abstraction
   (*c*) Inheritance (*d*) Polymorphism
6. Wrapping up of data and associated functions in one single unit is called _____.
   (*a*) Encapsulation (*b*) Abstraction
   (*c*) Inheritance (*d*) Polymorphism
7. The programming languages that require knowledge of computer machine's architecture and circuitry are known as :
   (*a*) High Level Languages (*b*) Low Level Languages
   (*c*) Machine Language (*d*) Assembly Language.

8. The programming language paradigm that emphasizes on 'doing' or 'procedure of doing' is called _____.
   (a) Modular paradigm
   (b) Procedural programming paradigm
   (c) Object oriented programming
   (d) Object based programming paradigm
9. The programming languages that use objects and classes and other *OOP* concepts except Polymorphism are called _____.
   (a) Object oriented programming, language
   (b) Object class programming language
   (c) Object based programming language
   (d) Simple object programming language
10. A class inheriting features of another class and adding its unique features to it is _____.
   (a) Base class
   (b) Super class
   (c) Subclass
   (d) Low class
11. When an act can be performed in multiple ways by objects and subclasses of a class, this is known as _____.
   (a) Encapsulation
   (b) Multiprogramming
   (c) Polymorphism
   (d) Inheritance

Section B : Subjective Type Questions

1. *Write a short note on the programming in low level languages.*

   Ans. Low level languages (machine language and assembly language) are machine-oriented and require extensive knowledge of machine circuitry. In machine language, the instructions are given in binary codes whereas in assembly language, instructions are given in symbolic names (mnemonics).

2. *Write a short note on the programming in high level languages.*

   Ans. High Level Languages (HLLs) like BASIC, PASCAL etc. offer English like keywords, programming constructs for sequence, selection (decision) and iteration (looping), and allow use of variables and constants. Programmers feel much at ease while working in HLLs as compared to low level languages.

3. *What do you understand by procedural programming paradigm ?*

   Ans. The programming approach that focuses on the procedures for the solution of a problem is known as procedural programming paradigm. This approach emphasizes on the 'doing' rather than the 'data'.

4. *How is modular programming approach different from procedural programming approach ?*

   Ans. In modular programming, a set of related procedures with the data they manipulate is combined under a unit called *module*. Where there is no grouping of procedures with related data, the procedural programming approach still applies.

5. *What is the difference between an object and a class ?*

   Ans. An object is an identifiable entity with some characteristics and behaviour. It represents an entity that can store data and its associated functions.

   A class is a blueprint representing a group of objects that share common properties and relationships. While class is a logical entity, an object is an actual entity and is an instance of a class.

6. *What do you mean by Abstraction and Encapsulation ? How are these two terms interrelated ?*

Ans. Abstraction is the act of representing essential features without including the background details.

Encapsulation is the way of combining both data and the functions that operate on the data under a single unit.

Encapsulation is the way of implementing abstraction.

7. *What is base class ? What is derived class ? How are these two interrelated ?*

Ans. A derived class is a class that inherits properties from some other class.

A base class is a class whose properties are inherited by derived class.

A derived class has nearly all the properties of base class but the reverse of it is not true.

8. *How are object-based programming languages different from object-oriented programming languages ?*

Ans. Object-based programming languages implement *OOP* features except **Polymorphism**. Object-oriented programming languages however implement all *OOP* features.

9. *What is polymorphism ?*

Ans. It is ability for a message or data to be processed in more than one form. It is a property by which the several different objects respond in a different way (depending upon its class) to the same message.

10. *Write two major differences between Object Oriented Programming and Procedural Programming.*

Ans. *Procedural programming* paradigm aims more at procedures. The emphasis is on doing things rather than the data being used. In procedural programming paradigm, data are shared among all the functions participating thereby risking data safety and security.

Object oriented programming paradigm is based on the principles of data hiding, abstraction, inheritance and polymorphism. It implements programs using classes and objects. In OOP paradigm, data and procedures, both are given equal importance. Data and functions are encapsulated to ensure data safety and security.

KEYWORDS

Abstraction The act of representing essential features without including the background details or explanations.

Base Class A class whose properties are inherited by other classes (its subclasses).

Class Group of objects that share common properties and relationships.

Derived Class A class which inherits properties from its base class.

Encapsulation Wrapping up of data and functions (that operate on the data) into a single unit.

Inheritance Capability of one class of things to inherit capabilities or properties from another class.

Modularity The property of decomposing a system into a set of cohesive and loosely coupled modules.

Modular Programming Paradigm Decide which modules you want ; partition the program so that data is hidden in modules.

Module A set of related procedures with the data they manipulate.

Object An identifiable entity with some characteristics and behaviours.

6. *What do you mean by Abstraction and Encapsulation ? How are these two terms interrelated ?*

Ans. Abstraction is the act of representing essential features without including the background details.

Encapsulation is the way of combining both data and the functions that operate on the data under a single unit.

Encapsulation is the way of implementing abstraction.

7. *What is base class ? What is derived class ? How are these two interrelated ?*

Ans. A derived class is a class that inherits properties from some other class.

A base class is a class whose properties are inherited by derived class.

A derived class has nearly all the properties of base class but the reverse of it is not true.

8. *How are object-based programming languages different from object-oriented programming languages ?*

Ans. Object-based programming languages implement OOP features except **Polymorphism**. Object-oriented programming languages however implement all OOP features.

9. *What is polymorphism ?*

Ans. It is ability for a message or data to be processed in more than one form. It is a property by which the several different objects respond in a different way (depending upon its class) to the same message.

10. *Write two major differences between Object Oriented Programming and Procedural Programming.*

Ans. *Procedural programming* paradigm aims more at procedures. The emphasis is on doing things rather than the data being used. In procedural programming paradigm, data are shared among all the functions participating thereby risking data safety and security.

Object oriented programming paradigm is based on the principles of data hiding, abstraction, inheritance and polymorphism. It implements programs using classes and objects. In OOP paradigm, data and procedures, both are given equal importance. Data and functions are encapsulated to ensure data safety and security.

KEYWORDS

Abstraction The act of representing essential features without including the background details or explanations.

Base Class A class whose properties are inherited by other classes (its subclasses).

Class Group of objects that share common properties and relationships.

Derived Class A class which inherits properties from its base class.

Encapsulation Wrapping up of data and functions (that operate on the data) into a single unit.

Inheritance Capability of one class of things to inherit capabilities or properties from another class.

Modularity The property of decomposing a system into a set of cohesive and loosely coupled modules.

Modular Programming Paradigm Decide which modules you want ; partition the program so that data is hidden in modules.

Module A set of related procedures with the data they manipulate.

Object An identifiable entity with some characteristics and behaviours.

Object Oriented Paradigm Decide which classes and objects are needed ; provide a full set of operations for each class.

Paradigm Organizing principle of a program. An approach to programming.

Polymorphism Property by which the same message can be sent to objects of several different classes, and each object can respond in a different way depending on its class.

Procedural Programming Paradigm Decide which procedures you want ; use the best algorithms you can find.

Subclass Derived class.

Assignment

1. What are the two major types of programming languages ?
2. Briefly explain the two major types of programming languages.
3. Why are low level languages considered close to the machine ? 12
4. Why is it easier to program with high level languages ? Why are high level languages considered close to the programmer ? 12
5. Can you give examples of programming languages belonging to each generation ? 3, 4
6. What are programming paradigms ? Give names of some popular programming paradigms. 1
7. What is major characteristic of procedural programming ? 4
8. What is major characteristic of modular programming ? How is it similar to procedural programming ? 5
9. What are the shortcomings of procedural and modular programming approaches ? 14
10. Write a short note on OO programming. 6,7
11. How does OOP overcome the shortcomings of traditional programming approaches ? 7
12. What is the difference between object and class ? Explain with examples. 7
13. Explain briefly the concept of data abstraction with the help of an example. 9
14. Explain briefly the concept of encapsulation with the help of an example. 9,10
15. How the data is hidden and safe if encapsulation is implemented ? Explain with example. 9
16. Simulate a daily life example (other than the one mentioned in the chapter) that explains encapsulation.
17. What is inheritance ? How is it useful in programming terms ? 10,11
18. What are the advantages offered by inheritance ? 10
19. How is reusability supported by inheritance ? Explain with example. 10
20. What is polymorphism ? Explain with example. 11
21. Do you think OOP is more closer to real world problems ? Why ? How ? 08
22. What are object-based languages ? Give an example of object-based language.
23. What are the advantages and disadvantages of using object oriented programming ? 12

Introduction to Java

CHAPTER 2

Chapter Outline

2.1 INTRODUCTION

Java is a popular third-generation programming language, which can be used to do any of the thousands of things that a computer software can do. With the features it offers, Java has become the language of choice for Internet and intranet applications. Java plays an important role for the proper functioning of many software-based devices attached to a network. The kind of functionality the Java offers, has contributed a lot towards the popularity of Java.

