Object oriented concepts

System Development (SD): -

o SD refers to all activities that go into producing an information systems solution. These activities consist of systems analysis, modeling, design, implementation, testing and maintenance.

o A software development methodology is a series of processes that if followed can lead to the development of an application.

o According to *Niklaus Wirth* – A software system is a set of mechanisms for performing certain action on certain data.

o There are *two orthogonal views* of the software differs in their primary focus.

1. The traditional approach focuses on the functions of the system and says software as a collection of programs (or functions) and isolated data.
2. Object oriented systems development centers on the object, which combines data and functionality i.e., Programs = Algorithms + Data Structures.

Object Oriented System Development: -

o Object oriented systems development is a way to develop software by building self – contained modules or objects that can be easily replaced, modified and reused.

o In an object–oriented environment, software is a collection of discrete objects that encapsulate their data as well as the functionality of model real–world events "*objects*" and emphasizes its cooperative philosophy by allocating tasks among the objects of the applications.

o A *class* is an object oriented system carefully delineates between its interface (specifications of what the class can do) and the implementation of that interface (how the class does what it does).

Need of Object Orientation

An object orientation produces systems that are easier to evolve, more flexible, more robust, and more reusable than a top – down approach. An object orientation

o *Allows higher level of abstraction*: Object oriented approach supports abstraction at the object level. Since objects encapsulate both data (attributes) & functions (methods), they work at a higher level of abstraction. This makes designing, coding, testing & maintaining the system much simpler.

o *Provides Seamless transition among different phases of software development*: Object oriented approach, essentially uses the same language to talk about analysis, design, programming and database design. This seamless approach reduces the level of complexity and redundancy and makes for clearer, more robust system development.

o *Encourage good development techniques*: In a properly designed system, the routines and attributes within a class are held together tightly, the classes will be grouped into subsystems but remain independently and therefore, changing one class has no impact on other classes and so, the impact is minimized.

o *Promotes of reusability*: Objects are reusable because they are modeled directly out of a real – world problem domain. Here classes are designed, with reuse as a constant background goal. All the previous functionality remains and can be reused without changed.

Object Basics: -

o ***Objects are Grouped in Classes*** **:** Class is a set of objects that share a common structure and a common behavior, a single object is simply an instance of a class. A class is a specification of structure (instance variables), behavior (methods) and inheritance for objects. A method or behavior of an object is defined by its class.

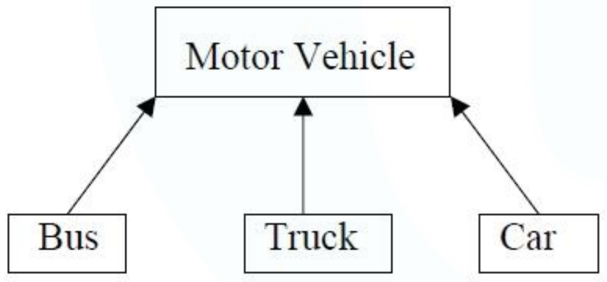
o ***Attributes*****:** Object state and properties. Properties represent the state of an object. Often, we refer to the description of these properties rather than how they are represented in a particular programming language.

o ***Object Behavior and Methods*****:** Behavior denotes the collection of methods that abstractly describes where an object is capable of doing. Methods encapsulate the behavior of the object, provide interfaces to an object and hide any of the internal structures and states maintained by the object.

o ***Objects respond to Messages*****:** Message is the instruction and method is the implementation. An object understands a message when it can match message to a method that has same name as of it.

o ***Encapsulation and Information Binding*****:** Information hiding is principle of concealing the internal data & procedures of an object and providing an interface to each object in such a way as to reveal as little as possible about its inner workings. Encapsulation protection deals with protection of an object capsule from other object by means of per–class protection and per–object protection.

o ***Class Hierarchy*****:** At the top of the class hierarchy are the most general classes & at the bottom are the most specific. A subclass inherits all of the properties and methods defined in its super class. Formal or abstract classes have no instances but define common behaviors that can be inherited by more specific classes.

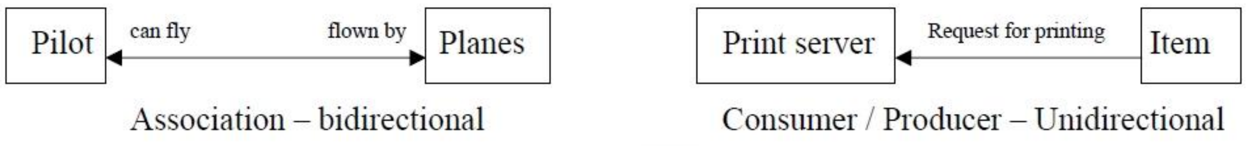


o ***Inheritance*****:** It is the property of object–oriented systems that allows objects to be built from other objects & allows explicitly taking of commonality of objects when constructing new classes. Dynamic inheritance allows objects to change and evolve over time and refers to the ability to add, delete, or change parents from objects (or classes) at run time.

o ***Multiple Inheritance*****:** Some object–oriented systems permit a class to inherit its state (attributes) and behaviors from more than one super class and is referred to as multiple inheritances.

o ***Polymorphism*****:** It means that the same operation may behave differently on different classes. Booach defines it as the relationship of objects of many different classes by some common super class; thus, any of the objects designated by this name is able to respond to some common set of operations in a different way.

o ***Object Relationships*:** Association represents the relationships between objects and classes. Associations are bidirectional with different annotations. Cardinality specifies how many instances of one class may relate to a single instance of an associated class. A special form of association is a consumer–producer relationship also known as client–server association or use-relationship and is can be viewed as one way interaction.

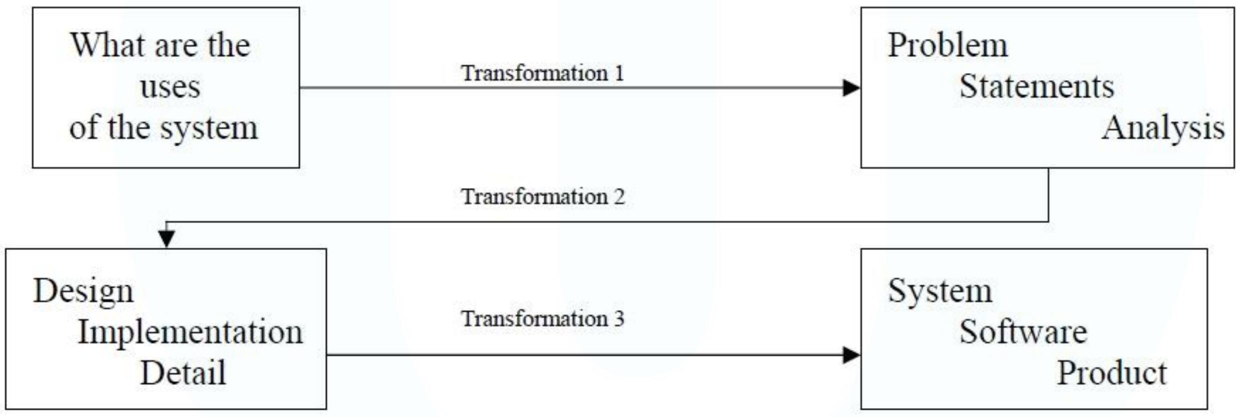

Pilot
can fly
flown by
Planes
Association – bidirectional
Print server
Request for printing
Item
Consumer / Producer – Unidirectional

o ***Aggregation and Object Containment*:** As each object has an identity, one object can refer to other objects and is known as aggregation, where an attribute can be an object itself.

