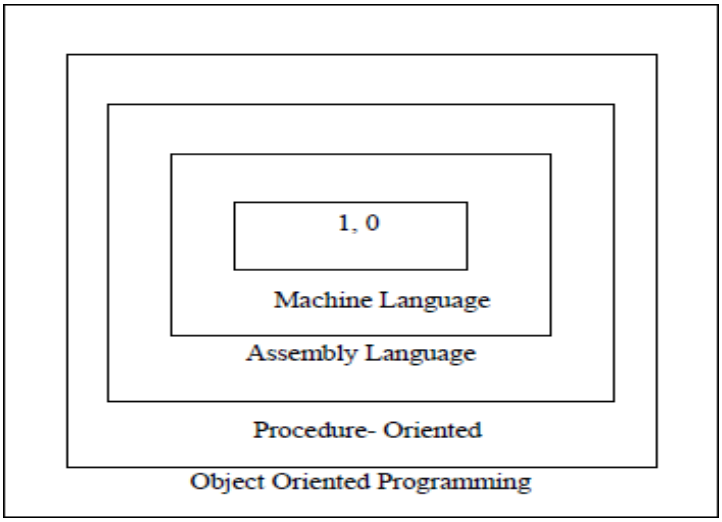
M. Sc. ST 1st Sem

PH533.1 Object Oriented Programming with Java

Software Evolution:

- Ernest Tello, A well-known writer in the field of artificial intelligence, compared the evolution of software technology to the growth of the tree.
- Like a tree, the software evolution has had distinct phases "layers" of growth.
- These layers were building up one by one over the last five decades as shown in fig. 1.1, with each layer representing and improvement over the previous one.



- However, the analogy fails if we consider the life of these layers.
- In software system each of the layers continues to be functional, whereas in the case of trees, only the uppermost layer is functional.
- Alan Kay, one of the promoters of the object-oriented paradigm and the principal designer of Smalltalk, has said: "As complexity increases, architecture dominates the basic materials".
  - To build today's complex software it is just not enough to put together a sequence of programming statements and sets of procedures and modules.
  - We need to incorporate sound construction techniques and program structures that are easy to comprehend implement and modify.
- With the advent of languages such as c, structured programming became very popular and was the main technique of the 1980's.
- Structured programming was a powerful tool that enabled programmers to write moderately complex programs fairly easily.

- However, as the programs grew larger, even the structured approach failed to show the desired result in terms of bug-free, easy-to-maintain, and reusable programs.
- Object Oriented Programming (OOP) is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several powerful new concepts.
- It is a new way of organizing and developing programs and has nothing to do with any particular language.

Procedure-Oriented Programming:

- In the procedure-oriented approach, the problem is viewed as the sequence of things to be done such as reading, calculating and printing such as COBOL, FORTRAN and C.
- The primary focus is on functions. A typical structure for procedural programming is shown in fig.1.2.

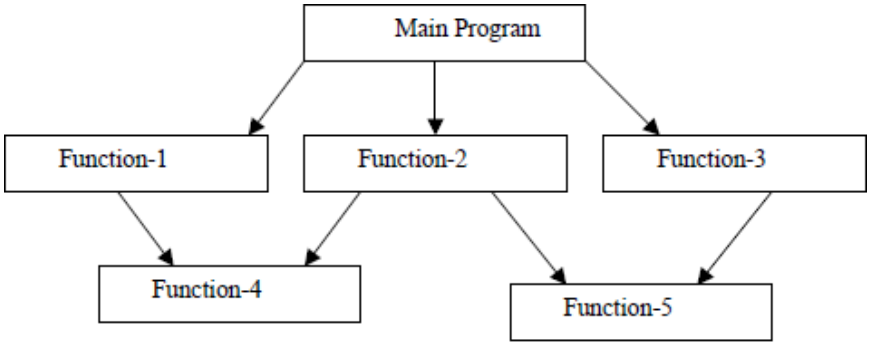


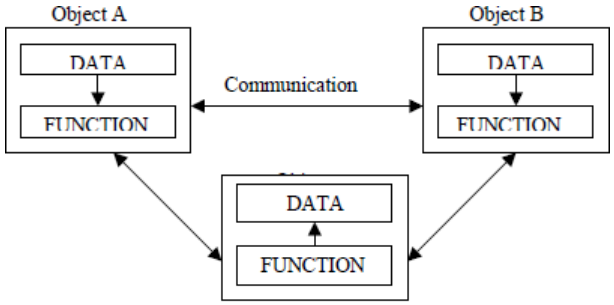
Fig. 1.2 Typical structure of procedural oriented programs

- The technique of hierarchical decomposition has been used to specify the tasks to be completed for solving a problem.
- Procedure oriented programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions.
- We normally use flowcharts to organize these actions and represent the flow of control from one action to another.
- In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions.
- Each function may have its own local data. Global data are more vulnerable to an inadvertent change by a function.
- In a large program it is very difficult to identify what data is used by which function.

- In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.
- Another serious drawback with the procedural approach is that we do not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the element of the problem.
- Some Characteristics exhibited by procedure-oriented programming are:
  - Emphasis is on doing things (algorithms).
  - Large programs are divided into smaller programs known as functions.
  - Most of the functions share global data.
  - Data move openly around the system from function to function.
  - Functions transform data from one form to another.
  - Employs top-down approach in program design.

Object Oriented Paradigm:

- The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach.
- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the function that operate on it, and protects it from accidental modification from outside function.
- OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects.
- The organization of data and function in object-oriented programs is shown in the following figure.



Organization of data and function in OOP

- The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of other objects.

- Some of the features of object oriented programming are:
  - Emphasis is on data rather than procedure.
  - Programs are divided into what are known as objects.
  - Data structures are designed such that they characterize the objects (class).
  - Functions that operate on the data of an object are ties together in the data structure.
  - Data is hidden and cannot be accessed by external function.
  - Objects may communicate with each other through function.
  - New data and functions can be easily added whenever necessary.
  - Follows bottom-up approach in program design.

Basic Concepts of Object Oriented Programming:

- It is necessary to understand some of the concepts used extensively in object-oriented programming. These include:
  - Objects
  - Classes
  - Data abstraction and encapsulation
  - Inheritance
  - Polymorphism
  - Dynamic binding
  - Message passing

Objects:

- Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.
- They may also represent user-defined data such as vectors, time and lists. Programming problem is analyzed in term of objects and the nature of communication between them.
- Program objects should be chosen such that they match closely with the real-world objects.
- Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in c.
- When a program is executed, the objects interact by sending messages to one another.
- For example, if “customer” and “account” are to object in a program,

then the customer object may send a message to the count object requesting for the bank balance.

- Each object contain data, and code to manipulate data. Objects can interact without having to know details of each other's data or code.
- It is a sufficient to know the type of message accepted, and the type of response returned by the objects.
- Although different author represents them differently fig 1.5 shows notation that are popularly used in object-oriented analysis and design.

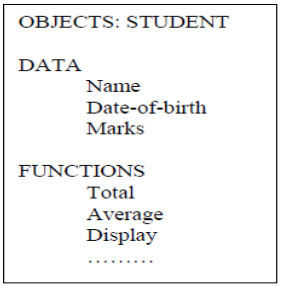


Fig. 1.5 representing an object

❖ **Classes:**

- We just mentioned that objects contain data, and code to manipulate that data.
- The entire set of data and code of an object can be made a user-defined data type with the help of class. In fact, objects are variables of the type class.
- Once a class has been defined, we can create any number of objects belonging to that class.
- Each object is associated with the data of type class with which they are created.
- A class is thus a collection of objects similar types. For e.g., Mango, Apple and orange members of class fruit.
- Classes are user-defined that types and behave like the built-in types of a programming language.
- The syntax used to create an object is not different then the syntax used to create an integer object in C.
- If fruit has been defines as a class, then the statement

```
Fruit Mango;  //creates an object Mango of class Fruit
```

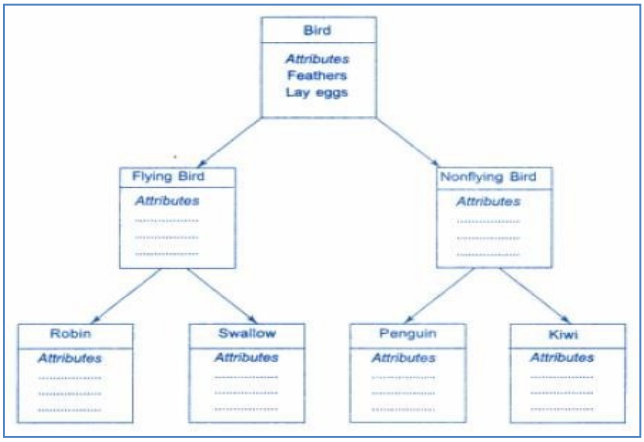
❖ **Data Abstraction and Encapsulation:**

- The wrapping up of data and function into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class.

- The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.
- These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called data hiding or information hiding.
- Abstraction refers to the act of representing essential features without including the background details or explanation.
- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight, and cost, and function operate on these attributes.
- They encapsulate all the essential properties of the object that are to be created.
- The attributes are sometime called data members because they hold information.
- The functions that operate on these data are sometimes called methods or member function.

❖ **Inheritance:**

- Inheritance is the process by which objects of one class acquired the properties of objects of another classes.
- It supports the concept of hierarchical classification.
- For e.g., the bird, 'robin' is a part of class 'flying bird' which is again a part of the class 'bird'.
- The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated below…

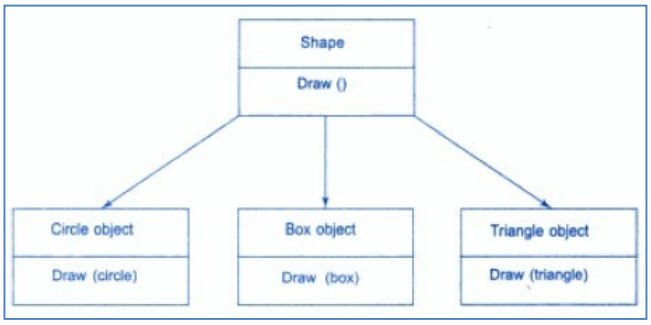


- In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it.
- This is possible by deriving a new class from the existing one. The

new class will have the combined feature of both the classes.

❖ **Polymorphism:**

- Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than on form.
- An operation may exhibit different behavior is different instances, whose behavior depends upon the types of data used in the operation.
- For example, consider the operation of addition.
  - For two numbers, the operation will generate a sum.
  - If the operands are strings, then the operation would produce a third string by concatenation.
- The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.
- The following figure illustrates that a single function name can be used to handle different number and different types of argument.
- This is something similar to a particular word having several different meanings depending upon the context.
- Using a single function name to perform different type of tasks is known as function overloading.



- Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface.
- This means that a general class of operations may be accessed in the same manner even though specific action associated with each operation may differ.
- Polymorphism is extensively used in implementing inheritance.

❖ **Dynamic Binding:**

- Binding refers to the linking of a procedure call to the code to be executed in response to the call.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time.

- It is associated with polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that reference.
- Consider the procedure "draw" in previous figure by inheritance, every object will have this procedure.
- Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object.
- At run-time, the code matching the object under current reference will be called.

❖ **Message Passing:**

- An object-oriented program consists of a set of objects that communicate with each other.
- The process of programming in an object-oriented language, involves the following basic steps…
    - Creating classes that define object and their behavior.
    - Creating objects from class definitions, and
    - Establishing communication among objects.
- Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another.
- The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts.
- A Message for an object is a request for execution of a procedure, and therefore will invoke a function in the receiving object that generates the desired results.
- Message passing involves specifying the name of object, the name of the function (message) and the information to be sent. Example…

Benefits of OOP:

- OOP offers several benefits to both the program designer and the user.
- Object-Orientation contributes to the solution of many problems associated with the development and quality of software products.
- The new technology promises greater programmer productivity, better

quality of software and lesser maintenance cost. The principal advantages are……

- Through inheritance, we can eliminate redundant code extend the use of existing.
- Classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure program that can not be invaded by code in other parts of a programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more detail of a model can implemental form.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

- While it is possible to incorporate all these features in an object-oriented system, their importance depends on the type of the project and the preference of the programmer.

Object Oriented Language:

- Object-oriented programming is not the right of any particular languages.
- Like structured programming, OOP concepts can be implemented using languages such as C and Pascal. However, programming becomes clumsy and may generate confusion when the programs grow large.
- A language that is specially is designed to support the OOP concepts makes it easier to implement them.
- The languages should support several of the OOP concepts to claim that they are object-oriented.
- Depending upon the features they support, they can be classified into the following two categories:

- Object-based programming languages, and
- Object-oriented programming languages.

- Object-based programming is the style of programming that primarily supports encapsulation and object identity.
- Major feature that are required for object-based programming are…
  - Data encapsulation, Data hiding and access mechanisms.
  - Automatic initialization and clear-up of objects, and Operator overloading.
- Languages that support programming with objects are said to the objects-based programming languages. They do not support inheritance and dynamic binding.
- Ada is a typical object-based programming language.
- Object-oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding.

Applications of OOP:

- The promising areas of application of OOP include:
- Real-time system.
- Simulation and modeling.
- Object-oriented data bases.
- Hypertext, Hypermedia, and expertext.
- AI and expert systems.
- Neural networks and parallel programming.
- Decision support and office automation systems.
- CIM/CAM/CAD systems.

History of Java:

- Java history is interesting to know. The history of java starts from Green Team.
- Java team members (also known as Green Team), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.
- For the green team members, it was an advance concept at that time. But, it was suited for internet programming.

- Later, Java technology as incorporated by Netscape.
- Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc.
- There are given the major points that describe the history of java……
  - James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
  - Originally designed for small, embedded systems in electronic appliances like set-top boxes.
  - Firstly, it was called "Greentalk" by James Gosling and file extension was .gt
  - After that, it was called Oak and was developed as a part of the Green project.
  - In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
  - Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

Java Version History:

- There are many java versions that has been released. Current stable release of Java is Java SE 8.
  - JDK Alpha and Beta (1995), JDK 1.0 (23rd Jan, 1996),
  - JDK 1.1 (19th Feb, 1997), J2SE 1.2 (8th Dec, 1998),
  - J2SE 1.3 (8th May, 2000), J2SE 1.4 (6th Feb, 2002)
  - J2SE 5.0 (30th Sep, 2004), Java SE 6 (11th Dec, 2006),
  - Java SE 7 (28th July, 2011), Java SE 8 (18th March, 2014)

Java Features:

- The inventors of Java wanted to design a language which could offer solutions to some of the problems encountered in modern programming.
- They wanted the language to be not only reliable, portable and distributed but also simple, compact and interactive.
- Sun Microsystems officially describes Java with the following attributes……

Java 2 Features	*Additional Features of J2SE 5.0*
• Compiled and Interpreted • Platform-Independent and Portable • Object-Oriented • Robust and Secure • Distributed • Familiar, Simple and Small • Multithreaded and Interactive • High Performance • Dynamic and Extensible	• Ease of Development • Scalability and Performance • Monitoring and Manageability • Desktop Client • Core XML Support • Supplementary character support • JDBCRowSet

❖ **Compiled and Interpreted:**

- Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system.
- First, Java compiler translates source code into what is known as bytecode instructions.
- Bytecodes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program.
- We can thus say that Java is both a compiled and an interpreted language.

❖ **Platform-Independent and Portable:**

- The most significant contribution of Java over other languages is its portability.
- Java programs can be easily moved from one computer system to another, anywhere and anytime.
- Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs.
- This is the reason why Java has become a popular language for programming on Internet.
- Java ensures portability in two ways.
  - First, Java compiler generates bytecode instructions that can be implemented on any machine.
  - Secondly, the size of the primitive data types are machine-independent.

❖ **Object-Oriented:**

- Java is a true object-oriented language. Almost everything in Java is an object. All program code and data reside within objects and classes.
- Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by inheritance.

- The object model in Java is simple and easy to extend.

❖ **Robust and Secure:**

- Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types.
- It is designed as a garbage-collected language relieving the programmers virtually all memory management problems.
- Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system.
- Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources are everywhere.
- Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet.

❖ **Distributed:**

- Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs.
- Java applications can open and access remote objects on Internet as easily as they can do in a local system.
- This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

❖ **Simple, Small and Familiar:**

- Java is a small and simple language.
- Many features of C and C++ that are either redundant or sources of unreliable code are not part of Java.
- For example, Java does not use pointers, preprocessor header files, goto statement and many others.
- It also eliminates operator overloading and multiple inheritance.
- Familiarity is another striking feature of Java.
- To make the language look familiar to the existing programmers, it was modelled on C and C++ languages.
- Java uses many constructs of C and C++ and therefore, Java code "looks like a C++" code.
- In fact, Java is a simplified version of C++.

❖ **Multithreaded and Interactive:**

- Multithreaded means handling multiple tasks simultaneously.
- Java supports multithreaded programs, means that we need not wait for the application to finish one task before beginning another.

❖ **High Performance:**

- Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode.
- According to Sun, Java speed is comparable to the native C/C++. Java architecture is also designed to reduce overheads during runtime.
- Further, the incorporation of multi-reading enhances the overall execution speed of Java programs.

❖ **Dynamic and Extensible:**

- Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods, and objects.
- Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.
- Java programs support functions written in other languages such as C and C++, which are known as native methods.
- This facility enables the programmers to use the efficient functions available in these languages. Native methods are linked dynamically at runtime.

❖ **Ease of Development:**

- Java 2 Standard Edition (J2SE) 5.0 supports features, such as Generics, Enhanced for Loop, Autoboxing or unboxing, Typesafe Enums, Varargs, Static import and Annotation.
- These features reduce the work of the programmer by shifting the responsibility of creating the reusable code to the compiler.
- The resulting source code is free from bugs because the errors made by the compiler are less when compared to those made by programmers.
- Thus, each of the linguistic features is designed to develop Java programs in an easier way.

❖ **Scalability and Performance:**

- J2SE 5.0 assures a significant increase in scalability and performance by improving the startup time and reducing the amount of memory used in Java 2 runtime environment.

- For e.g., the introduction of the class, data sharing in the Hotspot Java Virtual Machine (JVM) improves the startup time by loading the core classes from the jar files into a shared archive.
- Memory utilization is reduced by sharing data in the shared archive among multiple JVM processes.
- In the earlier versions, the data was replicated in each JVM instance.

HOW JAVA DIFFERS FROM C AND C++ …?:

- Although Java was modelled after C and C++ languages, it differs from C and C++ in many ways.
- Java does not incorporate a number of features available in C and C++.

Java and C:

- Java is a lot like C but the major difference between Java and C is that Java is an object- oriented language and has mechanism to define classes and objects.
- The Java team did not include some of the C features in Java, such as……
- Java does not include the C unique statement keywords. sizeof, and typedef.
- Java does not contain the data types struct and union.
- Java does not define the type modifiers keywords auto, extern, register, signed, and unsigned.
- Java does not support an explicit pointer type.
- Java does not have a preprocessor and therefore we cannot use #define, #include, and #ifdef statements.
- Java requires that the functions with no arguments must be declared with empty parenthesis and not with the void keyword as done in C.

Java and C++:

- Java is a true object-oriented language while C++ is basically C with object-oriented extension (that's why C++).
- C++ has maintained backward compatibility with C. It is therefore possible to write an old style C program and run it successfully under C++.
- Listed below are some major C++ features that were intentionally omitted from Java or significantly modified.
  - Java does not support operator overloading.

- Java does not have template classes as in C++.
- Java does not support multiple inheritance of classes. This is accomplished using a new feature called "interface".
- Java does not support global variables. Every variable and method is declared within a class and forms part of that class.
- Java does not use pointers and Java has replaced the destructor function with a finalize() function.
- There are no header files in Java.

JAVA ENVIRONMENT:

- Java environment includes a large number of development tools and hundreds of classes and methods.
  - The development tools are part of the system known as **Java Development Kit (JDK)** and
  - The classes and methods are part of the **Java Standard Library (JSL).** also known as the Application Programming Interface(API).

The Java Development Kit:

- The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include……
  - appletviewer (for viewing Java applets)
  - javac (Java compiler)
  - java (Java interpreter).
  - javap (Java disassembler)
  - javah (for C header files)
  - javadoc (for creating HTML documents)
  - jdb (Java debugger)
- Table 2.4 lists these tools and their descriptions.

Table 2.4 *Java Development Tools*

Tool	*Description*
appletviewer	Enables us to run Java applets (without actually using a Java-compatible browser).
java	Java interpreter, which runs applets and applications by reading and interpretin bytecode files.
javac	The Java compiler, which translates Java sourcecode to bytecode files that th interpreter can understand.
javadoc	Creates HTML-format documentation from Java source code files.
javah	Produces header files for use with native methods.
javap	Java disassembler, which enables us to convert bytecode files into a progra description.
jdb	Java debugger, which helps us to find errors in our programs.