This chapter is going to talk about a brief history of Java, its important features, functionality etc. This chapter also talks about how you can create Java programs on BlueJ environment.

2.2 About JAVA

Java is both a *programming language* and a *platform*. Like any other programming language, you can use Java to write or create various types of computer applications. Thus, Java fits the definition of a programming language.

Java is also a *platform* for application development. The word *platform* generally is used to refer to some combination of hardware and system software *e.g.*, operating system *Windows 10 on Intel Core i7* or *Windows 10 on DEC Alphas* or *MacOS 13.2 on PowerMacs* etc. The Java Platform is a software platform designed to deliver and run highly interactive, dynamic and secure applications on networked computer systems.

2.2.1 History of Java

Java, was originally named as **Oak**. But with the explosion of world wide web, Java rose to charts of popularity as it could cater to nearly all platforms. In the following lines, the history of Java can be found out in the **Java timeline**.

1991	*James Gosling* Develops **Oak** (later renamed ***Java***) language for programming intelligent consumer electronic devices.
1993	World Wide Web explodes.
1995	*Java* formally announced. Incorporated into Netscape web browser.
1996	*Java Development Kit* (jdk) 1.0 Ships. JavaBeans component architecture Corel Office for Java Preview. Sun announces JavaStation Network Computer. Sun announces 100% Pure Java initiative.
1997	JDK 1.1 launched. Java Servlet API released.
1998	Sun introduces Community Source "open" licensing. Sun produces a JDK 1.2 for Linux.
1999-2001	JDK 1.3 released. JDK's also produced by IBM on multiple platforms. Java based Application Servers (BEA, IBM webSphere etc.) become popular. J2EE (Java 2 Enterprise Edition), J2SE (Java 2 Standard Edition), J2ME (Java 2 Micro Edition) appear.
2002	Java support for Web services officially released via the *Java™ Web Services Developer Pack.*
2004	*Java 2 Platform, Standard Edition 5* (*Project Tiger*) is released; The Java technology-powered Mars Rover (*Spirit*) touches down on Mars; *Sun Java Studio Creator* is launched.
2005	Java technology celebrates its 10th birthday; Approximately 4.5 million developers use Java technology; Over 2.5 billion Java technology-enabled devices are available.
2006	The *NetBeans IDE 5.0* is released. Sun open-sourced Java EE components as the Glassfish Project to java.net. Java SE and ME initial components are released as open source.
2007	Java technology is in more than 5.5 billion devices and is used by more than six million developers.
2008	JavaFX 1.0 is released; At JavaOne, Neil Young announces his Archives Project to be released on Blu-ray Disc™, powered by Java technology.
2009	The NetBeans IDE is developer.com's 2009 Product of the Year.
2010	Oracle acquires Sun Microsystems. The JCP approves Java 7 and Java 8 roadmaps.
2013	After a stagnation phase, Java resurges with major improvements – major updates, lambdas, streams, method references, defender methods, new HTTP clients etc.
2017, 2018	Released updates of latest Java versions *Java SE 8* and *Java SE 9*.

2.2.2 Understanding Bytecode

After the programmer has written the program, the program-code (the **source code**) must be converted into machine-understandable executable code (**object-code/Native executable code**), so that computer can understand and run the code. This is done by compiler for different programming languages. (see Fig. 2.1(*a*) below)

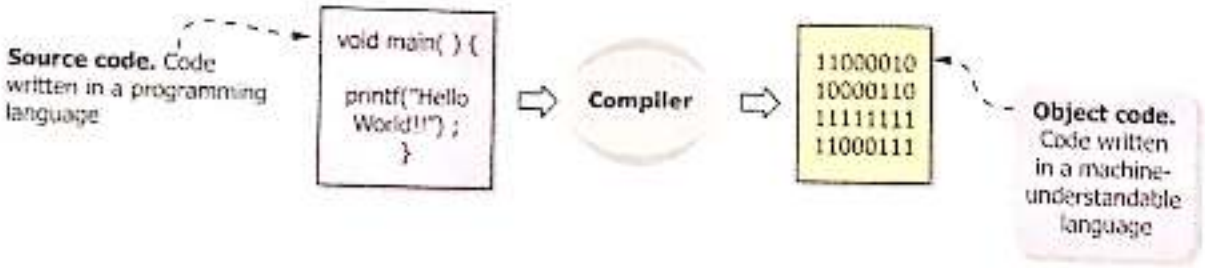


Figure 2.1 (*a*) Compilation process

For a different platform, you need to change the compiler and sometimes the code too[1]. But for the applications developed with Java, this is not the case. Java applications are compiled once only and the Java compiler does not produce ***object-code***, rather it produces bytecode. The ***bytecode*** makes the Java applications platform-independent.

In order to understand the bytecode, you must be clear about compilation process first, which is being discussed below.

2.2.2A Ordinary Compilation Process

The process of converting a source code into machine code is called **compilation**. [Fig. 2.1(*a*)] The converted machine code depends a lot on the platform it is executing upon. That means for different platforms different machine code is produced [Fig. 2.1(*b*)].

This resultant machine code is called ***native executable code***.

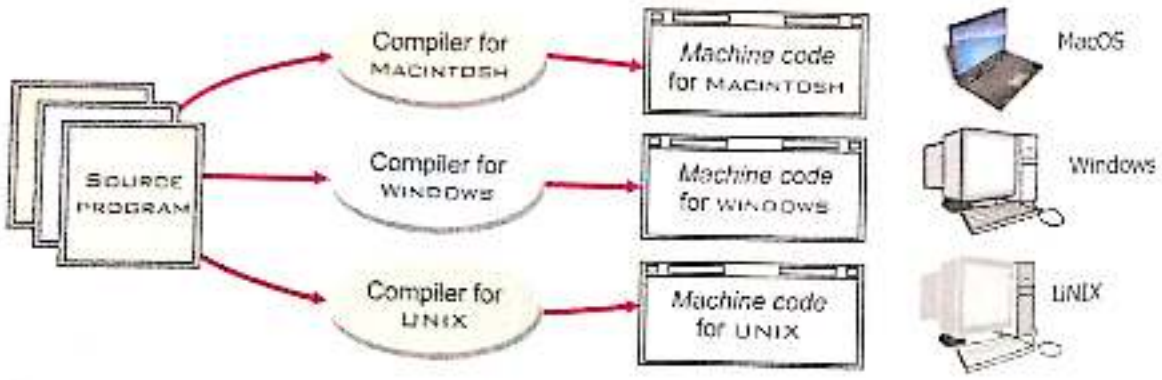


Figure 2.1 (*b*) Ordinary compiler produces platform-specific executable

The *Native executable code* (*machine code*) is directly executed on the corresponding computer for which it has been created. *For example*, as you can see in Fig. 2.1(*b*), a Macintosh compiler

1. Because each platform requires different type of executable, e.g., Windows requires an EXE file, Linux requires a Linux executable, and so on.

creates a native executable code, which is directly executed on a Macintosh computer. Similarly, Windows Compiler creates native executable code that is directly executed on a Windows computer and so on.

2.2.2B Java Compilation Process

Programs written in Java are compiled into a special type of machine language/But it is a machine language for a virtual computer known as the **Java Virtual Machine**, or **JVM**. The machine language for the *Java Virtual Machine* is called **Java bytecode.**

The bytecode still needs an interpreter to execute them on any given platform. The interpreter, also called **Just In Time Compiler (JIT compiler)** reads the bytecode and translates it to the native executable code of the host computer on which it is to be executed. The JIT compiler is part of the JVM, and it compiles bytecode into executable code as per the host machine, i.e., the machine on which the code is to be executed. (see Fig. 2.2).