<u>Object – Oriented System Development Life Cycle: -</u>

o The basic software development life cycle consists of ***analysis, design, implementation, testing and refinement.*** Its main aim is to transform users' needs into a software solution.

o The development is a process of change, refinement, transformation or addition to the existing product. The software development process can be viewed as a series of transformations, where the output of one transformation becomes input of the subsequent transformation.

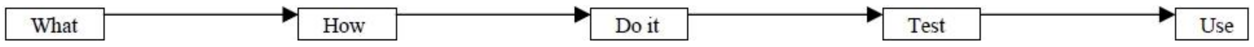

What are the uses of the system
Transformation 1
Problem Statements Analysis
Transformation 2
Design Implementation Detail
Transformation 3
System Software Product

o Transformation 1 (*analysis*) translates the users' needs into system requirements & responsibilities.

o Transformation 2 (*design*) begins with a problem statement and ends with a detailed design that can be transformed into an operational system. It includes the bulk of s/w development activity.

o Transformation 3 (*implementation*) refines the detailed design into the system deployment that will satisfy the users' needs. It represents embedding s/w product within its operational environment.

o An example of s/w development process is the waterfall approach which can be stated as below



<u>The software development process consists of</u>

(1)Analysis – Translates the user needs into system requirements and responsibilities.
(2) Design – It begins with a problem statement and ends with a detailed design that can be transformed into an operational system.
(3) Implementation – It regines the detailed design into the system deployment that will satisfy the users needs.
(4) Testing – Two basic approaches to system testing, they are
   (i) Test according to how it has been built for.
   (ii) What it should do?
(5) Maintenance/refinement.

This model has the following activities.

- System/Information Engineering and Modeling

As software is always of a large system (or business), work begins by establishing the requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when the software must interface with other elements such as hardware, people and other resources. System is the basic and very critical requirement for the existence of software in any entity. So if the system is not in place, the system should be engineered and put in place. In some cases, to extract the maximum output, the system should be re-engineered and spruced up. Once the ideal system is engineered or tuned, the development team studies the software requirement for the system.

- Software Requirement Analysis

This process is also known as feasibility study. In this phase, the development team visits the customer and studies their system. They investigate the need for possible software automation in the given system. By the end of the feasibility study, the team furnishes a document that holds the different specific recommendations for the candidate system. It also includes the personnel assignments, costs, project schedule, target dates etc.. The requirement gathering process is intensified and focused specially on software. To understand the nature of the program(s) to be built, the system engineer or "Analyst" must understand the information domain for the software, as well as required function, behavior, performance and interfacing. The essential purpose of this phase is to find the need and to define the problem that needs to be solved.

- System Analysis and Design

In this phase, the software development process, the software's overall structure and its nuances are defined. In terms of the client/server technology, the number of tiers needed for the package architecture, the database design, the data structure design etc.. are all defined in this phase. A software development model is thus created. Analysis and Design are very crucial in the whole development cycle. Any glitch in the design phase could be very expensive to solve in the later stage of the software development. Much care is taken during this phase. The logical system of the product is developed in this phase.

- Code Generation

The design must be translated into a machine-readable form. The code generation step performs this task. If the design is performed in a detailed manner, code generation can be accomplished without much complication. Programming tools like compilers, interpreters, debuggers etc.. are used to generate the code. Different high level programming languages like C, C++, Pascal, Java are used for coding. With respect to the type of application, the right programming language is chosen.

- Testing

Once the code is generated, the software program testing begins. Different testing methodologies are available to unravel the bugs that were committed during the previous phases. Different testing tools and methodologies are already available. Some companies build their own testing tools that are tailor made for their own development operations.

- Maintenance

The software will definitely undergo change once it is delivered to the customer. There can be many reasons for this change to occur. Change could happen because of some unexpected input values into the system. In addition, the changes in the system could directly affect the software operations. The software should be developed to accommodate changes that could happen during the post implementation period.

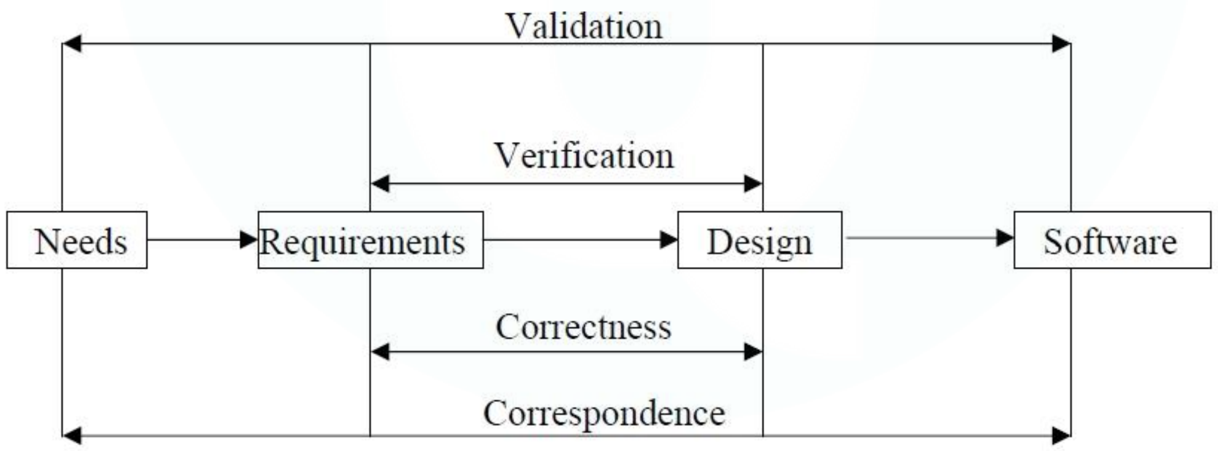
Building High–quality Software (BHS): -

o ***Ultimate*** *goal* of BHS is user satisfaction. Blum describes a means of system evaluation in terms of four quality measures: correspondence, correctness, verification & validation.

o ***Correspondence*** measures how well the delivered system matches the needs of the operational environment as described in the original requirements statement.