- The way these tools are applied to build and run application programs is illustrated in Fig. 2.5.
  - To create a Java program, we need to create a source code file using a text editor.
  - The source code is then compiled using the Java compiler javac and executed using the Java interpreter java.
  - The Java debugger jdb is used to find errors, if any, in the source code.
  - A compiled Java program can be converted into a source code with the help of Java disassembler javap.

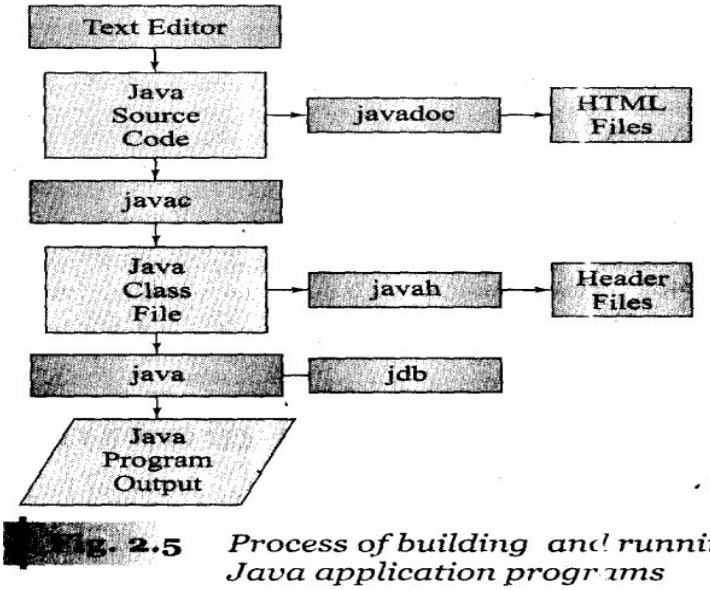
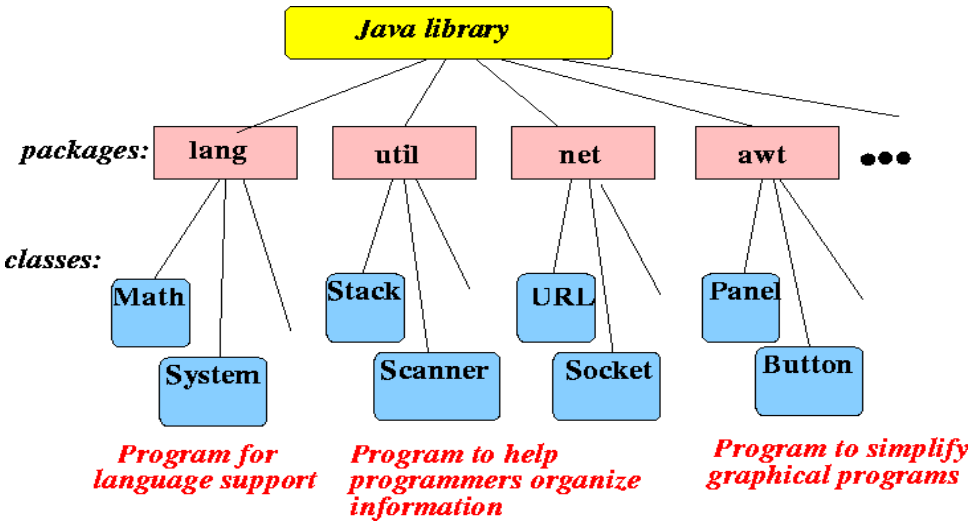


Fig. 2.5 *Process of building and running Java application programs*

Application Programming Interface:

- The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are……
  - **Language Support Package:** A collection of classes and methods required for implementing basic features of Java.
  - **Utilities Package:** A collection of classes to provide utility functions such as date and time functions.
  - **Input/Output Package:** A collection of classes required for input/output manipulation.
  - **Networking Package:** A collection of classes for communicating with other computers via Internet.
  - **AWT Package:** The Abstract Window Tool Kit package contains classes that implements platform- independent graphical user interface.

- **Applet Package:** This includes a set of classes that allows us to create Java app1et.



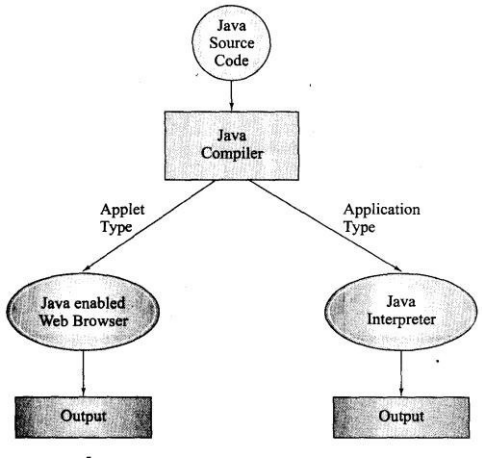
Java Runtime Environment (JRE):

- The Java Runtime Environment (JRE) facilitates the execution of programs developed in Java. It primarily comprises of the following:
  - **Java Virtual Machine (JVM):** It is a program that interprets the intermediate Java byte code and generates the desired output. It is because of byte code and JVM concepts that programs written in Java are highly portable.
  - **Runtime class libraries:** These are a set of core class libraries that are required for the execution of Java programs.
  - **User interface toolkits:** AWT and Swing are examples of toolkits that support varied input methods for the users to interact with the application program.
  - **Deployment technologies:** JRE comprises the following key deployment technologies:
  - **Java plug-in:** Enables the execution of a Java applet on the browser.
  - **Java Web Start:** Enables remote-deployment of an application. With Web Star, users can launch an application directly from the Web browser without going through the installation procedure.

Overview of Java Language:

- Java is a general-purpose, object-oriented programming language. We can develop two types of Java programs:
  - Stand-alone applications and Web applets.

- They are implemented as shown in Fig. 3.1.



- Stand-alone application programs written in Java to carry out certain tasks on a stand-alone local computer.
- Executing a stand-alone Java program by two steps…
  - Compiling source code into bytecode using javac compiler.
  - Executing the bytecode program using java interpreter.
- Applets are small Java programs developed for Internet applications.
- An applet located on a computer (Server) can be downloaded via Internet and executed on a local computer (Client) using a Java-capable browser.
- We can develop applets for doing everything from simple animated graphics to complex games and utilities.
- Since applets are embedded in an HTML (Hypertext Markup Language) document and run inside a Web page, creating and running applets are more complex than creating an application.
- Stand-alone programs can read and write files and perform certain operations that applets cannot do. An applet can only run within a Web browser.

SIMPLE JAVA PROGRAM:

- The best way to learn a new language is to write a few simple example programs and execute them. begin with a very simple program that prints a line of text as output.
- Program 3.1 is perhaps the simplest of all Java programs. Nevertheless, it brings out some salient feature of the language.
- Let us therefore discuss the program line by line and understand the unique features that constitute a Java program.

```
class SampleOne
{
    public static void main (String args[ ])
    {
        System. out .println ("Java is better than C+ + .");
    }
}
```

❖ **Class Declaration**

- The first line class SampleOne
- declares a class, which is an object-oriented construct.
- As stated earlier, Java is a true object-oriented language and therefore, everything must be placed inside a class, class is a keyword and declares that a new class definition follows.
- SampleOne is a Java identifier that specifies the name of the class to be defined.

❖ **Opening Brace:**

- Every class definition in Java begins with an opening brace "{" and ends with a matching closing brace "}", appearing in the last line in the example.
- This is similar to C++ class construct.

❖ **The Main Line**

- The third line public static void main (String args[])
- Defines a method named main. Conceptually, this is similar to the main() function in C/C++.
- Every Java application program must include the main() method. This is the starting point for the interpreter to begin the execution of the program.
- A Java application can have any number of classes but only one of them must include a main method to initiate the execution.
- This line contains a number of keywords, public, static and void.
- All parameters to a method are declared inside a pair of parentheses.
- Here, String args[] declares a parameter named args, which contains an array of objects of the class type String.

❖ **The Output Line**

- The only executable statement in the program is
- System.out.println("Java is better than C++");

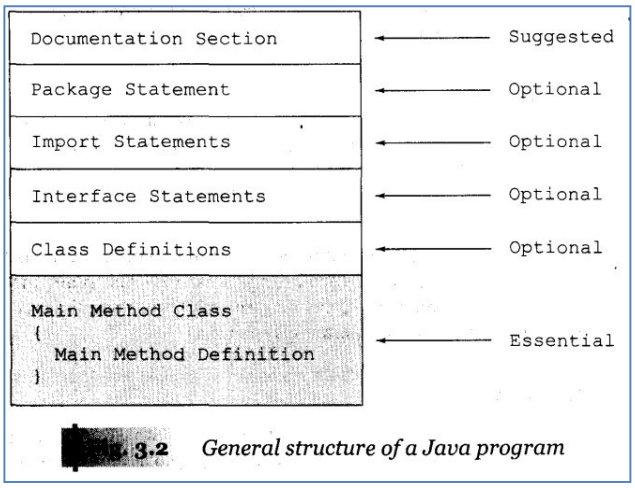
Public:	The keyword **public** is an access specifier that declares the **main** method as unprotected and therefore making it accessible to all other classes. This is similar to the C++ **public** modifier.
Static:	Next appears the keyword **static**, which declares this method as one that belongs to the entire class and not a part of any objects of the class. The **main** must always be declared as **static** since the interpreter uses this method before any objects are created. More about static methods ard variables will be discussed later in Chapter 8.
Void:	The type modifier **void** states that the **main** method does not return any value (but simply prints some text to the screen.)

- This is similar to the printf( ) statement of C or cout << construct of C++.
- Since Java is a true object-oriented language, every method must be part of an object.
- The println() method is a member of the out object, which is a static data member of System class.
- This line prints the string Java is better than C++.

JAVA PROGRAM STRUCTURE:

- As we have seen in the previous examples, a Java program may contain many classes of which only one class defines a main method.
- Classes contain data members and methods that operate on the data members of the class.
- Methods may contain data type declarations and executable statements.
- To write a Java program, we first define classes and then put them together.
- A Java program may contain one or more sections as shown in Fig. 3.2.
- **Documentation Section:**
  - The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage.

    // single line /* …… */ multiple lines.
  - Comments must explain why and what of classes and how of algorithms. This would greatly help in maintaining the program.