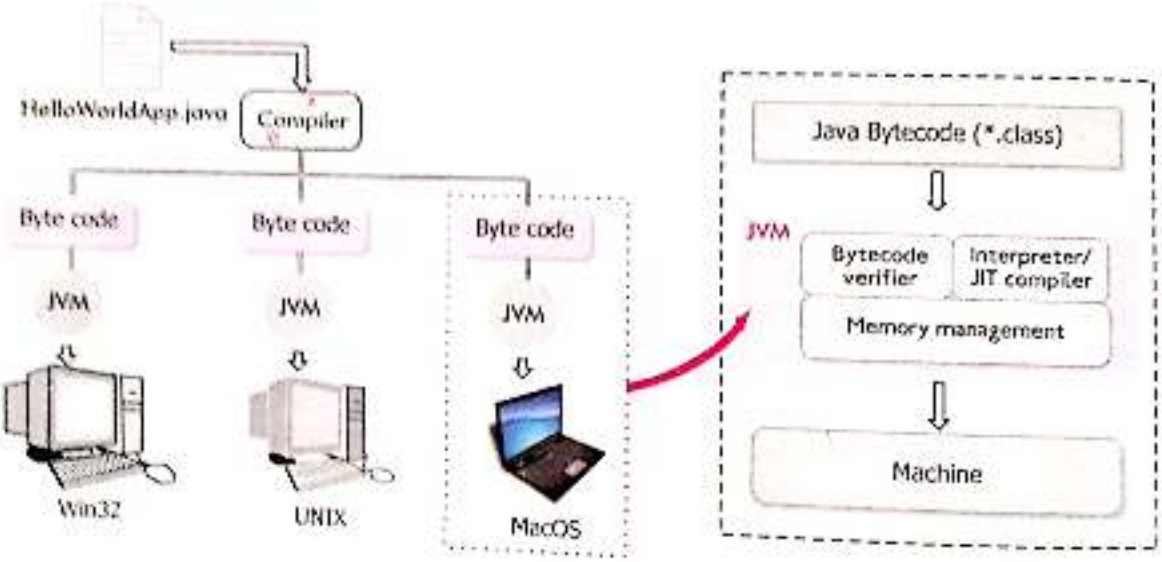


Figure 2.2 Java compilation

So we can summarize the Java compilation process as follows :

1. Java programs are written in ".java" file. (Source code) and then compiled by Java compiler.
2. **Bytecode.** Bytecode is the code generated after the java program is compiled. (.class file).
3. **Java Virtual Machine (JVM).** This is virtual machine which reads the bytecode and interprets into machine code depending upon the underlying operating system and hardware combination.
4. **Just In Time (JIT) compiler.** ***Just in time compiler*** is part of the Java Virtual Machine (JVM) and it compiles bytecode into executable code in real time, on a piece-by-piece, demand basis.

2.2.3 Characteristics of Java

Java has many characteristics that make it eligible for a powerful and popular language. In the following lines, we are going to discuss a few important characteristics of Java.

1. **Write Once Run Anywhere (WORA).** The Java programs need to be written just once, which can be run on different platforms without making changes in the Java program.
2. **Light Weight Code.** With Java, no huge coding is required.
3. **Security.** Java offers many enhanced security features.
4. **Object-Oriented Language.** Java is object-oriented language, thereby, very near to real world.
5. **Platform Independent.** Java is essentially platform independent. Change of platform does not affect the original Java program/application.

After this, let us move on to the discussion of how to create programs in BlueJ Java environment. It is not necessary to have BlueJ always to create and run Java programs. You can also create Java program in a simple text editor like Notepad and compile and run it by issuing few commands.

But before you can run Java programs, you must be aware of various components of a Java program.

2.3 Simple Java Program

Let us now have a look at an example Java program. Following lines show a simple and short example-java-program – **HelloWorld.**

Comment (can spread over multiple lines)

```
/* program HelloWorld */
class Hello World                          <- class name
{
    public void SayHello( )                <- method name
    {                                      <- body of the method
        System.out.println("Hello World!!") ;
    }

}       // class definition ends here      <- single-line comment
```

Let us now examine parts of this sample program.

- **class *HelloWorld*** The line *class HelloWorld* means that a class is being defined. After the class name, the data members and member methods (or functions) of the class are defined within curly braces { }.

- Method *SayHello* — The *HelloWorld* class contains one method – the **SayHello** method. This is a member method or member-function of class *HelloWorld*.
- System.out.println("Hello World!!")
  It prints *Hello World!!* on the standard output device which is generally your monitor.
- Comments /* ..... */ } and // ...
  The text enclosed within /* ..... */ called **comments**, is purely for enhancing readability. The comments are ignored by the compiler and not executed at all even if you write a valid Java statement within /* ..... */. Within /* ..... */ multi-line comments can be inserted but with // .... only single line comment can be inserted.

2.3.1 Types of Java Programs

There are *two* types of Java programs, Internet applets and stand-alone applications.

Internet Applets are small programs that are embedded in web pages and are run on the viewer's machine in a secured manner (which means limited access to system resources *i.e.*, the hard disk) by Java capable browsers.

Applets are designed to be delivered to Internet Web browsers and that is why an applet has a built-in graphical window. But Java applets have some security restrictions (*e.g.*, it cannot write to a local file).

The *second* type of Java program, **standalone application**, is much more interesting. It is generally a software application that does not require low level operating system or hardware access. This includes most off the usual desktop applications such as word processors or spreadsheets. Standalone Java applications could be distributed and installed on any Java capable machine. Every stand alone application of Java begins executing with a *main* method.

After learning about various parts of a Java program, let us now learn how to write a Java program on BlueJ environment. But before that let us know a little bit about "What is BlueJ ?"

2.4 BlueJ — A Quick Introduction

BlueJ is a Java™ development environment specifically designed for teaching at an introductory level. It was designed and implemented by the *BlueJ* team at *Monash University, Melbourne, Australia*, and the *University of Southern Denmark, Odense*.

BlueJ is basicafly an IDE (*Integrated Development Environment*). It includes the following tools in it :

- *an editor*, which you can use to write your programs.
- *a debugger*, which helps you find your mistakes.
- *a viewer*, which lets you see parts of your program graphically.

Apart from the above mentioned tools, BlueJ offers an easy way to run Java programs and to view documentation of your program/application.

2.5 Starting BlueJ

On Windows and MacOS, you need to run a program namely *BlueJ* that is installed when you install BlueJ software. We are running BlueJ on a Windows 7 platform, thus to start it we clicked at Start → All Programs → BlueJ → BlueJ (Fig. 2.3).

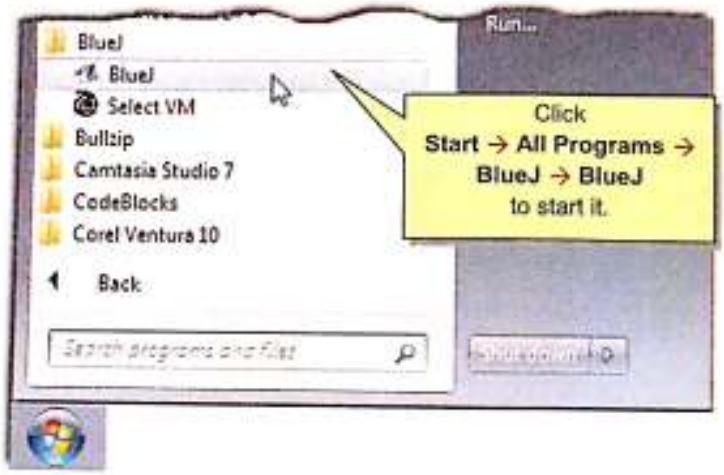


Figure 2.3 Starting BlueJ.