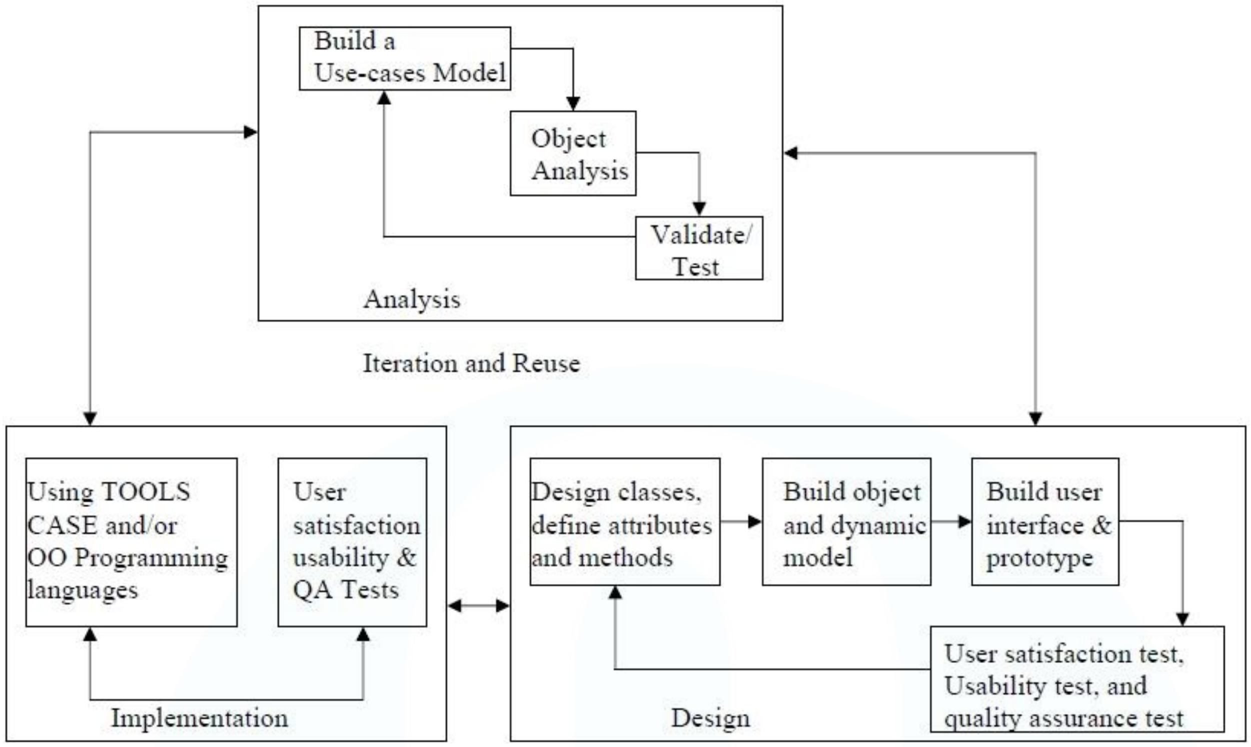
o ***Correctness*** measures consistency of product requirements with respect to the design specification.

o ***Verification*** is the exercise of determining correctness

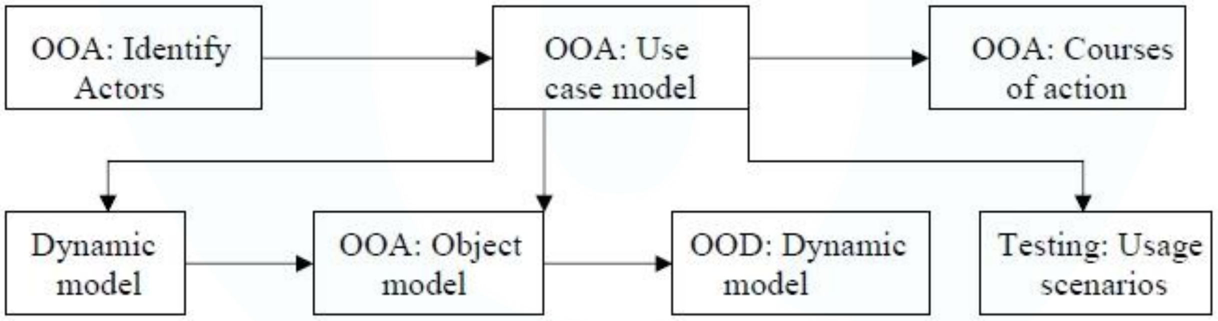


Object Oriented System Development: A use Case Driven Approach: -

o The object oriented software development life cycle (*SDLC*) consists of *three macro processes*: object–oriented analysis, object–oriented design and object–oriented implementation.


Build a Use-cases Model
Object Analysis
Validate/ Test
Analysis
Iteration and Reuse
Using TOOLS CASE and/or OO Programming languages
User satisfaction usability & QA Tests
Implementation
Design classes, define attributes and methods
Build object and dynamic model
Build user interface & prototype
User satisfaction test, Usability test, and quality assurance test
Design

o By following the life cycle model of *Jacobson, Ericsson and Jacobson*, once can produce designs that are traceable across requirements, analysis, design, implementation and testing. The main *advantage* is that all design decisions can be traced back directly to user requirements.



o Object–oriented system *development* includes object–oriented analysis, object–oriented design, Prototyping, Component–based development, Incremental testing.

<u>Object–Oriented Analysis – Use case driven (OOA) : -</u>

o This phase of s/w development is concerned with determining the system requirements and identifying classes and their relationship to other classes in the problem domain.

o Scenarios are a great way of examining who does what in the interactions among objects and what role they play; that is, their interrelationships. This intersection among objects' roles to achieve a given goal is called *collaboration*.

o In essence, a *use case* is a typical interaction between a user and system that captures users' goals and needs. Expressing high level processes & interactions with customers in scenario & analyzing it is referred to as use–case modeling. It represents users' view of the system or users' needs.

o Looking at the physical objects in the system also provides us important information on objects in the systems. The objects could be individuals, organizations, machines, units of information, pictures, or whatever else makes sense in the context of the real–world systems.

o In regarding documentation, *80 – 20 rule* is generally applies where 80 percent of the work can be done with 20 percent of the documentation. Good modeling implies good documentation.

<u>Object–Oriented Design (OOD) : -</u>

o The *goal of OOD* is to design the classes identified during the analysis phase and the user interface. Here, we identify & define additional objects, classes that support implementation of requirements.

o OOD is highly *incremental*. OOA can done, model it, create OOD them do some more on it.

o Here, we first, build the object model based on objects and their relationships, then iterate and refine the model such as

* Design and refine classes
* Design and refine attributes
* Design and refine methods
* Design and refine structures
* Design and refine associations.

o *Guidelines to use in OOD*:

* Reuse, rather than build, a new class. Know the existing classes
* Design a large number of simple classes, rather than a small number of complex classes
* Design methods
* Critique what you have proposed. If possible, go back and refine the classes.

<u>Prototyping : -</u>

o A *prototype* is a version of a software product developed in the early stages of the product's life cycle for specific, experimental purposes. It enables to fully understand how easy or difficult it will be to implement some of the features of the system.

o Prototyping can further define the use cases, and it actually makes use–case modeling much easier. The main idea is to build a prototype (use–case modeling) to design systems that users like & need.

o It provides the developer a means to test & refine user interface & increase usability of system.

o The following categories are some of accepted prototypes each having its own strength.