3.2 *General structure of a Java program*

- **Package Statement:**
  - The first statement allowed in a Java file is a package statement.
  - This statement declares a package name and informs the compiler that the classes defined here belong to this package. Example…… package student;
  - The package statement is optional.
- **Import Statements**
  - The next thing after a package statement (but before any class definitions) may be a number of import statements.
  - This is similar to the #include statement in C. Example: import student.test;
  - This statement instructs the interpreter to load the test class contained in the package student.
  - Using import statements, we can have access to classes that are part of other named packages.
- **Interface Statements:**
  - An interface is like a class but includes a group of method declarations.
  - This is also an optional section and is used only when we wish to implement the multiple inheritance feature in the program.
- **Class Definitions**
  - A Java program may contain multiple class definitions. Classes are the primary ad essential elements of a Java program.
  - These classes are used to map the objects of real-world problems. The number of classes used depends on the complexity of the problem.

- **Main Method Class**
  - Since every Java stand-alone program requires a main method as its starting point, this class is the essential part of a Java program.
  - A simple Java program may contain only this part. The main method creates objects of various classes and establishes communications between them.
  - On reaching the end of main, the program terminates and the control passes back to the operating system.

JAVA TOKENS:

- A Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements (see Fig. 3.3).
- Most statements contain expressions, which describe the actions carried out on data.
- Smallest individual units in a program are known as tokens. The compiler recognizes them for building up expressions and statements.
- In simple terms, a Java program is a collection of tokens, comments and white spaces.
- Java language includes five types of tokens. They are…
  - Reserved Keywords, Identifiers, Literals, Operators, Separators.

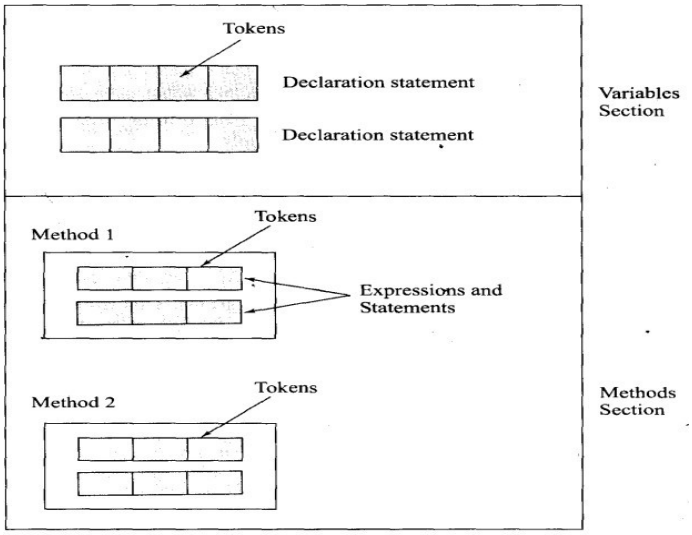


Fig. 3.3 *Elements of Java class*

❖ **Java Character Set:**

- The smallest units of Java language are the characters used to write Java tokens.
- These characters are defined by the Unicode character set, an emerging standard that tries to create characters for a large

number of scripts worldwide.

- The Unicode is a 16-bit character coding system and currently supports more than 34,000 defined characters derived from 24 languages from America, Europe, Middle East, Africa and Asia (including India).
- However, most of us use only the basic ASCII characters, which include letters, digits and punctuation marks, used in normal English.

❖ **Keywords**

- Keywords are an essential part of a language definition. They implement specific features of the language. Java language has reserved 50 words as keywords. Table 3.1 lists these keywords.

Table 3.1 *Java Keywords*

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

- These keywords, combined with operators and separators according to a syntax, form definition of the Java language.

❖ **Identifiers:**

- Identifiers are programmer-designed tokens.
- They are <u>used for naming classes, methods, variables, objects, labels, packages and interfaces</u> in a program.
- Java identifiers follow the following rule's:
  - They can have alphabets, digits, and the underscore and dollar sign characters.
  - They must not begin with a digit.
  - Uppercase and lowercase letters are distinct.
  - They can be of any length.
- Identifier must be meaningful, short enough to be quickly and easily typed and long enough to be descriptive and easily read.
- Java developers have followed <u>some naming conventions</u>……
  - Names of all public methods and instance variables start with a

leading lowercase letter.

- When more than one words are used in a name, the second 4 subsequent words are marked with a leading uppercase letters.
- All private and local variables use only lowercase letters combined with underscores.
- All classes and interfaces start with a leading uppercase letter (and each subsequent word with a leading uppercase letter).

• Variables that represent constant values use all uppercase letters and underscores between words.

• It should be remembered that all these are conventions and not rules.

❖ **Literals:**

• Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variable.

• Java language specifies five major types of literals……

• Integer literals, Floating_point literals, Character literals, String literals and Boolean literals.

• Each of them has a type associated with it. The type describes how the values behave and how they are stored.

❖ **Operators:**

• An operator is a symbol that takes one or more arguments and operates on them to produce a result.

❖ **Separators:**

• Separators are symbols used to indicate where groups of code are divided and arranged.

• The basically define the shape and function of our code.

• Table 3.2 lists separators and their functions……

Table 3.2 *Java Separators*

Name	*What it is used for*
parentheses ( )	Used to enclose parameters in method definition and invocation, also used for defining precedence in expressions, containing expressions for flow control, and surrounding cast types.
braces { }	Used to contain the values of automatically initialized arrays and to define a block of code for classes, methods and local scopes
brackets [ ]	Used to declare array types and for dereferencing array values
semicolon ;	Used to separate statements
comma ,	Used to separate consecutive identifiers in a variable declaration, also used to chain statements together inside a 'for' statement
period .	Used to separate package names from sub-packages and classes; also used to separate a variable or method from a reference variable.

JAVA STATEMENTS:

- The statements in Java are like sentences in natural languages.
- A statement is an executable combination of tokens ending with a semicolon (;) mark. Statements are usually executed in sequence in the order of occurrence.
  - However, it is possible to control the flow of execution, if necessary, using special statements. Java implements several types of statements as illustrated in Fig. 3.12 and described in Table 3.3.

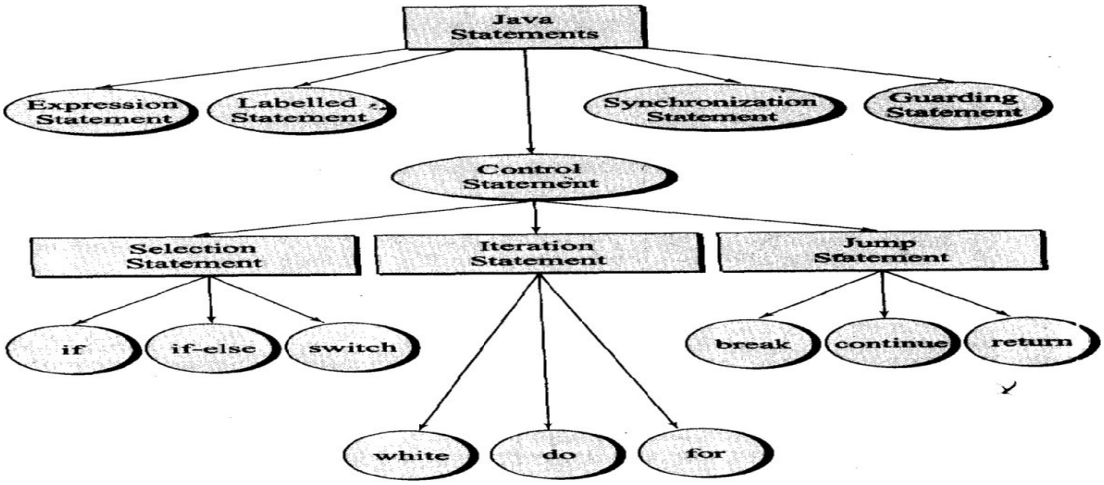


Fig. 3.12 *Classification of Java statements*

Table 3.3 Summary of Java Statements

Statement	*Description*	*Remarks*
Empty Statement	These do nothing and are used during program development as a place holder.	Same as C and C++
Labelled Statement	Any Statement may begin with a label. Such labels must not be keywords, already declared local variables or previously used labels in this module. Labels in Java are used as the arguments of Jump statements, which are described later in this list.	Identical to C and C++ except their use with jump statements
Expression Statement	Most statements are expression statements. Java has seven types of Expression statements: **Assignment, Pre-Increment, Pre-Decrement, Post-Increment, Post-Decrement, Method Call and Allocation Expression.**	Same as C++
Selection Statement	These select one of several control flows. There are Three types of selection statements in Java: **if, if-else,** and **switch**.	Same as C and C++

Iteration Statement	These specify how and when looping will take place. There are three types of iteration statements; **while, do** and **for**.	Same as C and C++ except for jumps and labels
Jump Statement	Jump Statements pass control to the beginning or end of the current block, or to a labeled statement. Such labels must be in the same block, and **continue** labels must be on an iteration statement. The four types of Jump statement are **break, continue, return** and **throw**.	C and C++ do not use labels with jump statements
Synchronization Statement	These are used for handling issues with multithreading.	Now available in C and C++
Guarding Statement	Guarding statements are used for safe handling of code that may cause exceptions (such as division by zero). These statements use the keywords **try, catch,** and **finally**.	Same as in C++ except finally statement.

JAVA VIRTUAL MACHINE:

- All language compilers translate source code into machine code for a specific computer.
- Java compiler also does the same thing. Then, how does Java achieve architecture neutrality?
- The answer is that the Java compiler produces an intermediate code known as bytecode for a machine that does not exist.
- This machine is called the Java Virtual Machine and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer.
- Figure illustrates the process of compiling a Java program into bytecode which is also referred to as virtual machine code.

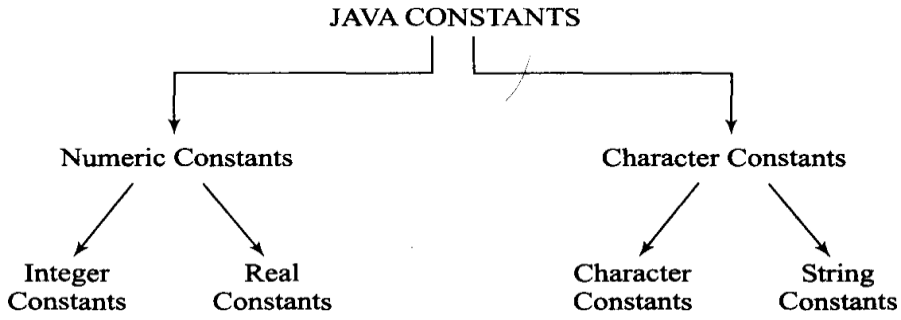
- The virtual machine code is not machine specific.
- The machine specific code (known a machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in following figure…

- Remember that the interpreter is different for different

Constants, Variables and Data Types:

CONSTANTS:

- Constants in Java refer to fixed values that do not change during the execution of a program.
- Java supports several types of constants as illustrated in following figure…



Integer Constants:

- An integer constant refers to a sequence of digits. There are three types of integers, namely decimal, octal and hexadecimal integer.
- Decimal integers consist of a set of digits, 0 through 9, preceded by an optional minus sign. Valid examples of decimal integer constants are: 123 –321 0 654321
- Embedded spaces, commas, and non-digit characters are not permitted between digits.

