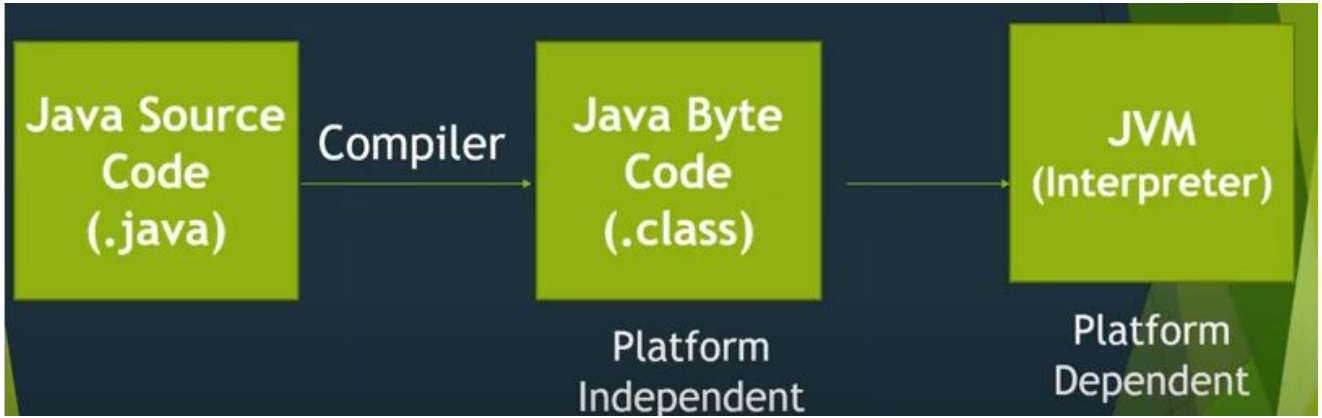
JAVA Environment

- Java Environment includes a large number of development tools, hundreds of classes and methods.
- The development tools are part of the system known as Java Development Kit (JDK).
- The classes and methods are part of the Java Standard Library (JSL) also called as Application Programming Interface (API).
- The Java Development Kit comes with a collection of tools that are used for developing and running Java Programs.

Standard JDK Tools

- **appletviewer** – enables to run Java applets.
- **javac** – The Java compiler, translates Java source code to byte code files that the interpreter can understand.
- **java** – Java interpreter which runs applets and applications by reading and interpreting bytecode files.
- **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 program description.
- **jdb** – Java debugger, helps us to find errors in programs.

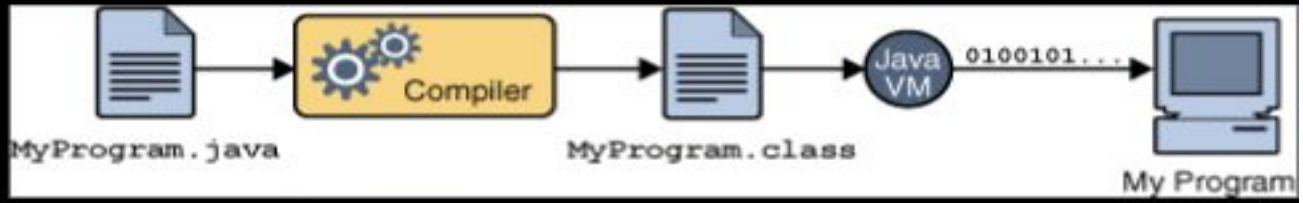
Java Bytecode



- Bytecode is a set of instructions designed to be executed by Java Runtime system
- The Java Virtual Machine (JVM) is the execution engine for Java bytecode. The bytecode is interpreted and executed by the JVM

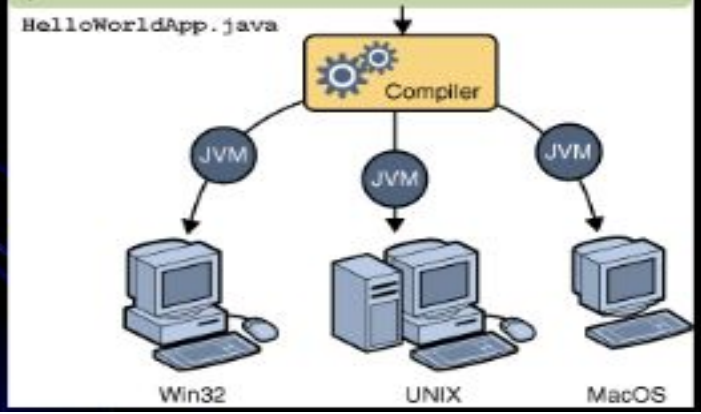
- JVM must be implemented for each platform, bytecode is platform independent
- When the JVM interprets the bytecode it makes security checks
- The JVM use a Just-in-time (JIT) compiler to compile bytecode into native code
- JIT compiler is a part of the JVM, selected portion of the bytecode are compiled into executable code in real time, on a piece-by-piece, demand basis
- JIT compiler compiles code as it is needed during execution