Horizontal Prototype: It is a simulation of interface but contain no functionality. Advantages: Very quick to implement, good overall of system, user to evaluate interface on normal base

Vertical Prototype: It is a subset of system features with complete functionality. Advantage: few implemented functions can be tested in great depth

Analysis Prototype: It is an aid for exploring problem domain. It used to inform user & demonstrate proof of a concept

Domain Prototype: It is an aid for incremental development of ultimate software development

o Prototyping should involve representation from all user groups that will be affected by the project, especially the end users and management members to ascertain that the general structure of the prototype meets the requirements established for the over all design.

o The *purpose of the review* is three fold are

- To demonstrate that the prototype has been developed according to the specification and that the final specification is appropriate.
- To collect information about errors or other problems in the system, such as user interface problems that need to be addressed in the intermediate prototype stage.
- To give management and everyone connected with the project the first glimpse of what the technology can provide.

o Prototyping is a useful exercise at almost any stage of the development. It should be done in parallel with the preparation of the functional specification.

Implementation: Component Based Development (CBD): -

o Software components are built & tested in–house using a wide range of technologies. Computer–aided S/w Engineering (*CASE*) tools allow their users to rapidly develop information systems.

o The main *goal* of CASE technology is the automation of the entire information system's development life cycle process using a set of integrated software tools, such as modeling, methodology, and automatic code generation.

o CBD is an *industrialized approach* to the s/w development process. Application development moves from custom development to assembly of pre-built, pre-tested, reusable s/w components that operate with each other.

o *Two basic* ideas underlie CBD move application development from a craft activity to an industrial process fit to meet the needs of modern, highly dynamic, competitive, global businesses.

- Application development can be improved significantly if application can be assembled quickly from prefabricated software components.
- An increasingly large collection of interpretable s/w components could be made available to developers in both general and specialist catalogs.

o The *s/w components* are the functional units of a program, building blocks offering a collection of re-usable services. A s/w component can request a service from another component or deliver its own services on request.

o Rapid Application Development (*RAD*) is a set of tools and techniques that can be used to build an application faster than typically possible with traditional methods.

o The *main objective of RAD* is to build a version of an application rapidly to see whether we actually have understood the problem (analysis) & determines whether the system does what it is supposed to do (Design).

Incremental Testing: -

o The problem was that developers would turn over application to a quality assurance (QA) group for testing only after development was completed.

o In most cases testing an application for bugs and performance would waste of time and money. *Glenn Shimamoto*, vice president of technology and strategic planning at the New York Bank says "Our testing was very complete and good, but it was costing a lot of money and would add months onto a project". Thence, no clear picture of the system characteristics are been categorized.

Reusability: -

o A *major benefit of OOSD* is reusability and this is the most difficult promise to deliver on. For an really reusable object, more effort must be spent designing it.

o The potential benefits of *reuse are clear*: increased reliability, reduced time and cost for development, and improved consistency.

o For effective reusability of existing s/w components we are suppose ask two questions – Has my problem already been solved or partially solved? What has been done before to solve a problem similar to this one?

o The *reuse strategy* can be based on the following: -

* Information hiding (encapsulation) * Conformance to naming standards

* Creation and administration of an object repository.

* Encouragement by strategic management of reuse as opposed to constant redevelopment.

* Establishing targets for a percentage of the objects in the project to be reused. (50% of objects).

Modeling: -

o A model is an *abstract representation* of a system constructed to understand the system prior to building or modifying it. The term system refers to any process of structure. As a general aspects, models can represent static or dynamic situations

o *Static Model*: It can be viewed as a snapshot of a system's parameters at rest or at a specific point in time. It is used to represent structural or static aspect of system. It assume stability and absence of change in data over time. E.g: UML – class diagram

o *Dynamic Model*: It can viewed as a collection as collection of procedures or behaviors taken together which reflect behavior of system over time. Dynamic relationships show the business objects interaction in performing task. Dynamic modeling is most useful during design and implementation phases of system development. E.g.,: UML – interaction diagram

o *Need of modeling*: A model for s/w is similar to blueprint for building. Good models are essential for communication among project teams. A modeling language must include

Model elements – fundamental modeling concepts and semantics

Notation – visual rendering of model elements

Guidelines – expression of usage within trade

o The use of visual notation to represent or model a problem can provide us several benefits relating to *clarity*, *familiarity*, *maintenance* and *simplification*. Turban states following advantages of modeling:

1. Models make it easier to express complex ideas
2. Reduction of complexity makes easier understanding of complex situations
3. Models enhance and reinforce learning and training
4. Cost of modeling analysis is much lower than experimentation on real system
5. Manipulation of model (changing variables) is easier than in real system

Unified Modeling Language (UML) : -

o UML is a language for specifying, constructing, visualizing and documenting the s/w system and its components. UML is a graphical language with set of rules & semantics expressed in English in form known as *object constraint language* (OCL)

o The *goals of unification* of all modeling concepts are to cast away elements of existing Booch, OMT & OOSE methods that didn't work in practice, to add elements from other methods that are more effective and to invent new methods only when an existing solution was unavailable

o The *primary goals* of design of UML are as follows:

- Provides users a ready–to–use, expressive visual modeling language so they can develop and exchange meaningful models
- Provide extensibility and specialization mechanisms to extend the core concepts
- Be independent of particular programming languages and development processes
- Provided a formal basis for understanding the modeling language
- Encourage the growth of OO tools market
- Support higher – level development concepts
- Integrate best practices and methodologies

o ***UML Diagrams*****:** Every complex system is best approached through small set of nearly independent views of a model with number of diagrams at each development level. UML defines *9 graphical diagrams*:

1. Class diagram (static diagram),
2. Use–case diagram,
3. Behavior diagram (dynamic),
    a. Interaction diagram
        – Sequence diagram, Collaboration diagram
    b. State chart diagram,
    c. Activity diagram
4. Implementation diagram : Component diagram, Deployment diagram

o ***Class diagram***: It is referred to object modeling, is main static structure analysis diagram for system. It represents the class structure of a system with relationships between classes and inheritance structure. It is developed through use–case, sequence and collaboration diagrams

a. *Class notation*: Drawn as rectangle with three components – name, attributes, list of operations

b. *Object diagram*: A static object diagram is an instance of class diagram. It shows as snapshot of detailed state of system at a point in time. Notation is same as class notation

c. *Class Interface* notation is used to describe externally visible behavior of a class. It is a design activity and its notation is doted arrow attached to small circle

d. *Association Role*: A simple association is binary association drawn as solid line connector

e. *Qualifier*: It is an association attribute. Eg: A person object associated to Bank object and an attribute of this association is account# which is a qualifier of this association