  15 750 20.000 $1000 are illegal numbers.

- An octal integer constant consists of any combination of digits from the set 0 through 7, with a leading 0. Some examples of octal integer are:037 0 0435 0551.
- A sequence of digits preceded by Ox or OX is considered as hexadecimal integer (hex integer). They may also include alphabets A through F or a through f.

Real Constants:

- Integer numbers are inadequate to represent quantities that vary

continuously, such as distances, heights, temperatures, prices, and so on.

- These quantities are represented by numbers containing fractional parts like 17.548. Such numbers are called real (or floating point) constants.
- Further examples of real constants are: 0.0083 -0.75 435.36
- These numbers are shown in decimal notation, having a whole number followed by a decimal point and the fractional part, which is an integer.

Single Character Constants:

- A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks.
- Examples of character constants are: '5' 'b' ';' ' '
- Note that the character constant '5' is not the same as the number 5. The last constant is a blank space.

String Constants:

- A string constant is a sequence of characters enclosed between double quotes.
- The characters may be alphabets, digits, special characters and blank spaces.
- Examples are:"WELL DONE" "?...!" "Hello Java" "1997"

Backslash Character Constants:

- Java supports some special backslash character constants that are used in output methods. For e.g., the symbol '\n' stands for newline character.
- A list of such backslash character constants is given in Table 4.1.

Table 4.1 *Backslash Character Constants*

Constant	*Meaning*	*Constant*	*Meaning*
'\b'	back space	'\t'	horizontal tab
'\f'	form feed	'\''	single quote
'\n'	new line	'\"'	double quote
'\r'	carriage return	'\\'	backslash

- Note that each one of them represents one character, although they consist of two characters. These characters combinations are known as escape sequences.

VARIABLES:

- A variable is an identifier that denotes a storage location used to store a data value.
- Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during the execution of the program.
- A variable name can be chosen by the programmer in a meaningful way so as to reflect what it represents in the program.
- As mentioned earlier, variable names may consist of alphabets, digits, the underscore(_) and dollar characters, subject to the **following conditions…**
  - They must not begin with a digit.
  - Uppercase and lowercase are distinct.
  - It should not be a keyword.
  - White space is not allowed.
  - Variable names can be of any length.

DATA TYPES:

- Every variable in Java has a data type. Data types specify the size and type of values that can be stored.
- Java language is rich in its data types. The variety of data types available allow the programmer to select the type appropriate to the needs of the application.
- Data types in Java under various categories are shown in Fig. 4.2.

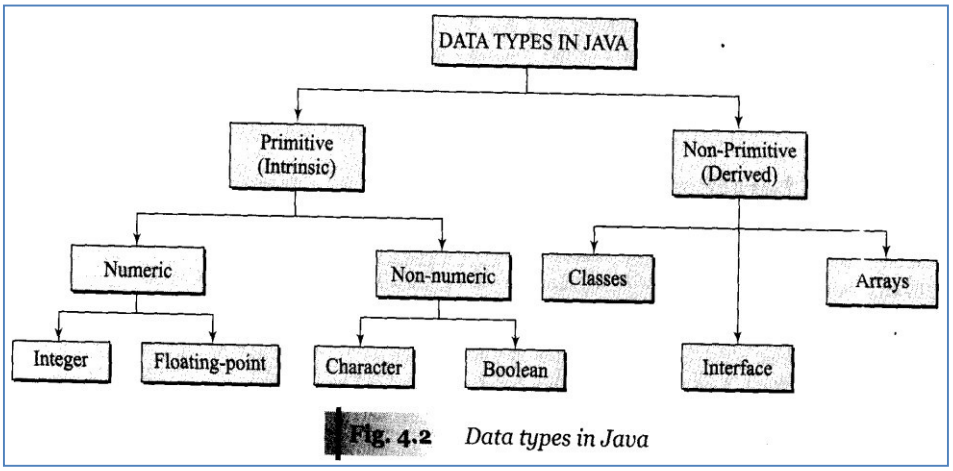


Fig. 4.2 *Data types in Java*

Integer Types:

- Integer types can hold whole numbers such as 123, -96, and 5639.
- The size of the values that can be stored depends on the integer data

type we choose. Java supports four types of integers such as byte, short, int, and long.

- Java does not support the concept of unsigned types and therefore all Java values are signed meaning they can be positive or negative.
- The below table shows the memory size and range of all the four integer data types…

Type	*Size*	*Minimum value*	*Maximum value*
byte	One byte	–128	127
short	Two bytes	–32, 768	32, 767
int	Four bytes	–2, 147, 483, 648	2, 147, 483, 647
long	Eight bytes	–9, 223, 372, 036, 854, 775., 808	9, 223, 372, 036, 854, 775, 807

- It should be remembered that wider data types require more time for manipulation and therefore it is advisable to use smaller data types, wherever possible.
- For example, instead of storing a number like 50 in an int type variable, we must use a byte variable to handle this number.
- This will improve the speed of execution of the program.
- We can make integers long by appending the letter L or l at the end of the number. Example… 123L or 123l

Floating Point Types:

- Integer types can hold only whole numbers and therefore we use another type known as floating point type to hold numbers containing fractional parts such as 27.59 and –1.375 (known as floating point constants).
- There are two kinds of floating point storage in Java such as float and double.
- The float type values are single-precision numbers, while the double types represent double- precision numbers.
- Table 4.3 gives the size and range of these two types.

Table 4.3 *Size and Range of Floating Point Types*

Type	*Size*	*Minimum value*	*Maximum value*
float	4 bytes	3.4e–038	1.7e+0.38
double	8 bytes	3.4e–038	1.7e+308

- Floating point numbers are treated as double-precision quantities. To force them to be in single- precision mode we must append f or F to the numbers. e.g.: 34.44f

- Floating point data types support a special value known as Not-a-Number(NaN), which is used to result of operations such as dividing zero by zero.

Character Type:

- In order to store character constants in memory, Java provides a character data type called char.
- The size of char type assumes a size of 2 bytes but basically it can hold only a single character.

Boolean Type:

- It is used when we want to test a particular condition during the execution of the program.
- There are only two values that a boolean type can take: true or false. It is denoted by a keyword boolean and uses only one bit of storage.

Declaration of Variables:

- In Java, after designing suitable variable names, we must declare them to the compiler. Declaration does three things……
  - It tells the compiler what the variable name is…?
  - It specifies what type of data the variable will hold…?
  - The place of the declaration (in the program) decides the scope of the variable.
- A variable can be used to store a value of any data type. That is, the name has nothing to do with the type.
- The declaration statement defines the type of variable, the general form of declaration of a variable is…

```
type var1, var2, …… varN;
```

- Variables are separated by commas and declaration statement must ends with a semicolon. Some valid declaration are……

```
int count;      float x, y;      double pi=3.1452;
```

Giving values to Variables:

- A variable must be given a value after it has been declared but before it is used in an expression. This is achieved in two ways……
  - By using an assignment statement.
  - By using a read statement.

Assignment statement:

- A simple method of giving value to a variable is through the assignment statement as follows…
- variableName = value; → num = 50;
- We can also string assignment expressions as shown below… x = y = z = 0;
- It is also possible to assign a value to a variable at the time of declaration as follows……
- type var = value; → float pi = 3.1452f;
- Read Statement:
- We may also give values to variables interactively through the keyboard using some built-in classes, e.g., Scanner class, DataInputStream etc.,

Scanner class:

- Scanner is the complement of Formatter. Added by JDK 5, Scanner reads formatted input and converts it into its binary form.
- Although it has always been possible to read formatted input, it required more effort than most programmers would prefer.
- Because of the addition of Scanner, it is now easy to read all types of numeric values, strings, and other types of data, whether it comes from a disk file, the keyboard, or another source.
- Scanner can be used to read input from the console, a file, a string, or any source that implements the Readable interface or ReadableByteChannel.
- The Scanner defines different types of constructors. In general, a Scanner can be created for a String, an InputStream, a File, or any object that implements the Readable or ReadableByteChannel interfaces.
- This next line creates a Scanner that reads from standard input, which is the keyboard by default:

Scanner conin = new Scanner(System.in);

- This works because System.in is an object of type InputStream. Thus, the call to the constructor maps to Scanner(InputStream).
- The next sequence creates a Scanner that reads from a string.

```
String instr = "10 99.88 scanning is easy.“;
Scanner conin = new Scanner(instr);
```

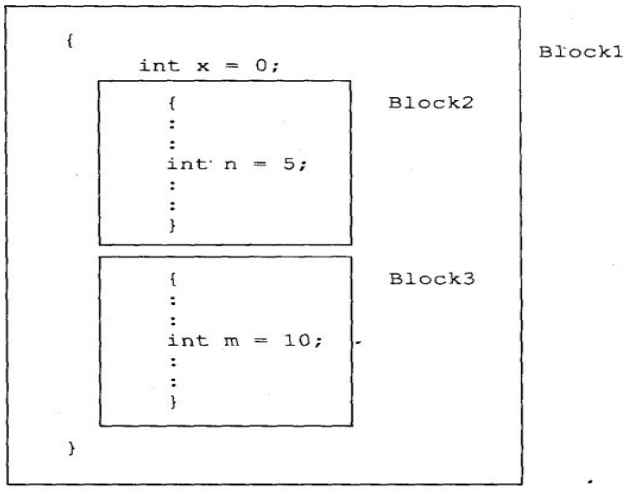
Method	Description
String next( )	Returns the next token of any type from the input source.
String next(Pattern *pattern*)	Returns the next token that matches the pattern passed in *pattern* from the input source.
String next(String *pattern*)	Returns the next token that matches the pattern passed in *pattern* from the input source.
boolean nextBoolean( )	Returns the next token as a **boolean** value.
byte nextByte( )	Returns the next token as a **byte** value. The default radix is used. (Unless changed, the default radix is 10.)
byte nextByte(int *radix*)	Returns the next token (using the specified radix) as a **byte** value.
double nextDouble( )	Returns the next token as a **double** value.
float nextFloat( )	Returns the next token as a **float** value.
int nextInt( )	Returns the next token as an **int** value. The default radix is used. (Unless changed, the default radix is 10.)
int nextInt(int *radix*)	Returns the next token (using the specified radix) as an **int** value.
String nextLine( )	Returns the next line of input as a string.
long nextLong( )	Returns the next token as a **long** value. The default radix is used. (Unless changed, the default radix is 10.)
long nextLong(int *radix*)	Returns the next token (using the specified radix) as a **long** value.
short nextShort( )	Returns the next token as a **short** value. The default radix is used. (Unless changed, the default radix is 10.)
short nextShort(int *radix*)	Returns the next token (using the specified radix) as a **short** value.