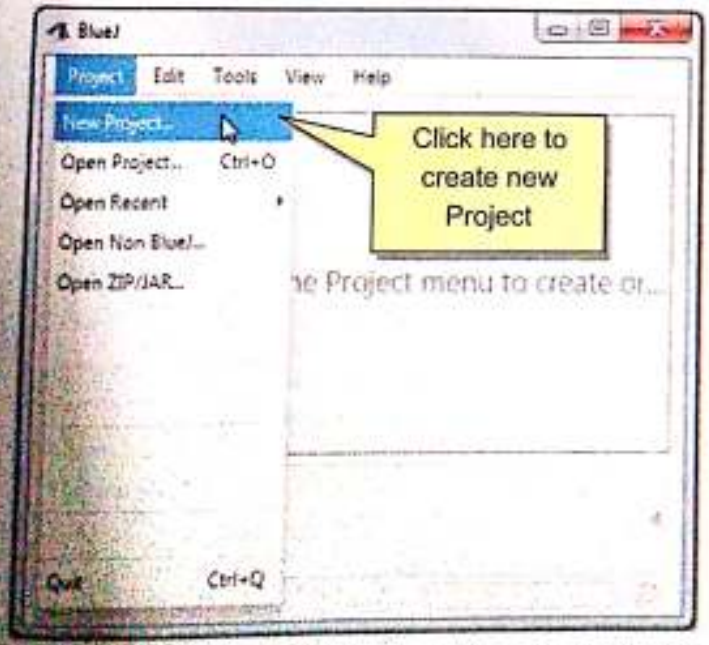
2.6 Writing Programs on BlueJ Environment

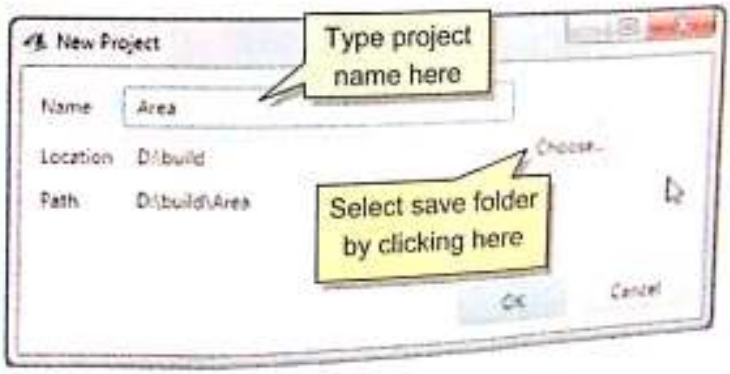
Once you start BlueJ, if you want to write a Java program, you need to create a *BlueJ project*. This is because all Java programs are created as part of a BlueJ project. Let us now see how you can create a new project.

2.6.1 Creating a BlueJ Project

To create a new project, you need to follow these steps :

1. Select **New Project** from **Project** menu of BlueJ IDE. [Fig. 2.4(*a*)].
2. Now a file selection dialog box-**New Project dialog** pops up. Here you need to specify the name and location for the new project [Fig. 2.4(*b*)].





(*b*) New Project dialog

3. Once you have selected the location and typed the name for new project, click **Create** [Fig. 2.4(*b*)].
4. The moment you click **Create**, a project folder gets created and the main BlueJ window shows the new empty project [Fig. 2.4(*c*)]. The icon you can see in the new project window is for a "readme" file where you can add some notes or additional documentation about your project. Just double click this icon and editor window will open wherein you can add desired notes/documentation. Click **Close** to close this editor window.
5. To add code, you need to add class(es) to your project as explained in coming section.

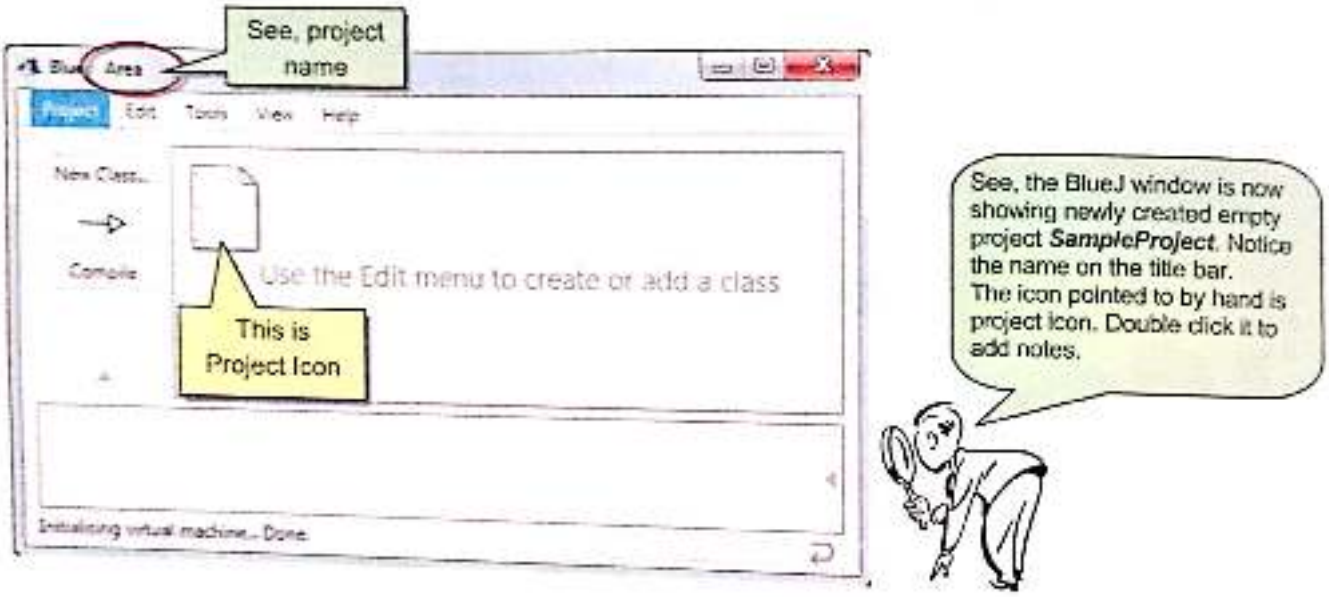


Figure 2.4 (*c*) Project window

2.6.2 Adding a New Class to Your Project

As all the functionality is written inside classes, you will need to add classes to your project. To add a new class to your project, follow these steps :

1. While in your project window (*i.e.*, if your project is not open, then open it by following *Project → Open Project* command), either click at **New Class** button on Project toolbar [Fig. 2.5(*a*)] or click **New Class** command on **Edit** menu [Fig. 2.5(*b*)].

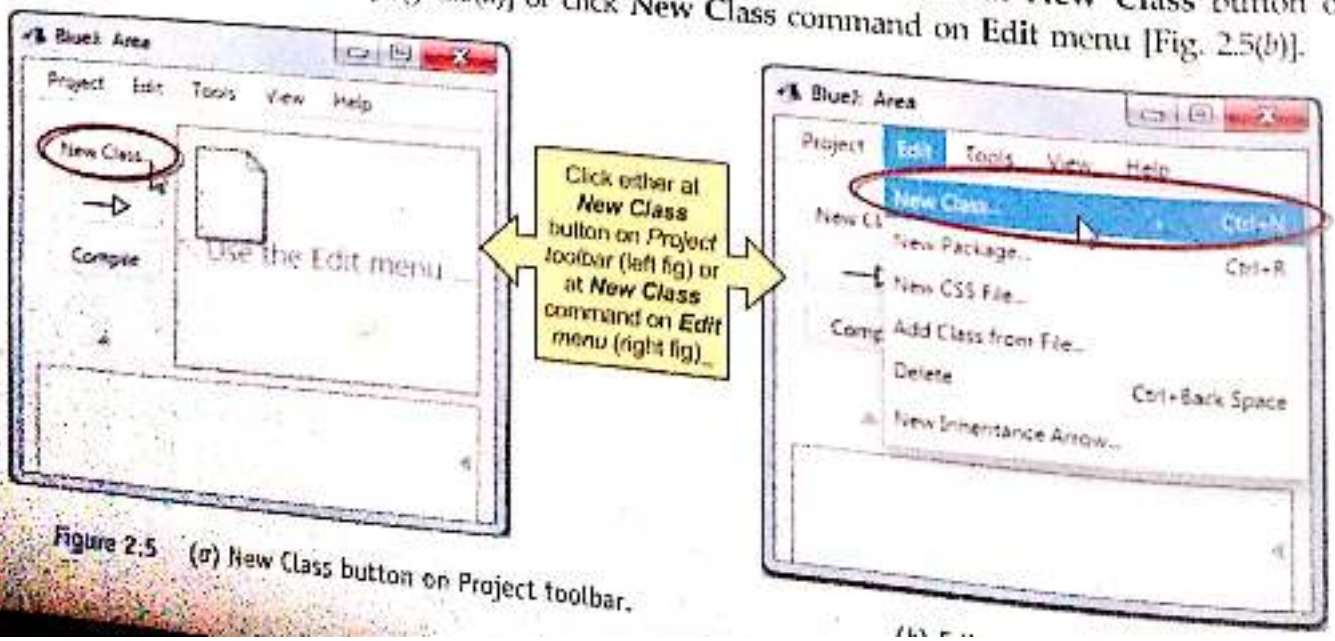


Figure 2.5 (*a*) New Class button on Project toolbar.