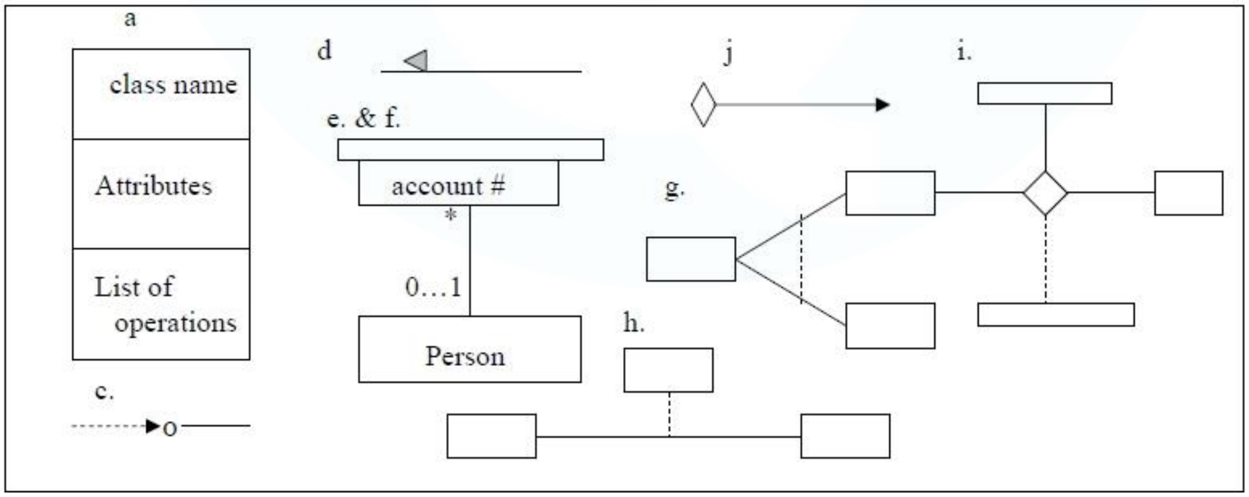
f. *Multiplicity*: It specifies range of allowable associated classes. It is given for roles with associations, parts within compositions, repetitions denoted by lower bound ……. Upper bound

g. *OR Association*: Any instance of class participates in, at most, one of associations at some time. It is denoted by dashed line connecting two associations with {or} labeling

h. *Association class*: It is an association that also has class properties. It is shown as class symbol attached by dashed line to an association line

i. *N–Ary Association*: An association between more than one class & shown in large diamond

j. *Generalization*: It is relationship between a more general class and more specific class. It is displayed as a directed line with a closed, hollow arrowhead at super class end



o ***Use–case diagram***: It captures information on how the system or business works or how you wish it to work. It is a scenario –building approach in which you model the processes of the system. It is an excellent way to lead into OOA of the system.

A use – case diagram is a graph of actors, as set of use cases enclosed by a system boundary, communication associations between the actors and use cases and generalization among use cases. 3 relationships shown are communication, uses and extends

When will be Extends association used?

The extends association is used when you have one case that is similar to another use case but does a bit more specialized; in essence, it is like a subclass. In our example, checking out a book is the basic use case. This is the case that will represent what happens when all goes smoothly.

When uses association will occur?

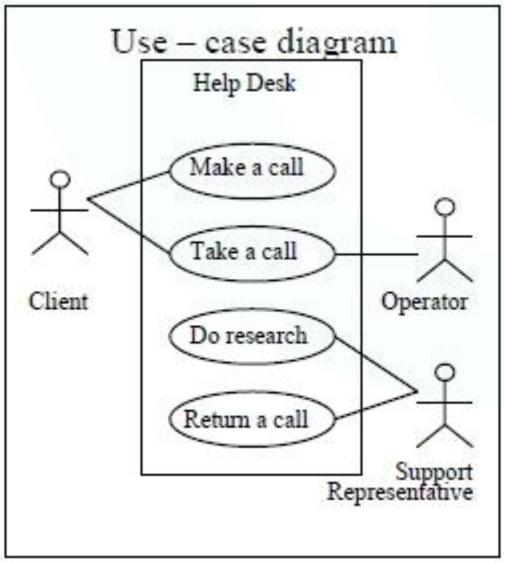
The uses association occurs when you are describing you use cases and notice that some of them have subflows in common. The relationship among the other use cases and this new extracted use case is called a uses association. The uses association helps us avoid redundancy by allowing a use case to be shared. For example, checking a library card is common among the borrow books, return books, and interlibrary loan use cases.

Steps for finding usecases.

1. For each actor, find the tasks and functions that the actor should be able to perform or that the system needs the actor to perform. The use case should represent a course of events that leads to a clear goal.

2. Name the use cases.

3. Describe the use cases briefly by applying terms with which the user is familiar. This makes the description less ambiguous.



o *Sequence diagram*: It is for dynamic modeling, where objects are presented by vertical lines, messages passed back & forth between objects are modeled by horizontal vectors between objects o *Collaboration diagram*: An alternative view of sequence diagram, showing in a scenario how objects interrelate with one another

o *State chart diagram*: It is another form of dynamic modeling, focus on events occurring within a single object as it responds to messages.

o *Activity diagram*: It is used to model an entire business process. Thus, an activity model can represent several different classes

o *Implementation diagram*: It show the implementation phase of systems development, such as source code and run –time implementation structures. Implementation diagram shows structure of code itself and deployment diagram diagrams show the structure of run-time system.

What is Java

Java is a **programming language** and a **platform**. Java is a high level, robust, secured and object-oriented programming language.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

Where it is used?

There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in, javatpoint.com etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

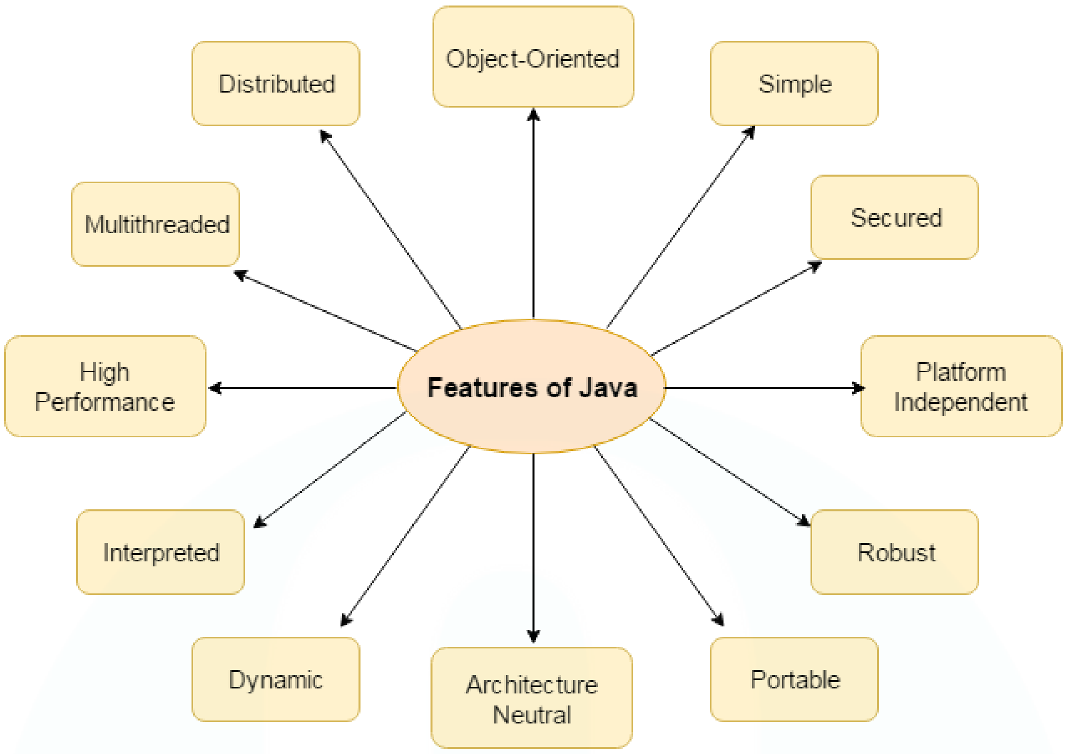
4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