TABLE 18-16 The **Scanner next** Methods

SCOPE OF VARIABLES:

- Java variables are actually classified into three kinds:
  - instance variables,
  - class variables, and
  - local variables.
- Instance and class variables are declared inside a class.
- Instance variables are created when the objects are instantiated and therefore they are associated with the objects. They take different value for each object.
- On the other hand, class variables are global to a class and belong to the entire set of objects that class creates.
- Only one memory location is created for each class variable.
- Variables declared and used inside methods are called local variables. They are called so because they are not available for use outside the method definition.

- Local variables can also be declared inside program blocks that are defined between an opening brace { and a closing brace }.
- These variables are visible to the program only from the beginning of its program block to the end of the program block, When the program control leaves a block, all the variables in the block will cease to exist.
- The area of the program where the variable is accessible (i.e., usable) is called its scope. We can have program blocks within other program blocks (called nesting) as shown in Fig. 4.5.



4.5 *Nested program blocks*

- Each block can contain its own set of local variable declarations. We cannot, however, declare a variable to have the same name as one in an outer block.

SYMBOLIC CONSTANTS:

- We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program.
- One example of such a constants is 3.142, representing the value of the mathematical constant "pi".
- Another example is the total number of students whose mark-sheets are analyzed by a 'test analysis program'.
- We face two problems in the subsequent use of such programs……
  - Problem in modification of the program.
  - Problem in understanding the program.

Modifiability:

- We may like to change the value of "pi" from 3.142 to 3.14159 to improve the accuracy of calculations.

- In this cases, we will have to search throughout the program and explicitly change the value of the constant wherever it has been used.
- If any value is left unchanged, the program may produce disastrous outputs.

Understandability:

- When a numeric value appears in a program, its use is not always clear, especially when the same value means different things in different places.
- For e.g., the number 50 may mean the number of students at one place and the 'pass marks' at another place of the same program.
- We may forget what a certain number meant, when we read the program some days later.
- Assignment of a symbolic name to such constants frees us from these problems.
- For example, we may use the name STRENGTH to denote the number of students and PASS_MARK to denote the pass marks required in a subject.
- Constant values are assigned to these names at the beginning of the program.
- Subsequent use of the names STRENGTH and PASS_MARK in the program has the effect of causing their defined values to be automatically substituted at the appropriate points.
- A constant is declared and valid examples of constant declaration are as follows…

```
final type symbolic-name = value;
```

```
final     int      STRENGTH = 100;
final     int      PASS_MARK = 50;
final     float    PI = 3.14159;
```

- Note that:
  - Symbolic names take the same form as variable names. But, they are written in CAPITALS to visually distinguish them from normal variable names. This is only a convention, not a rule.
  - After declaration of symbolic constants, they should not be assigned any other value within the program by using an assignment statement. For example, STRENGTH = 200; is illegal.

- Symbolic constants are declared for types. This is not done in C and C++ where symbolic constants are defined using the #define statement.
- They can NOT be declared inside a method. They should be used only as class data members in the beginning of the class.

TYPE CASTING:

- We often encounter situations where there is a need to store a value of one type into a variable of another type.
- In such situations, we must cast the value to be stored by proceeding it with the type name in parentheses. The syntax is……

```
type variable1 = (type) variable2;
```

- The process of converting one data type to another is called casting. Examples:

```
int m = 50 ;     byte n = (byte)m;     long count = (long)m;
```

- Casting is often necessary when a method returns a type different than the one we require.
- Four integer types can be cast to any other type except boolean. Casting into a smaller type may result in a loss of data.
- Similarly, the float and double can be cast to any other type except boolean. Casting a floating-point value to an integer will result in a loss of the fractional part.
- Table 4.4 lists those casts, which are guaranteed to result in no loss of information.

AUTOMATIC CONVERSION:

- For some types, it is possible to assign a value of one type to a variable of a different type without a cast.

Table 4.4 *Casts that Results in No Loss of Information*

From	*To*
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

- Java does the conversion of the assigned value automatically. This is known as automatic type conversion.

- Automatic type conversion is possible only if the destination type has enough precession to store the source value.
- For example, int is large enough to hold a byte value. Therefore, byte b = 75; int a = b; are valid statements.
- The process of assigning a smaller type to a larger one is known as widening or promotion and that of assigning a larger type to a smaller one is known as narrowing.

STANDARD DEFAULT VALUES:

- In Java, every variable has a default value.
- If we don't initialize a variable when it is first created, Java provides default value to that variable type automatically as shown in Table 4.5.

Table 4.5 *Default Values for Various Types*

Types of variable	*Default value*
byte	Zero : (byte) 0
Short	Zero : (short) 0
int	Zero : 0
long	Zero : 0L
float	0.0f
double	0.0d
char	null character
boolean	false
reference	null

OPERATORS AND EXPRESSIONS:

- Java supports a rich, set of operators. We have already used several of them, such as, +, -, and *.
- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.

- They usually form a part of mathematical or logical expressions. Java operators can be classified into a number of related categories as below……

1. **Arithmetic operators**
2. **Relational operators**
3. **Logical operators**
4. **Assignment operators**
5. **Increment and decrement operators**
6. **Conditional operators**
7. **Bitwise operators**
8. **Special operators**

ARITHMETIC OPERATORS:

- Arithmetic operators are used to construct mathematical expressions as in algebra. Java provides all the basic arithmetic operators. They are listed in Table 5.1.

Operator	*Meaning*
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division (Remainder)

- The operators +, –, , and / all work the same way as they do in other languages. These can operate on any built-in numeric data type of Java.
- We cannot use these operators on boolean type. The unary minus operator, in effect, multiplies its single operand by –1. Therefore, a number preceded by a minus sign changes its sign.

RELATIONAL OPERATORS:

- We often compare two quantities, and depending on their relation, take certain decisions.
- For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.
- An expression containing a relational operator is termed as a relational expression, value of which is either true or false.
- Arithmetic operators have higher priority over relational operators.
- Relational expressions are used in decision statements such as, if and while to decide the course of action of a running program.

Table 5.2 *Relational Operators*

Operator	*Meaning*
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
= =	is equal to
!=	is not equal to

Table 5.3 *Relational Expressions*

Expression	*Value*
4.5 <= 10	TRUE
4.5 < –10	FALSE
–35 >= 0	FALSE
10 < 7 + 5	TRUE
a + b = = c + d	TRUE*

* Only if the sum of values of a and b is equal to the sum of values of c and d.

LOGICAL OPERATORS:

- In addition to the relational operators, Java has three logical operators, which are given in table.

Table 5.4 *Logical Operators*

Operator	*Meaning*
&&	logical AND
||	logical OR
!	logical NOT

- The logical operators && and || are used when we want to form compound conditions by combining two or more relations.
- An example is: a > b && x == 10
- An expression of this kind which combines two or more relational expressions is termed as a logical expression or a compound relational expression.
- Like the simple relational expression a logical expression also yields a value of true or false, according to the truth table shown in Table 5.5.

Table 5.5 *Truth Table*

		Value of the expression	
op – 1	*op – 2*	*op – 1 && op – 2*	*op – 1 || op – 2*
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Note:
- *op–1 && op–2 is true if both op–1 and op–2 are true and false otherwise.*
- *op–1 || op–2 is false if both op–1 and op–2 are false and true otherwise.*

ASSIGNMENT OPERATORS:

- Assignment operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator, '='.
- In addition, Java has a set of shorthand assignment operators which we used in the form here v is a variable. exp is an expression and op is a Java binary operator.
- The operator op= known as the shorthand assignment operator.
- The assignment statement…

  v op= exp; is equivalent to v = v op(exp); with v accessed only once.

Table 5.6 *Shorthand Assignment Operators*

Statement with simple assignment operator	*Statement with shorthand operator*
a = a + 1	a += 1
a = a–1	a –= 1
a = a*(n+1)	a *= n+1
a = a/(n+1)	a /= n+1
a = a%b	a %= b

INCREMENT AND DECREMENT OPERATORS:

- Java has two very useful operators not generally found in many other languages. These are the increment and decrement operators: ++ and --
- The operator ++ adds 1 to the operand while -- subtracts 1. Both are unary operators and are used in the following form……

  ++m; or m++; --m; or m--;
- We use the increment and decrement operators extensively in for and while loops.
- While ++m and m++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.
- Consider m = 5; y = ++m; In this case, the value of y and m would be 6.
- Suppose, if we rewrite the above statement as
  m=5; y = m++;
  then, the value of y would be 5 and m would be 6.

CONDITIONAL OPERATOR: ? :

- The character pair ?: is a ternary operator available in Java. This operator is used to construct conditional expressions of the form…
  exp1 ? exp2 : exp3;

  where exp1, exp2, and exp3 are expressions.
- The operator ?: works as follows…
  - exp1 is evaluated first. If it is nonzero (true), then the expression exp2. is evaluated and becomes the value of the conditional expression.
  - If exp1 is false, exp3 is evaluated and is value becomes the value of the conditional expression.
  - Note that only one of the expressions (either exp2 exp3) is evaluated.

BITWISE OPERATORS:

- Java has a distinction of supporting special operators known as bitwise operators for manipulation of data at values of bit level.
- These operators are used for testing the bits, or shifting them to the right or left.
- Bitwise operators may not be applied to float or double. Table 5.7 lists the bitwise operators.

Table 5.7 Bitwise Operators

Operator	Meaning
&	bitwise AND
!	bitwise OR
^	bitwise exclusive OR
~	one's complement
<<	shift left
>>	shift right
>>>	shift right with zero fill

SPECIAL OPERATORS:

- Java supports some special operators of interest such as instanceof operator and member selection operator(.).

instanceof Operator:

- The instanceof is an object reference operator and returns true if the object on the left-hand side is an instance of the class given on the right-hand side.
- This operator allows us to determine whether the object belongs to a particular class or not.
- Example: person instanceof student
- is true if the object person belongs to the class student; otherwise it is false.