2. Now **Create New Class** dialog will pop up [Fig. 2.5(*c*)]. Here you'll be asked to specify a name for the class.

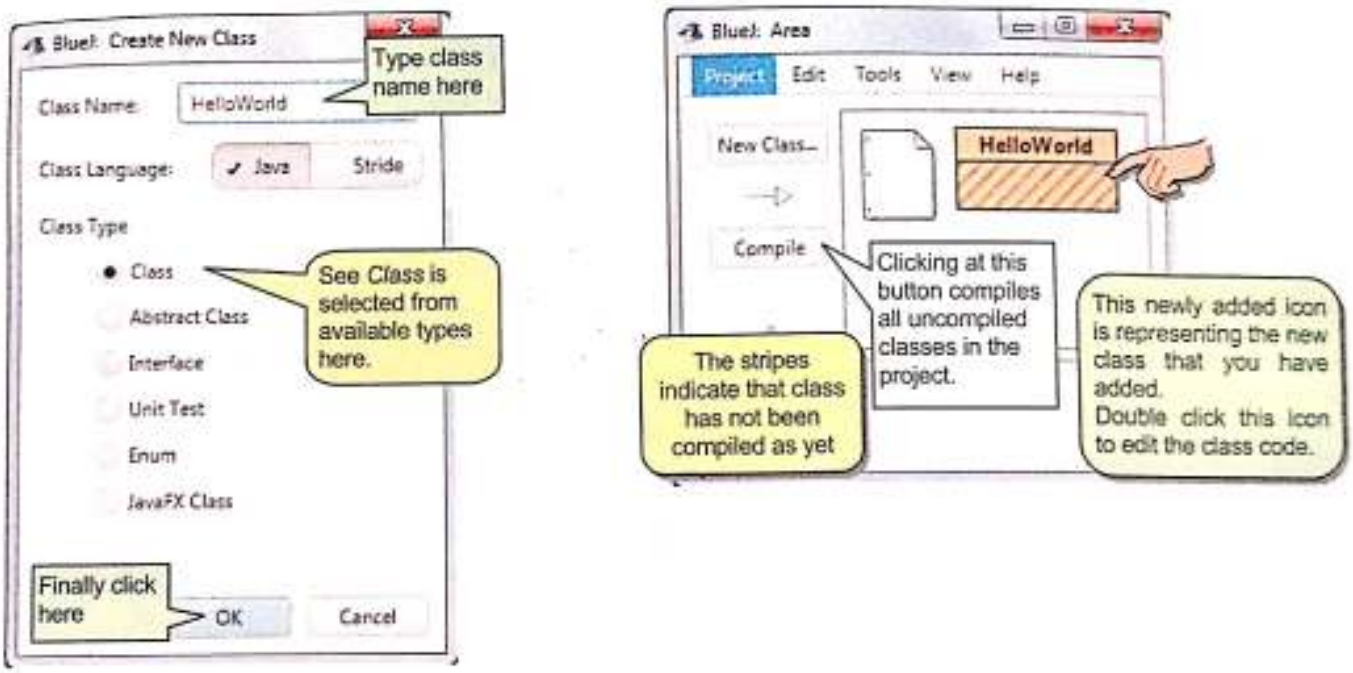


Figure 2.5 (*c*) Create new class dialog. (*d*) Class icon in the project window.

3. You can see that now a new icon gets added to your project window [Fig. 2.5(*d*)]. This new icon is representing the newly added class. As you can see that there are stripes on the class icon, the stripes mean that class has not been compiled as yet.

2.6.3 Editing a Class Code

To edit or write the code for a class, just double click it. It will open up the editor window showing class' skeleton code. Here make desired changes and then compile the class. BlueJ provides you with a sample class. But surely you can modify what BlueJ gives you, or you may simply replace it with your own program.

To enter your source code you need to open the editor for that class either by double-clicking.

Or

Right click on the class icon and select **Open Editor** (Fig. 2.6). An editor window will open for the class wherein you can make desired changes. (Fig. 2.7)

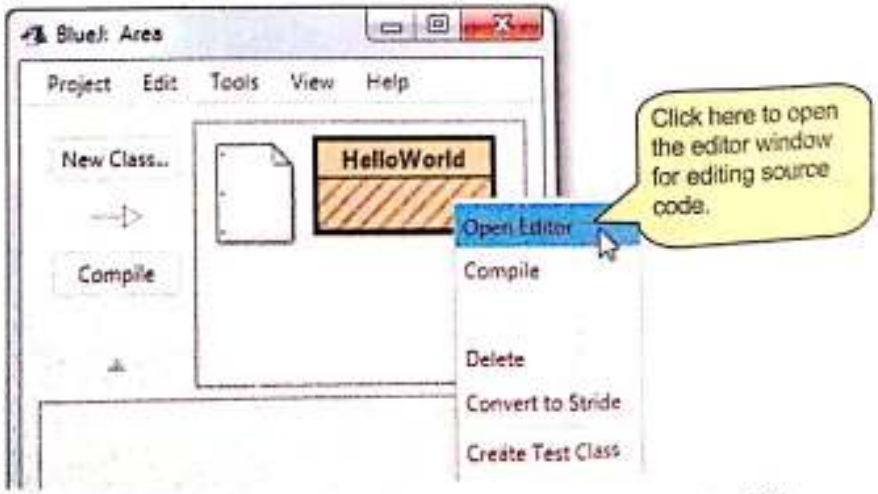


Figure 2.6 Open editor command on class' shortcut menu.

2.6.4 Compiling the Source Code in BlueJ

To compile the skeleton code, right click on the class icon and select **Compile**. Notice that the stripes on the class icon disappear, indicating that the class has been compiled.

If you have opened your class' source-code in editor window, then you can compile your class by directly clicking at **Compile** button on the top of editor window (see Fig. 2.7).

NOTE

In BlueJ, compilation automatically saves the source code on the disk.