Module 1 Part II



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Simple

According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are:

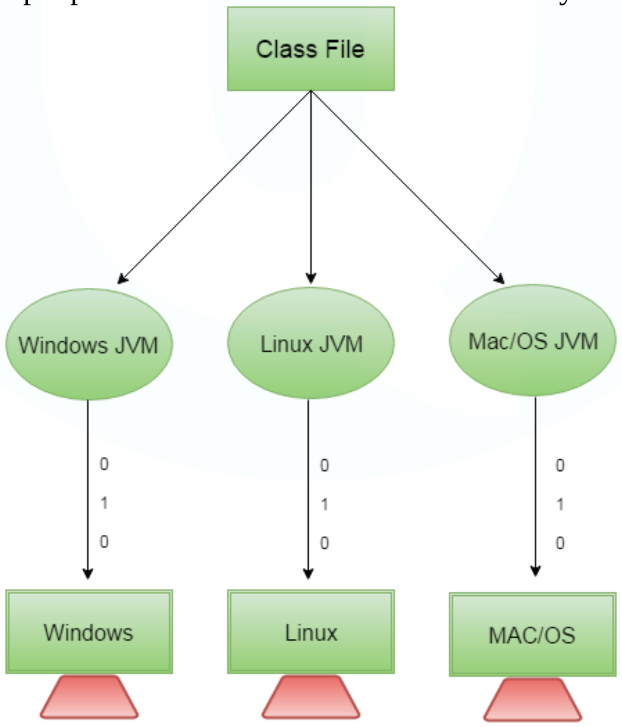
1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

A platform is the hardware or software environment in which a program runs.
There are two types of platforms software-based and hardware-based. Java provides software-based platform.
The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

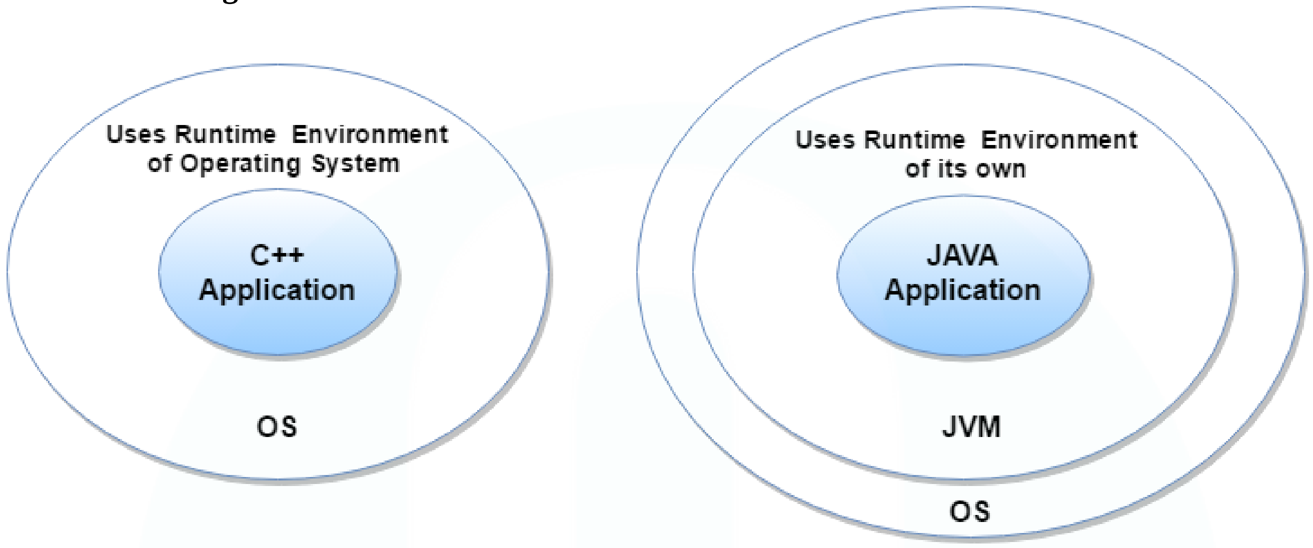
Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).



Secured

Java is secured because:

- **No explicit pointer**
- **Java Programs run inside virtual machine sandbox**


Uses Runtime Environment of Operating System
C++ Application
OS
Uses Runtime Environment of its own
JAVA Application
JVM
OS

- **Classloader:** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, Cryptography etc.

Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is fixed.
In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Portable

We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

C++ vs Java

There are many differences and similarities between C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Goto	C++ supports goto statement.	Java doesn't support goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.

Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.

Difference between JDK, JRE and JVM

Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief overview of JVM here.
If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the basic differences between the JDK, JRE and JVM.

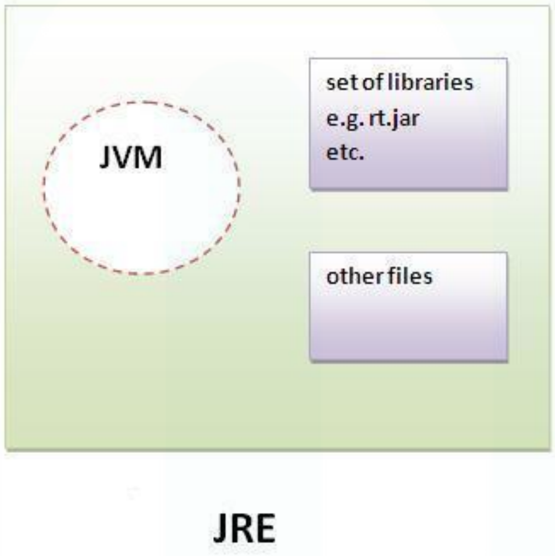
JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed.
JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.
The JVM performs following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

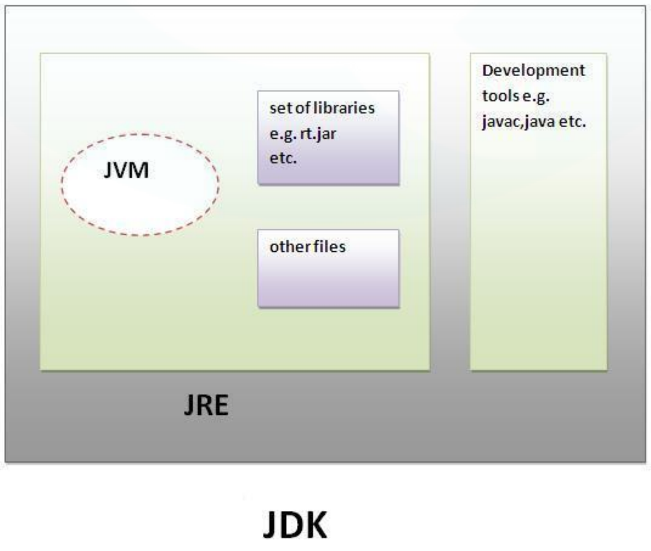
JRE

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.
Implementation of JVMs is also actively released by other companies besides Sun Micro Systems.