Dot Operator:

- The dot operator (.) is used to access the instance variables and methods of class objects. For example……
- person1.age // Reference to the variable age.
- personl.salary( ) // Reference to the method salary()
- It is also used to access classes and sub-packages from a package.

ARITHMETIC EXPRESSIONS:

- An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.
- Java can handle any simple / complex mathematical expressions. Some of the examples of Java expressions are shown in Table 5.8.

Table 5.8 Expressions

Algebraic expression	Java expression
ab–c	a*b–c
(m+n) (x+y)	(m+n)*(x+y)
$\frac{ab}{c}$	a*b/c
$3x^2+2x+1$	3*x*x+2*x+1
$\frac{x}{y} + c$	x/y+c

EVALUATION OF EXPRESSIONS:

- Expressions are evaluated using an assignment statement of the form variable = expression; variable is any valid Java variable name.
- When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side.
- All variables used in the expression must be assigned values before evaluation is attempted.

OPERATOR PRECEDENCE AND ASSOCIATIVITY:

- Each operator in Java has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated.
- There are distinct levels of precedence and an operator may belong to one of the levels.
- The operators at the higher level of precedence are evaluated first.
- The operators of the same precedence are evaluated either from left to right or from right to left, depending on the level. This is known as the associativity property of an operator.

Table 5.11 *Summary of Java Operators*

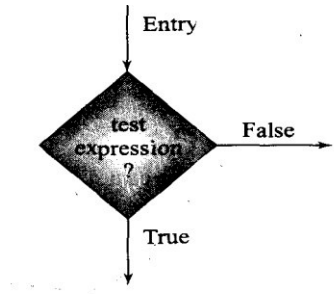
Operator	*Description*	*Associativity*	*Rank*
.	Member selection	Left to right	1
( )	Function call		
[ ]	Array element reference		
-	Unary minus	Right to left	2
+ +	Increment		
- -	Decrement		
!	Logical negation		
~	Ones complement		
(type)	Casting		
*	Multiplication	Left to right	3
/	Division		
%	Modulus		
+	Addition	Left to right	4
-	Subtraction		
<<	Left shift	Left to right	5
>>	Right shift		
>>>	Right shift with zero fill		
<	Less than	Left to right	6
<=	Less than or equal to		
>	Greater than		
> =	Greater than or equal to		
instanceof	Type comparison		
= =	Equality	Left to right	7
! =	Inequality		
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional operator	Right to left	13
=	Assignment operators	Right to left	14
op=	Shorthand assignment		

DECISION MAKING AND BRANCHING:

- A Java program is a set of statements, which are normally executed sequentially in the order in which they appear.
- However, in practice, we have a number of situations, where we may have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain conditions are met.
- This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.
- When a program breaks the sequential flow and jumps to another part of the code, it is called branching. When the branching is based on a particular condition, it is known as conditional branching.
- Java language possesses such decision making capabilities and supports the following statements as control or decision making statements…
  - if statement
  - switch statement.
  - Conditional operator statement.

DECISION MAKING WITH if STATEMENT:

- The if statement is a powerful decision making statement and is used to control the flow of execution of statements.
- It is basically a two-way decision statement and is used in conjunction with an expression. It takes the following form: **if (test expression)**
- It allows the computer to evaluate the expression first and then, depending on whether the value of the expression (relation or condition) is 'true' or 'false', it transfers the control to a particular statement.



6.1 *Two-way branching*

- This point of program has two paths to follow, one for the true condition and the other for the false condition as shown in Fig. 6.1.

- The if statement may be implemented in different forms depending on the complexity of conditions to be tested.
  - Simple if statement
  - if. . else statement
  - Nested if.. else statement
  - else if ladder.

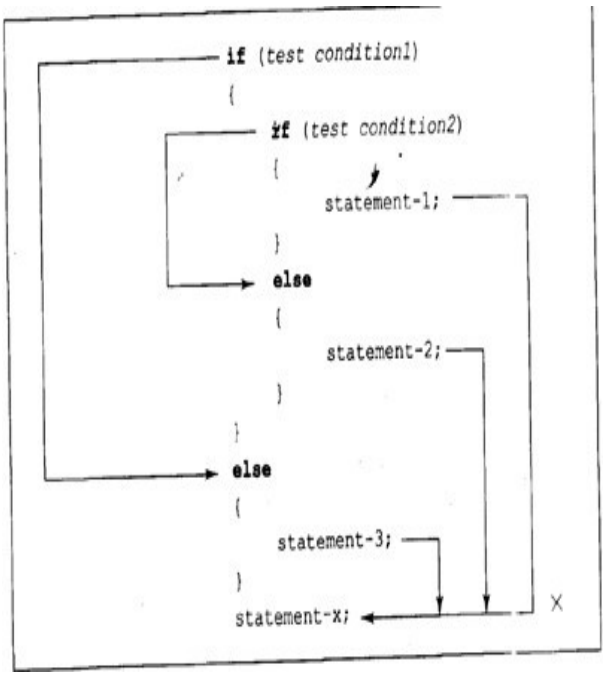
Simple if statement:

```
if (test expression)
{
   statement - block;
}
statement - X;
```

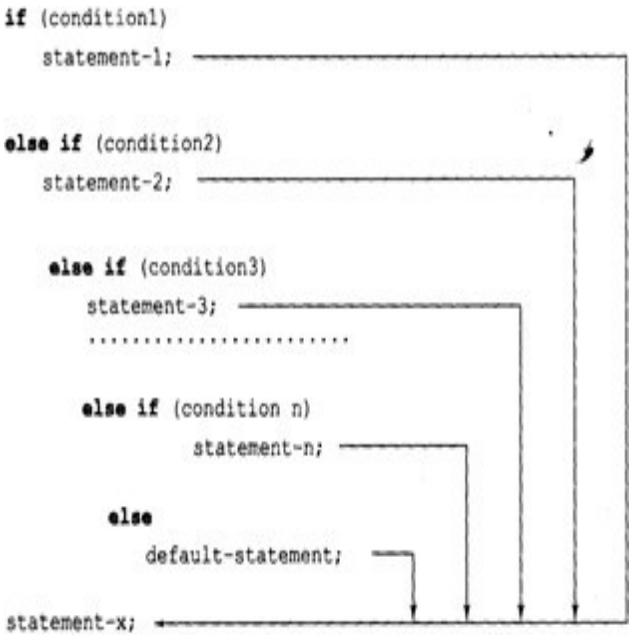
if …… else statement

```
if(test expression)
{
   True-block statement(s)
}
else
{
   False-block statement(s)
}
statement-X
```

Nested if.. else statement



else-if Ladder statement



THE SWITCH STATEMENT:

- We have seen that when one of the many alternatives is to be selected, we can design a program using if statements to control the selection.

- However, the complexity of such a program increases dramatically when the number of alternatives increases. The program becomes difficult to read and follow.
- At times, it may confuse even the designer of the program. Fortunately, Java has a built-in multiway decision statement known as switch.
- The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.
- The general form of the switch statement is as shown below……
- The expression is an integer expression or characters. value-1, value-2 .... are constants or constant expressions (evaluable to an integral constant) and are known as case labels.
- Each of these values sent, block-1, block-2 are statement lists and may contain zero or more statements.

- There is no need to put braces around these blocks but it is important to note that case labels end with a colon(:).

DECISION MAKING AND LOOPING:

- A computer is well suited to perform repetitive operations. It can do it tirelessly for 10, 100 or even 10,000 times.
- Every computer language must have features that instruct a computer perform such repetitive tasks. The process of repeatedly executing a block of statements is looping.
- The statements in the block may be executed any number of times, from zero to infinite number. If a loop continues forever, it is called an infinite loop.
- Java supports such looping features which enable us to develop concise programs containing repetitive processes without using unconditional branching statements like goto statement.
- In looping, a sequence of statements are executed until some conditions for the termination of the loop are satisfied.
- A program loop therefore consists of two segments, one known as the body of the loop and the other known as the control statement.
- The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.
- Depending on the position of the control statement in the loop, a control structure may be classified either as the entry-controlled loop or as exit-controlled loop.

- The flowcharts in figure 7.1 illustrate these structures…

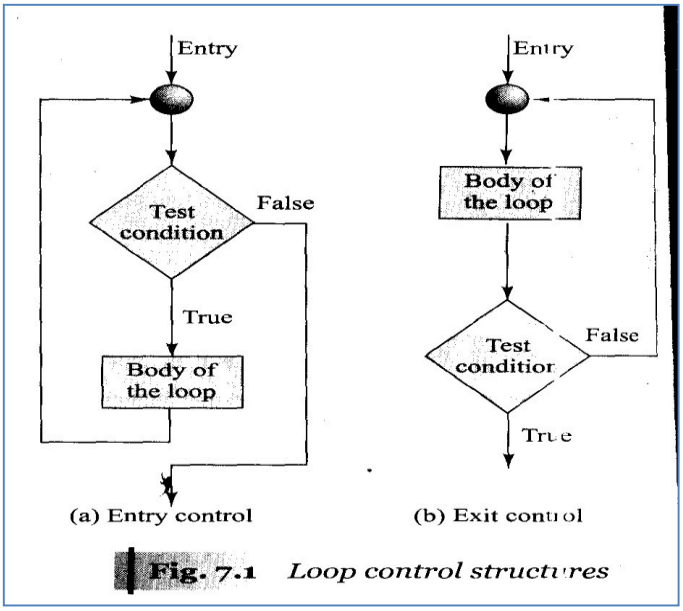


Fig. 7.1 *Loop control structures*

- A looping process, in general, would include the following four steps……
  - Setting and initialization of a counter.
  - Execution of the statements in the loop.
  - Test for a specified condition for execution of the loop.
  - Incrementing the counter.
  - The test may be either to determine whether the loop has been repeated the specified number of times or to determine whether a particular condition has been met with.
- The Java language provides for three constructs for performing loop operations. They are.
  - while construct, do-while construct and for construct

THE while STATEMENT:

- The simplest of all the looping structures in Java is the while statement. The while is an entry- controlled loop statement.
- The basic format of the while statement is…

```
Initialization:
While (test condition)
{
    Body of the loop
}
```

The do - while STATEMENT:

- The while loop construct that we have discussed makes a test condition before the loop is executed.
- Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt.
- On some occasions it might be necessary to execute the body of the loop before the test is performed.
- Such situations can be handled with the help of the do-while statement. This takes the form……
- Since the test condition is evaluated at the bottom of the loop, the do while construct provides an exit-controlled loop and therefore the body of the loop is always executed at least once

```
Initialization;
do
{
  Body of the loop
}
while (test condition);
```

Table 7.1 *Difference between **while** and **do-while** loops*

while	*do-while*
It is a looping construct that will execute only if the test condition is true.	It is a looping construct that will execute at least once even if the test condition is false.
It is an entry-controlled loop.	It is an exit-controlled loop.
It is generally used for implementing common looping situations.	It is typically used for implementing menu-based programs where the menu is required to be printed at least once.