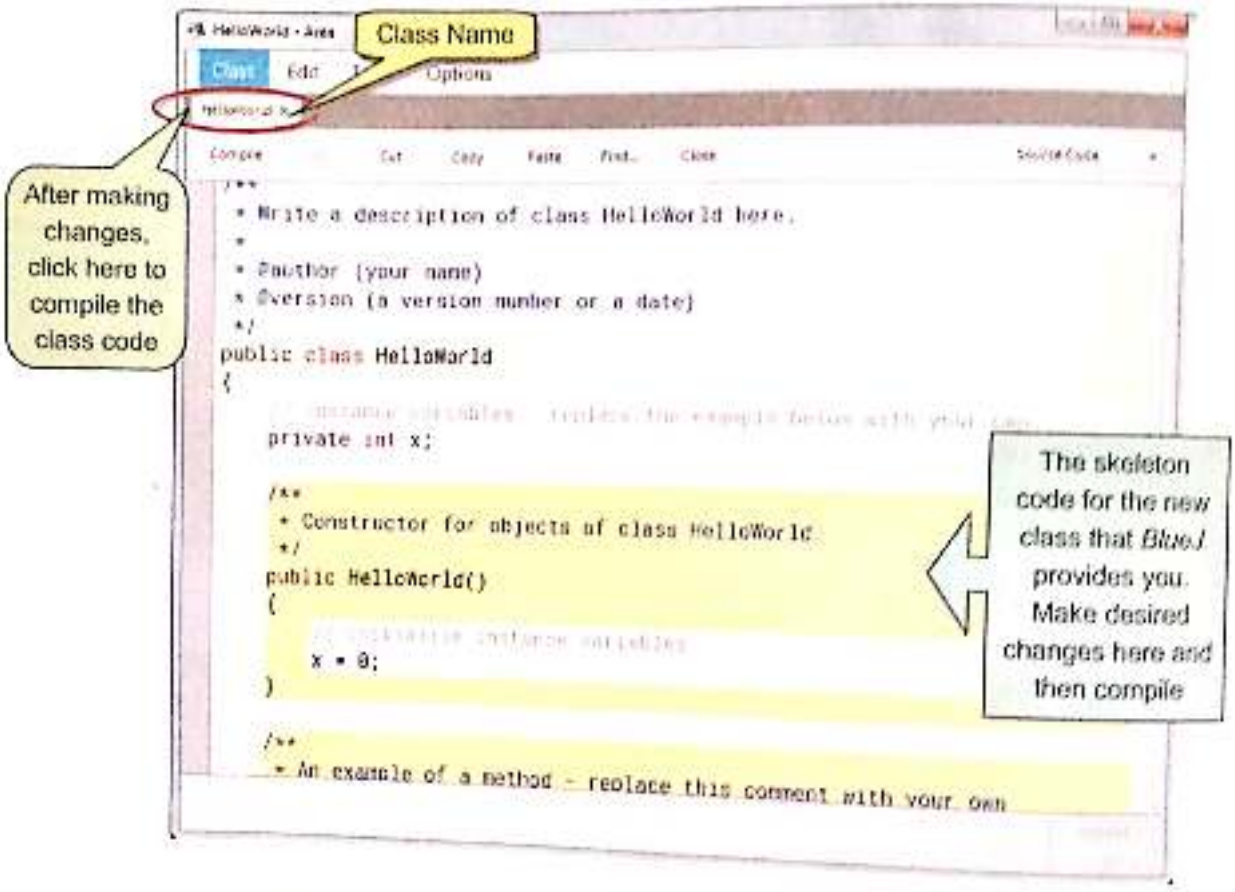


Figure 2.7 Editor window sharing class' default skeleton code.

2.6.5 Saving Your Code

Generally, there is no need to explicitly save the class source-code. This is because *Sources*[2] get automatically saved whenever it is appropriate (*e.g.*, when the editor is closed or before a class is compiled).

However, you can explicitly save if you like by using the **Save** option in the editor's Class menu (Fig. 2.8) *i.e.*, you need to open your class source in editor window for this.

2. I can safely call a source-code as a Source.

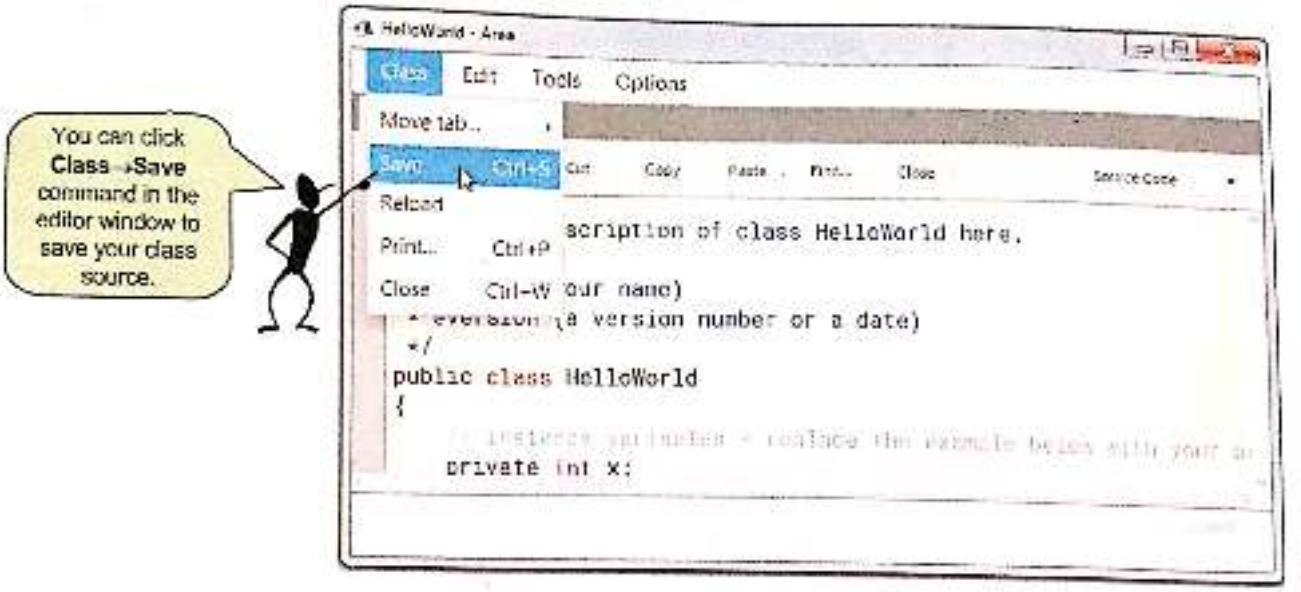


Figure 2.8 Class → Save command in Editor window.

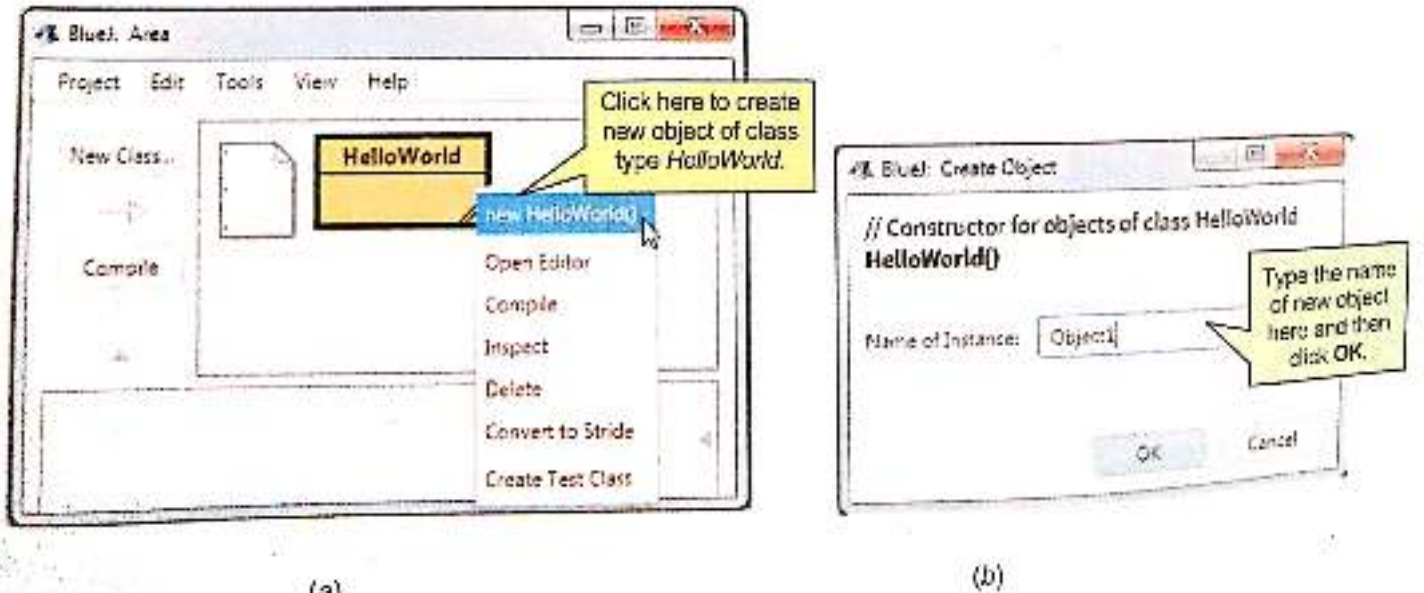
2.6.6 Creating Objects in BlueJ

The ease offered by BlueJ is so much that it even allows you to test classes individually as soon as they have been written. There is no need to write the complete application first.

Execution in BlueJ usually done by creating an object first and then by invoking any of object's (public) methods.