JDK

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.


JVM
set of libraries e.g. rt.jar etc.
other files
Development tools e.g. javac,java etc.
JRE
JDK

Module 1 Part II

JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

What is JVM

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

What it does

The JVM performs following operation:

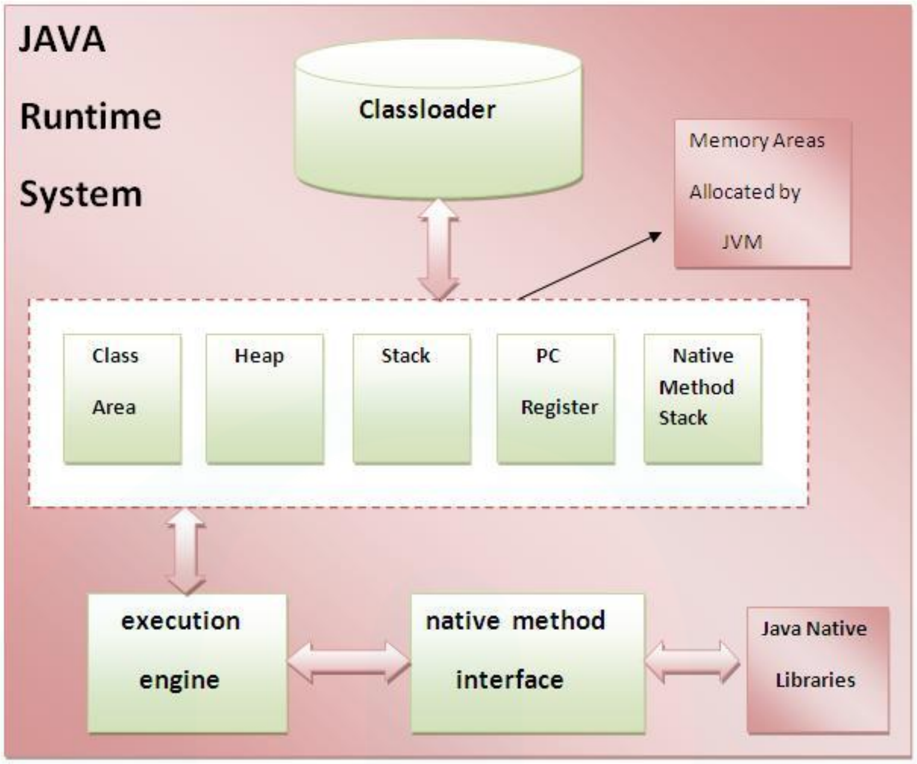
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Internal Architecture of JVM

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



1) Class loader
Class loader is a subsystem of JVM that is used to load class files.

2) Class (Method) Area
Class (Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap
It is the runtime data area in which objects are allocated.

4) Stack
Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register
PC (program counter) registe. It contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack
It contains all the native methods used in the application.

7) Execution Engine

It contains:

1) A virtual processor

2) Interpreter: Read bytecode stream then execute the instructions.

3) Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

Simple Program of Java

```
class Simple
{
        public static void main(String args[])
        {
                System.out.println("Hello Java");
        }
}
```

save this file as Simple.java

To compile: javac Simple.java

To execute: java Simple

Understanding first java program

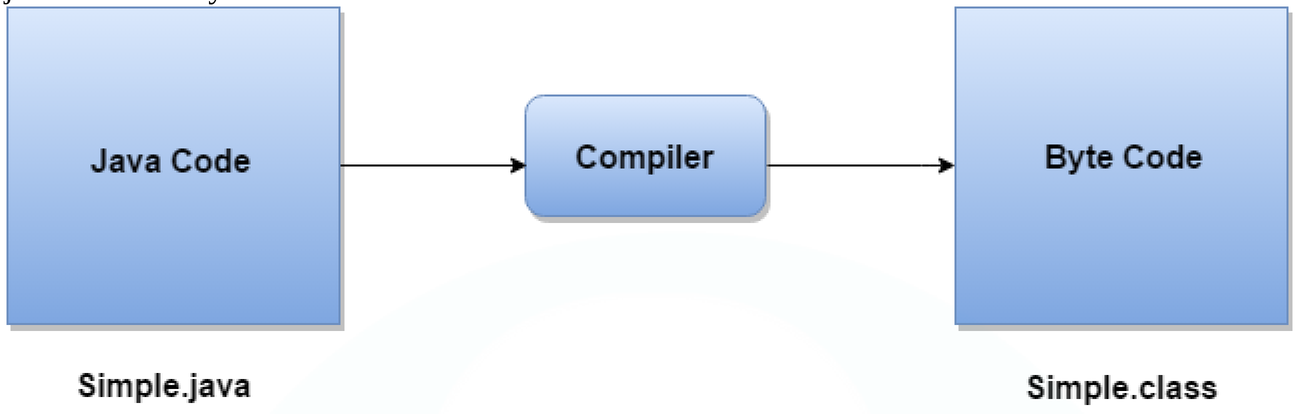
Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

Module 1 Part II

What happens at compile time?

At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



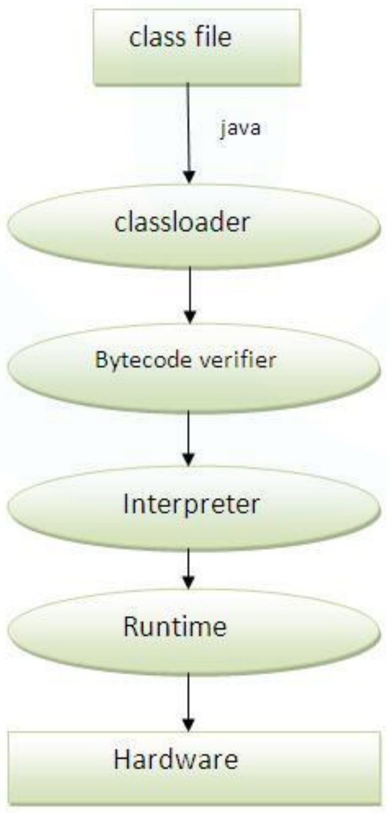
What happens at runtime?

At runtime, following steps are performed:

Classloader: is the subsystem of JVM that is used to load class files.

Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.