JVM (Java Virtual Machine)



Java Program

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



Java Client/Server equation

- **Applets - Client Side:**
  - special kind of Java program that is designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser.
  - An applet is downloaded on demand.
  - typically used to display data provided by the server, handle user input, or provide simple functions, such as a loan calculator, that execute locally, rather than on the server.

Java Client/Server equation

- **Servlets - Server Side:**
  - a small program that executes on the server.
  - Servlets are used to create dynamically generated content that is then served to the client.
  - servlets are compiled into bytecode and executed by the JVM, so they are highly portable.
  - Eg: price update in online shopping portals, authenticating online banking transactions etc.

Unit I

An Overview of Java

Topics Covered

1. Object-Oriented Programming
2. Two Paradigms
3. Abstraction
4. The Three OOP Principles
5. A First Simple Program
6. Entering the Program
7. Compiling the Program
8. A Closer Look at the First Sample Program

Object-Oriented Programming

- Object-oriented programming (OOP) is at the core of Java
- All Java programs are to at least some extent object-oriented
- Object-oriented programming is about modeling a system as a collection of objects, where each object represents some particular aspect of the system
- Objects contain both functions (or methods) and data (or properties)

Example:

A car is a object with following properties and methods

Properties
Model
Year
Price
Color

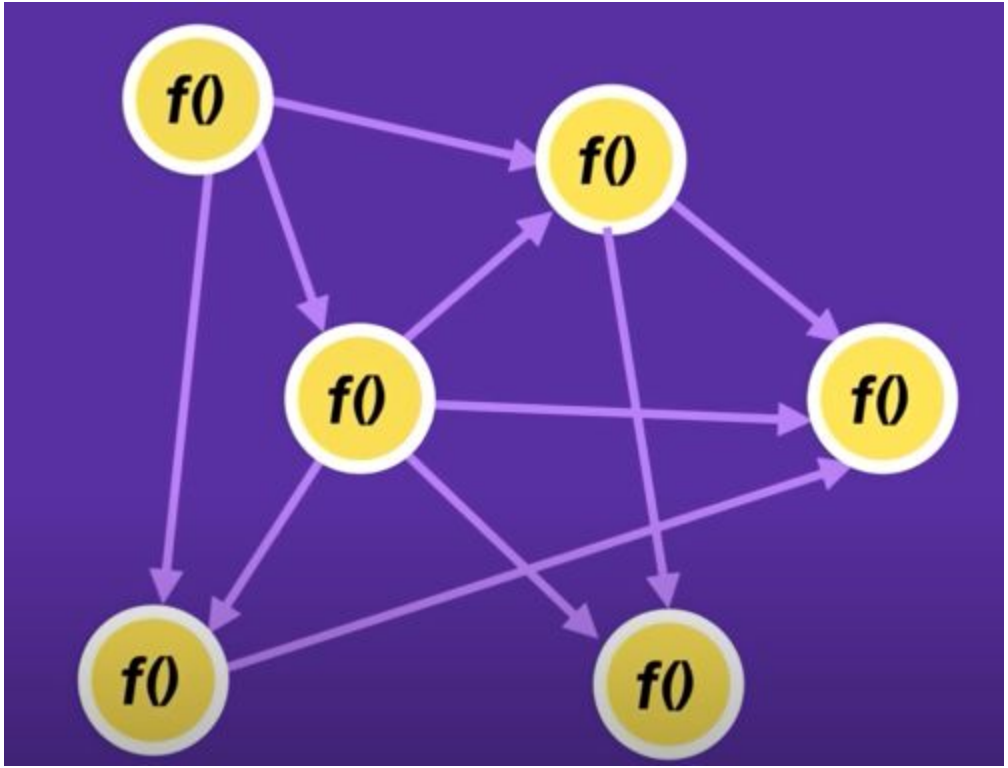
Methods
Start()
Drive()
Reverse()
Stop()

Two Paradigms

All computer programs consist of two elements: code and data. Furthermore, a program can be conceptually organized around its code or around its data. That is, some programs are written around “what is happening” and others are written around “who is being affected.” These are the two paradigms that govern how a program is constructed.