To create object of a class, firstly compile the class and then follow these steps :

1. Right click on the class and from the shortcut menu select *new <classname>( ) e.g.,* from the shortcut menu of class *HelloWorld,* you'll select **newHelloWorld( )** [Fig. 2.9(*a*)].



(*a*) (*b*)

Figure 2.9 (*a*) Creating Objects. (*b*) Create object dialog.

2. Now create object dialog will appear. Here specify the name for new object and click **OK** [Fig. 2.9(*b*)].

3. Once you click OK in Create Object dialog, BlueJ will create an object with that name and show its icon in the **Object bench** [Fig. 2.9(*c*)].

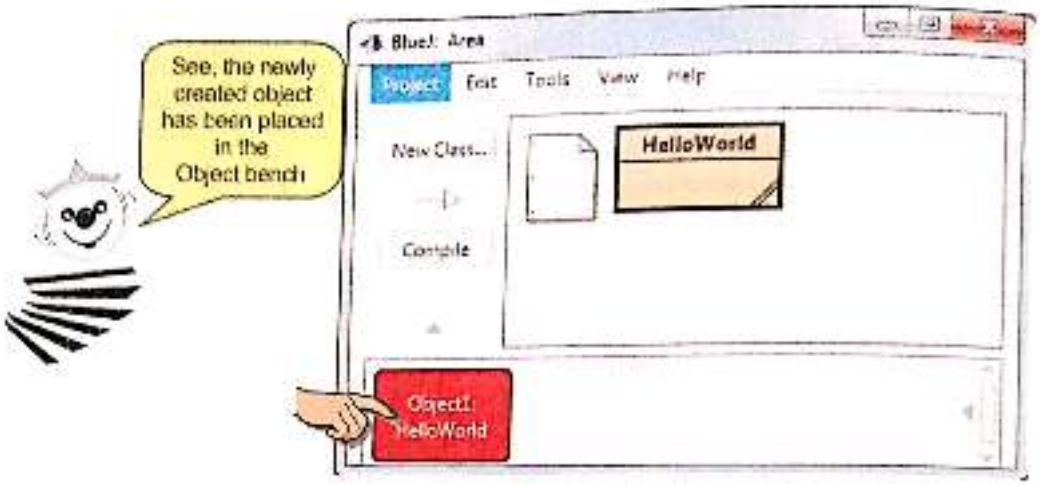


Figure 2.9 (*c*) Created object on object bench.

2.6.7 Executing an Object's Method

To learn to do this, let us edit our class and add following method to it :

```
public void display( ) {
    System.out.println("Hi ! BlueJ is very easy to work with.") ;
}
```

Now your HelloWorld class should look like :

```
public class HelloWorld  {
    public void sayHello( ) {
        System.out.println("Hello World!!") ;
    }
    public void display( )  {
        System.out.println("Hi! BlueJ is very easy to work with.") ;
    }
}
```

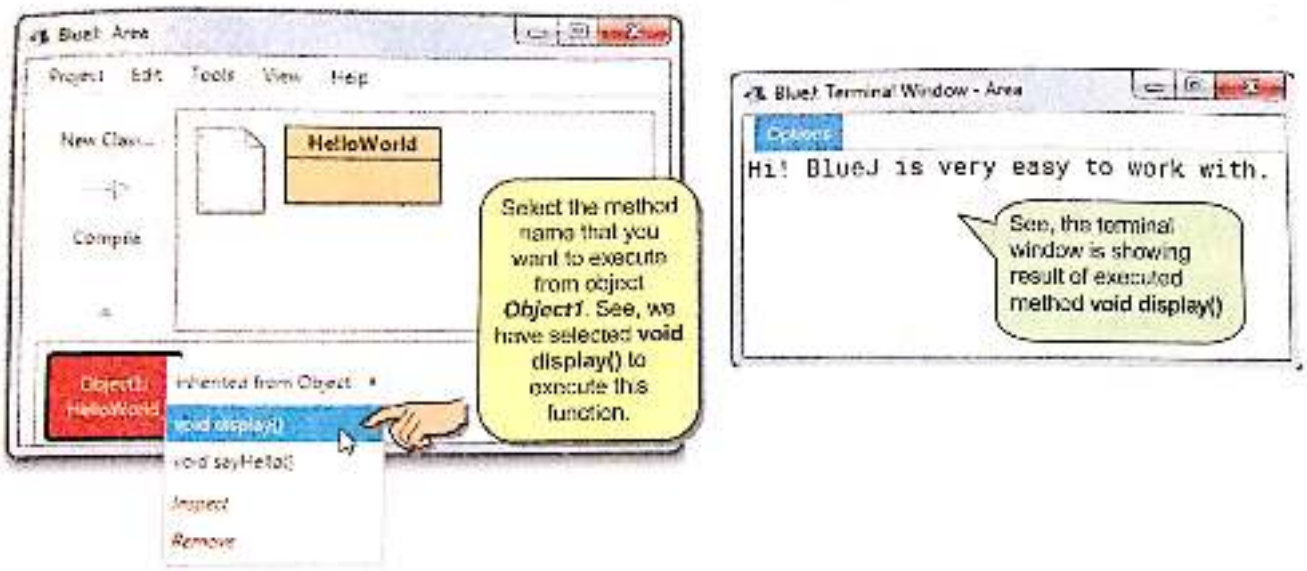
Compile this class and create its object as *object1*. Now to execute the method **display( )** of object *object1*, follow these steps :

1. Right click object *Object1's* icon and select the name of the method to be executed. We shall select **display**. [Fig. 2.10(*a*)].

> **NOTE**
> A method is also known as *member-function*.

2. BlueJ will ask for values if the method being executed requires some values, the parameters (You'll learn about parameters and methods later).

Otherwise, it will simply execute the function and display its result in the *Terminal window* [Fig. 2.10(*b*)] as it did in our case.



(*a*) (*b*)

Figure 2.10

2.6.8 Saving Output from the Console Window

Output from the console window can be saved as a text file by selecting **Options → Save to file** from the menu bar in terminal window. This text file can then be printed from a text editor like *Notepad*.

NOTE

Selecting **Project → Print** from the menu bar in the project window will print the contents of the project window. It will not print your source code for the project.

2.6.9 Printing Your Source Code

To print the source code from a class, open the edit window for that class and select **Class → Print** from the menu bar.

2.6.10 Closing a Project

To close your project, select **Project → Close** from the menu bar in the project window.

2.6.11 Opening an Existing BlueJ Project

To open an existing project select **Project → Open** from the menu bar in the project window. An open project dialog box will open. Navigate to the project name and click **Open**.

NOTE

While opening non-BlueJ files, the source files (package) must be in one folder.

2.7 The Method main( )

One more thing that you should be aware of is about method main(). As you have seen that in BlueJ, you can execute or run any method of the class by creating an object and then right clicking and selecting the desired method to be executed. But if you want to run your Java

program outside BlueJ then your programs may compile but would not run unless you add a method with following syntax to your class :

```
public static void main( String[ ] args)
{   // execution code here
}
```

OR

```
public static void main( String args[ ])
{       // execution code here
}
```

Please note all these keywords are in lowercase except for *String*.

Also recall that a Java program can become a standalone application only if it has method **main( )**.

In a Java stand-alone program, the execution begins from *main( )*. Therefore, to execute any other method, you need to execute it through *main( )*. A program having main( ) method can be executed by selecting **void main(String[ ] args)** from the class's menu that appears when you right click on the class's icon. (see Fig. 2.11)

NOTE

Java is case sensitive i.e., small letters are treated differently from Upper-case letters. That means main() is not same as Main() or MAIN().