Interpreter: read bytecode stream then execute the instructions.



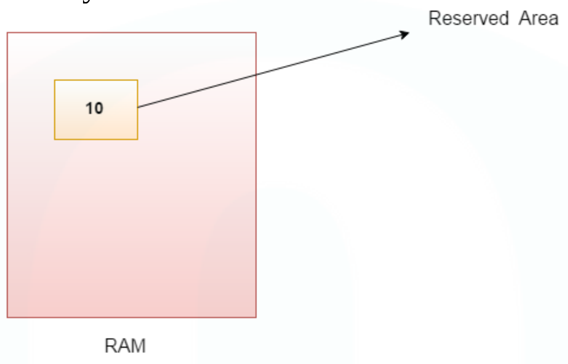
Module 1 Part II

Variables and Data Types in Java

Variable is a name of memory location. There are three types of variables in java: local, instance and static.
There are two types of data types in java: primitive and non-primitive.

Variable

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

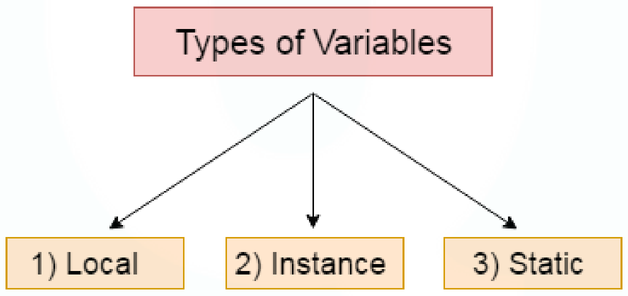


int data=50;//Here data is variable

Types of Variable

There are three types of variables in java:

- local variable
- instance variable
- static variable



1) Local Variable

A variable which is declared inside the method is called local variable.

2) Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

3) Static variable

A variable that is declared as static is called static variable. It cannot be local.

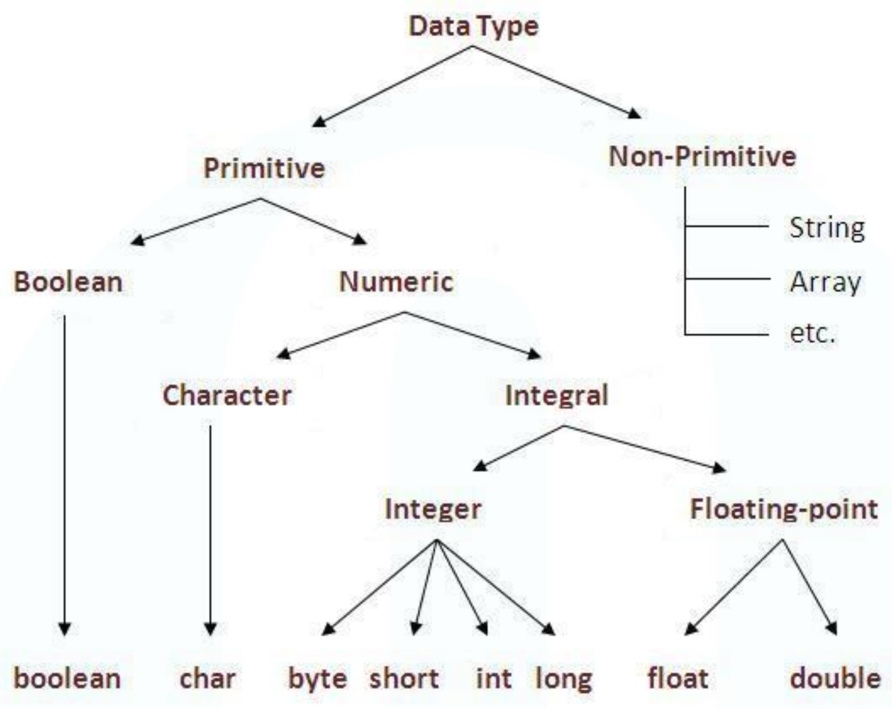
We will have detailed learning of these variables in next chapters.

Module 1 Part II

Data Types in Java

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types



Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

```java
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

```
15
5
50
2
0
```

Java If-else Statement

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

Module 1 Part II

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

Java IF Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
if(condition){
//code to be executed
}
```

```
public class IfExample {
public static void main(String[] args) {
    int age=20;
    if(age>18){
        System.out.print("Age is greater than 18");
    }}}
```

Output:

```
Age is greater than 18
```

Java IF-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

```
public class IfElseExample {
public static void main(String[] args) {
    int number=13;
    if(number%2==0){
        System.out.println("even number");
    }else{
        System.out.println("odd number");
}}}
```

Output:

```
odd number
```

Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
```

```
else{
//code to be executed if all the conditions are false
}
```

```
public class IfElseIfExample {
public static void main(String[] args) {
   int marks=65;
   if(marks<50){
      System.out.println("fail"); }
   else if(marks>=50 && marks<60){
      System.out.println("D grade");  }
     else if(marks>=60 && marks<70){
      System.out.println("C grade"); }
   else if(marks>=70 && marks<80){
      System.out.println("B grade");  }
   else if(marks>=80 && marks<90){
      System.out.println("A grade");}
   else if(marks>=90 && marks<100){
      System.out.println("A+ grade");  }
   else{  System.out.println("Invalid!");
   }}}
```

Output:

```
C grade
```

Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

Syntax:

```
switch(expression)
{
case value1: //code to be executed;
  break;  //optional
case value2: //code to be executed;
   break;  //optional
......
   default:
  code to be executed if all cases are
not matched;
}
```

```
public class SwitchExample {
public static void main(String[] args) {
   int number=20;
   switch(number){
   case 10: System.out.println("10");break;
   case 20: System.out.println("20");break;
   case 30: System.out.println("30");break;
   default:System.out.println("Not in 10, 20 or 30")
;    }}}
```

Output:

```
20
```

Java for Loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

Syntax:

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

```
public class ForExample {
public static void main(String[] args) {
   for(int i=1;i<=10;i++){
      System.out.println(i);
   }}}
```

Java While Loop

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition){
//code to be executed
}
```

```
public class WhileExample {
public static void main(String[] args) {
   int i=1;
   while(i<=10){
      System.out.println(i);
   i++;
   }}}
```

Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop. The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed
}while(condition);
```

```
public class DoWhileExample {
public static void main(String[] args) {
   int i=1;
   do{
      System.out.println(i);
   i++;
   }while(i<=10);}}
```

Java Break Statement

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Syntax:

```
jump-statement;
break;
```

```
public class BreakExample {
public static void main(String[] args) {
   for(int i=1;i<=10;i++){
      if(i==5){
         break;
      }
      System.out.println(i);  } }
```

```
Output:1 2 3 4
```

Java Continue Statement

The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.
Syntax:

```
jump-statement;
continue;
```

```
public class ContinueExample {
public static void main(String[] args) {
   for(int i=1;i<=10;i++){
      if(i==5){
         continue;
      }
      System.out.println(i);
} } }
```

```
Output:1 2 3 4 6 7 8 9 10
```