THE for STATEMENT:

- The for loop is another entry-controlled loop that provides a more concise loop control structure. The general form of the for loop is…

```
for (initialization ; test condition ; increment)
{
     Body of the loop
}
```

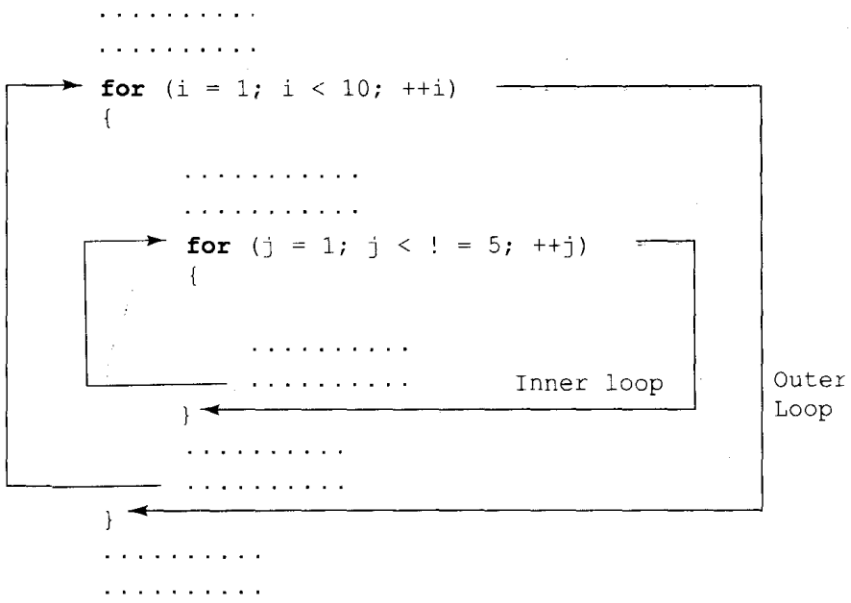
- The execution of the for statement is as follows:

- Initialization of the control variables is done first, using assignment statements such as i = 1 and count=0. The variables i and count are known as loop-control variables.
- The value of the control variable is tested using the test condition. The test condition is a relational expression, such as i < 10 that determines when the loop will exit.
- If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
- When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the control variable is incremented using an assignment statement such as i=i+1 and the new value of the control variable is again tested to see whether it satisfies the loop condition.
- If the condition is satisfied, the body of the loop is again executed.

- This process continues till the value of the control variable fails to satisfy the test condition.

Nesting of for Loops:

- Nesting of loops, that is, one for statement within another for statement, is allowed is Java. The for loops can be nested as follows…



The Enhanced for Loop:

- The enhanced for loop, also called for-each loop, is an extended language feature introduced with the J2SE 5.0 release.

- This feature helps us to retrieve the array of elements efficiently rather than using array indexes.
- We can also use this feature to eliminate the iterators in a for loop and to retrieve the elements from a collection.
- The enhanced for loop takes the following form……

```
for (Type Identifier : Expression)
{
    //statements;
}
```

where,

- Type represents the data type or object used;
- Identifier refers to the name of a variable; and
- Expression is an instance of the java.lang.Iterable interface or an array.

JUMPS in LOOPS:

- Loops perform a set of operations repeatedly until the control variable fails to satisfy the test condition.

- The number of times a loop is repeated is decided in advance and the test condition is written to achieve this.
- Sometimes, when executing a loop it becomes desirable to skip a part of the loop or to leave the loop as soon as a certain condition occurs.
- For example, consider the case of searching for a particular name in a list containing, say, 100 names.
- A program loop written for reading and testing the names a 100 times must be terminated as soon as the desired name is found.
- Java permits a jump from one statement to the end or beginning of a loop as well as a jump out of a loop.

Jumping Out of a Loop:

- An early exit from a loop can be accomplished by using the break statement. We have already seen the use of the break in the switch statement.
- break statement can also be used within while, do or for loops as illustrated in Fig. 7.2

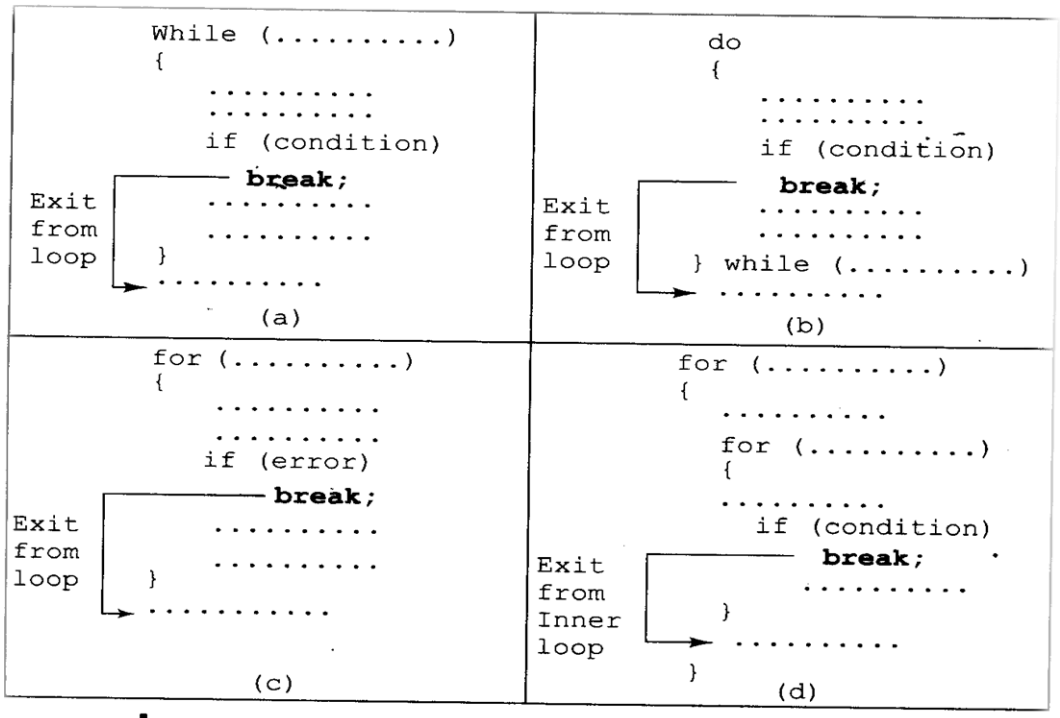


Fig. 7.2 *Exiting a loop with break statement*

- When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
- When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

Skipping a Part of a Loop:

- During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions.
- Like the break statement, Java supports another similar statement called the continue statement.
- However, unlike the break which causes the loop to be terminated, the continue, as the name implies, causes the loop to be continued with the next iteration after skipping any statements in between.
- The continue statement tells the compiler……
- "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".

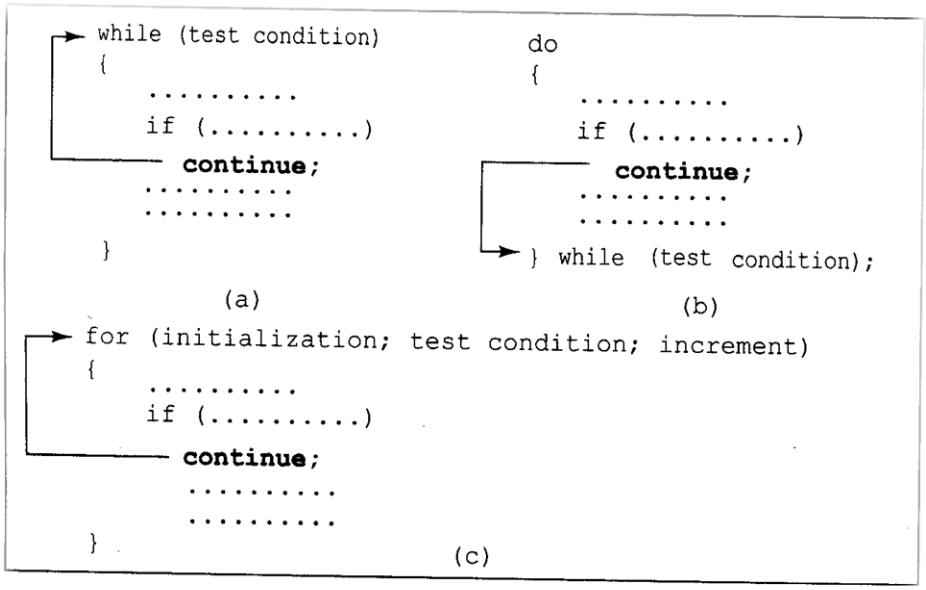


Fig. 7.3 *Bypassing and continuing in loops*

****

SAMPLE QUESTIONS:

1. Define Procedure Oriented Programming. Explain its characteristics.
2. What are the features of Object Oriented Programming? Explain.
3. Write a short note on: a) Data Abstraction and Encapsulation b) Inheritance and Polymorphism
4. Define OOP. Explain the Principle advantages of OOP.
5. Differentiate Object Oriented and Object Based Languages with its applications.
6. Explain any five features of Java Language.
7. Discuss how Java Language differs from C and C++ Language.
8. Discuss the importance of tools available in JDK with its usage.
9. Explain the importance of JSL or API and JRE in Java Language.
10. Define a Token. Explain the different tokens available in Java.
11. Define Constants. Explain briefly the different types of Constants available in Java Language.
12. Define a Data type. Explain briefly the different data types in Java.
13. Discuss any five methods available in Scanner to accept the input from user with its general syntax.

14. Write a short note on: a) Scope of Variables b) Symbolic constants
15. Illustrate the effect of pre-increment and post-increment and decrement operator in assignment statement with an example.
16. Write a java program to calculate the area and circumference of a circle by given radius.
17. Explain the different types of Decision making statements available with its general syntax and example.
18. Write a Java program to find the largest among given three integers.
19. Differentiate the Entry controlled and Exit controlled looping with an example.
20. Write a Java program to generate first 'N' terms of Fibonacci series.
21. Illustrate how the break and continue statements will behave in looping statements with an example.
22. Write a Java program to check whether given number is a palindrome or not.

****