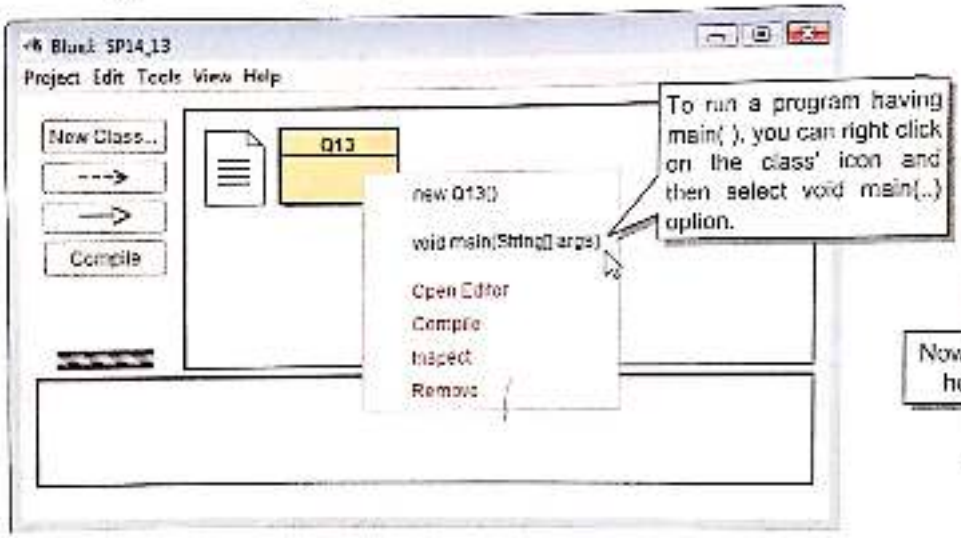


Figure 2.11 To run a program having main, click on **void main(..)** .

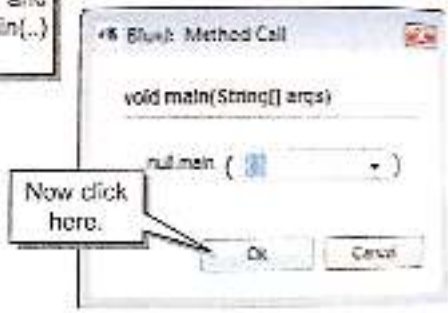


Figure 2.12 Click OK in main's method call dialog.

It will then display a dialog where simply click on OK, if no value is to be passed to main( ). (see Fig. 2.12). Passing values to main( ) is an advanced topic, which we are not covering here.

LET US REVISE

- *Java is a popular object-oriented programming language.*
- *Java was originally named as Oak.*
- *Java byte code is a machine instruction for a Java processor chip called Java virtual machine.*
- *The Java byte code is independent of the computer system it has to run upon.*
- *The byte codes are always exactly the same irrespective of the computer system they are to execute upon.*
- *Java programs are compiled into Java bytecode.*
- *Java virtual machine is an abstract machine which is implemented on the top of existing processors.*
- *In a Java program there can be many classes but only one initial class.*
- *There are two types of Java applications — Internet applet and stand alone application.*
- *An Internet applet runs within a web browser.*
- *A standalone application can run on any platform.*
- *BlueJ is a Java development environment specifically designed for teaching at an introducing level.*
- *BlueJ is an IDE (Integrated Development Environment) to create Java applications.*

Conceptual Questions

Section A : Objective Type Questions

1. How would you describe Java ?
   (*a*) a programming language (*b*) a platform
   (*c*) both (*a*) and (*b*) (*d*) none of the above
2. Java applications are
   (*a*) very big (*b*) very small
   (*c*) platform independent (*d*) platform dependent
3. Which of the following is the reason of Java programs' platform independence ?
   (*a*) Java compiler (*b*) Java virtual machine
   (*c*) Java byte code (*d*) all the above
4. Bytecodes are always _____ for different platforms.
5. The machine language for the JVM is called _____ .
6. An abstract machine that hides underlying operating systems for Java applications and executes Java byte code is
   (*a*) Java machine (*b*) Java compiler
   (*c*) Java platform (*d*) Java Virtual Machine.
7. JVM combined with _____ makes Java platform.
   (*a*) Java Programs (*b*) Java API
   (*c*) Java Byte code (*d*) all the above
8. A class in Java program, that contains *main* method, is called _____ .
9. A Java program must have name similar to that of _____ and *.java* extension.
10. Which of the following prints some text on the monitor (in context of Java) ?
    (*a*) print (*b*) println
    (*c*) System.out.println (" ... ") (*d*) all the above.
11. Which of the following is not a valid comment ?
    (*a*) /** comment */ (*b*) /* comment */
    (*c*) /* comment (*d*) // comment

Section B : Subjective Type Questions

1. *When you compile a program written in the Java programming language, the compiler converts the human readable source file into platform independent code that a JVM can understand ? What is this platform independent code called ?*

   Ans. Byte code.

2. *How can you say that Java is both a programming language and a platform ?*

Ans. Like any other *programming language*, we can use Java to write or create various types of computer applications. The word *platform* generally is used to refer to some combination of hardware and system software. The Java platform is a new software platform designed to deliver and run highly interactive, dynamic and secure applications on networked computer systems.

3. *How is ordinary compilation process different from Java compilation ?*

Ans. In ordinary compilation, the source code is converted to a machine code, which is dependent upon the machine or the platform. This resultant machine code is called *native executable code*.

Contrary to ordinary compilers, the Java compiler does not produce *native executable code* for a particular machine. Instead it produces a special format called ***bytecode***. The **Java bytecode** looks a lot like machine language, but unlike machine language *Java byte code is exactly the same on every platform*.

4. *What do you understand by JVM ?*

Ans. The *Java Virtual Machine* is an abstract machine designed to be implemented on the top of existing processors. It hides the underlying operating system from Java applications. Programs written in Java are compiled into *Java Byte code*, which is then interpreted by a special *Java Interpreter* for a specific platform. Actually this *Java interpreter* is known as the **Java Virtual Machine (JVM)**.

5. *What is 'Write Once Run Anywhere' characteristic of Java ?*

Ans. The Java programs need to be written just once, which can be run on different platforms without making changes in the Java program. Only the Java interpreter is changed depending upon the platform. This characteristic is known as *Write Once Run Anywhere*.

6. *What will be the result produced by following Java program ?*

```
class MyClass {
    public void MyClassTest( )
    {
        System.out.println ("Printed from inside of") ;
        System.out.println ("MyClass") ;
    }
}
```

Ans.
```
Printed from inside of
MyClass
```

7. *Change the program of question 6, so that it prints :*

```
Printed from inside of MyClassRoom
Thank You.
```

Ans.
```
class Myclass
{   public void MyClassTest( )
    {
        System.out.println ("Printed from inside of MyClassRoom") ;
        System.out.println ("Thank You") ;
    }
}
```

8. *What do you know about BlueJ ?*

Ans. BlueJ is a Java development environment. It is an IDE (Integrated Development Environment), which includes an editor, a debugger and a viewer.

It offers an easy way to run Java programs and view documentation.

9. *Name two types of Java programs.*

Ans. The two types of Java applications are *Internet applets* and *Stand alone applications*.

KEYWORDS

Bytecode A machine instruction for a Java processor chip called Java Virtual Machine (JVM).

Java Virtual Machine (JVM) An abstract machine that hides underlying operating system from Java applications and executes the Java byte code.

Assignment

1. Why is Java often termed as a platform ?
2. What is bytecode ?
3. How is Java platform independent ?
4. What is JVM ?
5. Describe ordinary compilation process.
6. Describe Java compilation.
7. How is Java bytecode executed ?
8. What are Java APIs ?
9. Write a simple Java program that prints your name. Compile and run it on BlueJ environment.
10. Add comments to your previous program. Once again compile and run it.
11. Java is both a programming language and a platform. Comment.
12. Briefly tell the history of Java.
13. How are bytecode and platform independence interlinked ?
14. What is the role of JVM in platform independence ?
15. What are different characteristics of Java ?
16. Create a Java program in BlueJ having a function *SayNa( )* that prints

    JAVA IS EASY TO LEARN WITH BLUEJ

17. Now compile and run the program you created just now.
18. Add another class (given below) in the same program.

```
public class Two {
    public static void display()  {
        System.out.println ("Hi There !!") ;
        System.out.println ("I am in class Two") ;
    }
}
```

19. Now add following code in the method *SayNa( )* before statement

```
System.out.println("JAVA IS EASY TO LEARN WITH BLUEJ") ;
        Two.display() ;
```

Recompile and run your program. See what happens.

20. Now create another program with following class in it :

```
class Three {
    public static void check()
    {
        System.out.println ("I am in class three") ;
        Two.display() ;
    }
}
```

Compile and run it. See what happens.

(Notice that static method of a class can be invoked directly using a class name, without any objec