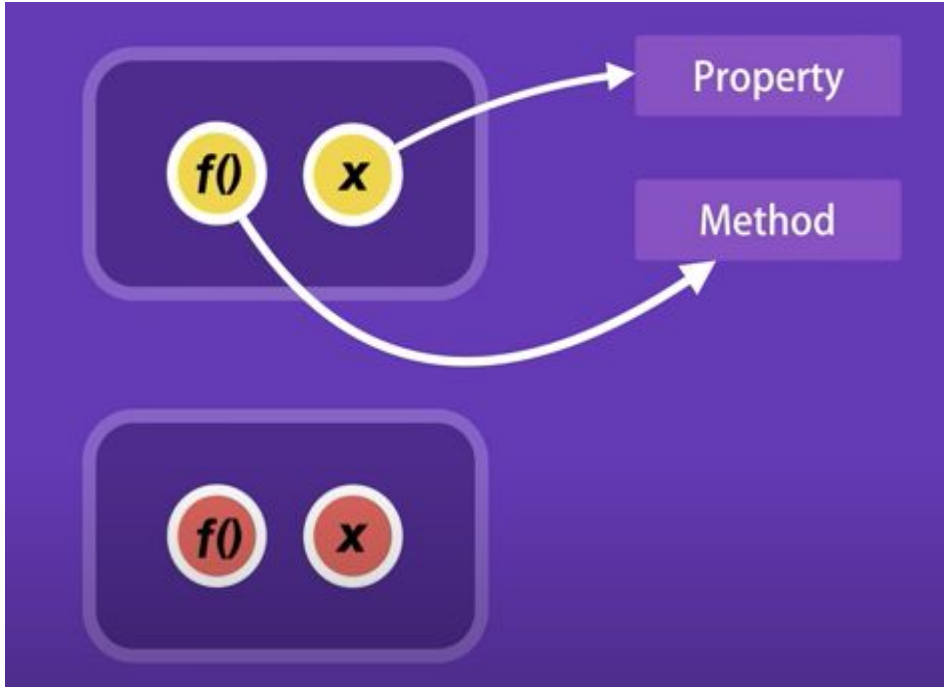
The first way is called the **process-oriented model**

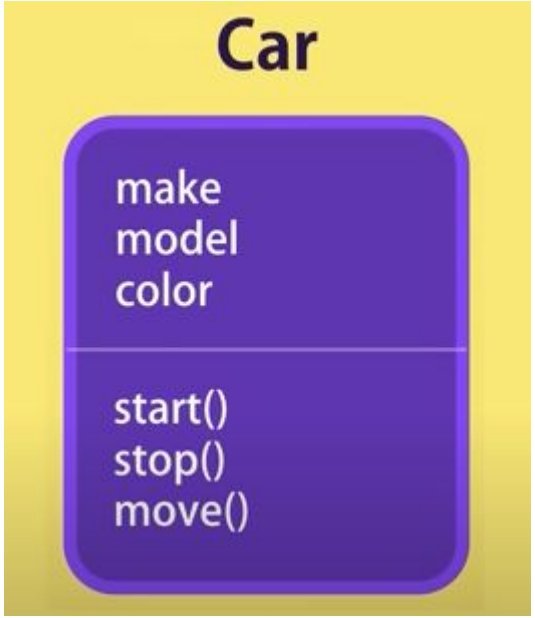
- This approach characterizes a program as a series of linear steps (that is, code). The process-oriented model can be thought of as code acting on data.
- Procedural languages such as C employ this model to considerable success.
- However problems with this approach appear as programs grow larger and more complex


f0
f0
f0
f0
f0
f0

To manage increasing complexity, the second approach, called **object-oriented programming**, was conceived

- Object-oriented programming organizes a program around its data (that is objects) and a set of well-defined interfaces to that data.
- An object-oriented program can be characterized as data controlling access to code.


Property
Method
f0
x
f0
x


Car
make
model
color
start()
stop()
move()

Abstraction

- An essential element of object-oriented programming is abstraction.
- Abstraction is the act of representing the essential feature without knowing the background details.
- Humans manage complexity through abstraction.

  For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior. This abstraction allows people to use a car to drive without being overwhelmed by the complexity of the parts that form the car. They can ignore the details of how the engine, transmission and braking systems work. Instead, they are free to utilize the object as a whole.


Using Abstraction
Tutorial4us
Without Abstraction
Activate Windows

- A powerful way to manage abstraction is through the use of hierarchical classifications.
- This allows to layer the semantics of complex systems, breaking them into more manageable pieces.
- From the outside, the car is a single object. Once inside, we see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone and so on. In turn, each of these subsystems is made up of more specialized units. For instance, the sound system consists of a radio, a CD player, a tape or MP3 player.
- The point is that we manage the complexity of the car (or any other complex system) through the use of hierarchical abstractions.

The Three OOP Principles

All object-oriented programming languages provide mechanisms that help to implement the object-oriented model. They are

- Encapsulation
- Inheritance
- Polymorphism

Encapsulation

- Encapsulation is the mechanism that binds together code and the data it manipulates and keeps both safe from outside interference and misuse.
- One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper.
- Access to the code and data inside the wrapper is tightly controlled through a well-defined interface.

- In Java, the basis of encapsulation is the class.
- A class defines the structure and behavior (data and code) that will be shared by a set of objects.
- Each object of a given class contains the structure and behavior defined by the class.
- For this reason, objects are sometimes referred to as instances of a class.
- Thus, a class is a logical construct; an object has physical reality.

- When we create a class, we will specify the code and data that constitute that class. Collectively, these elements are called members of the class.

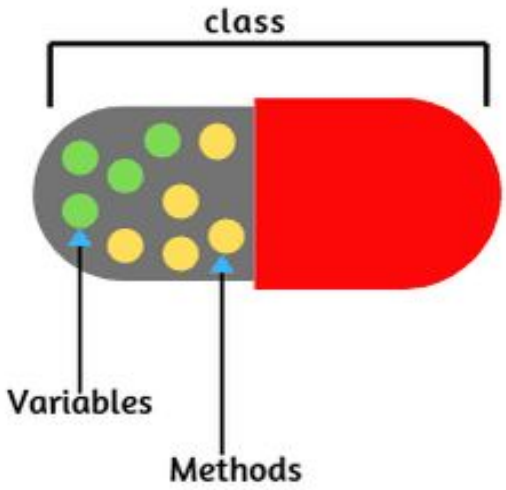
- Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods.

- In properly written Java programs, the methods define how the member variables can be used. This means that the behavior and interface of a class are defined by the methods that operate on its instance data.

```
class
{

  data members
        +
  methods (behavior)

}
```



Encapsulation in Java

- Each method or variable in a class may be marked private or public.

- The public interface of a class represents everything that external users of the class need to know, or may know.

- The private methods and data can only be accessed by code that is a member of the class.

- Therefore, any other code that is not a member of the class cannot access a private method or variable.

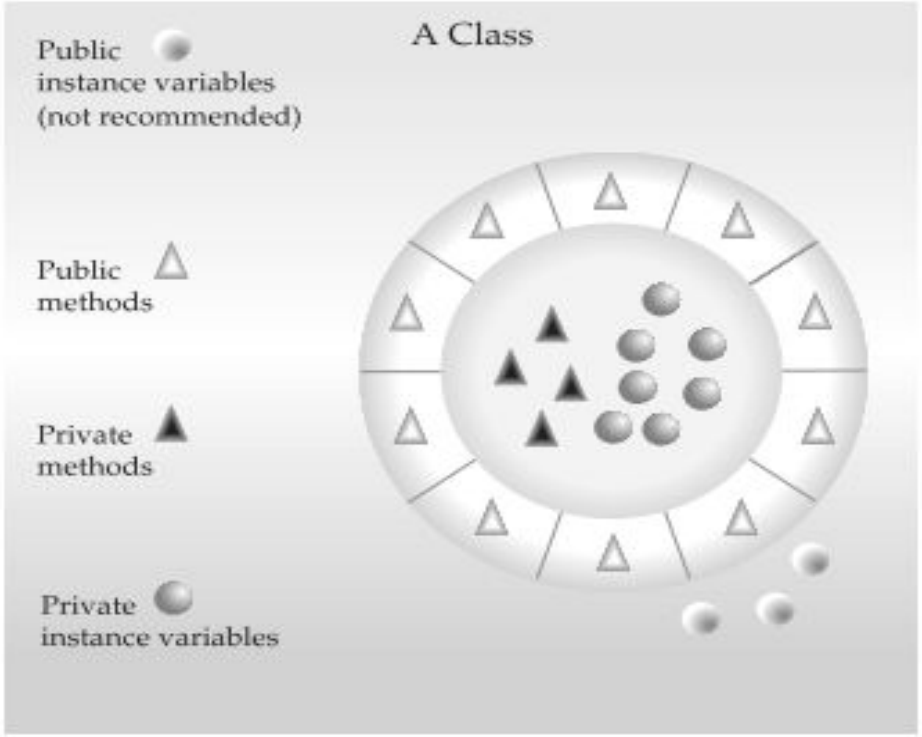


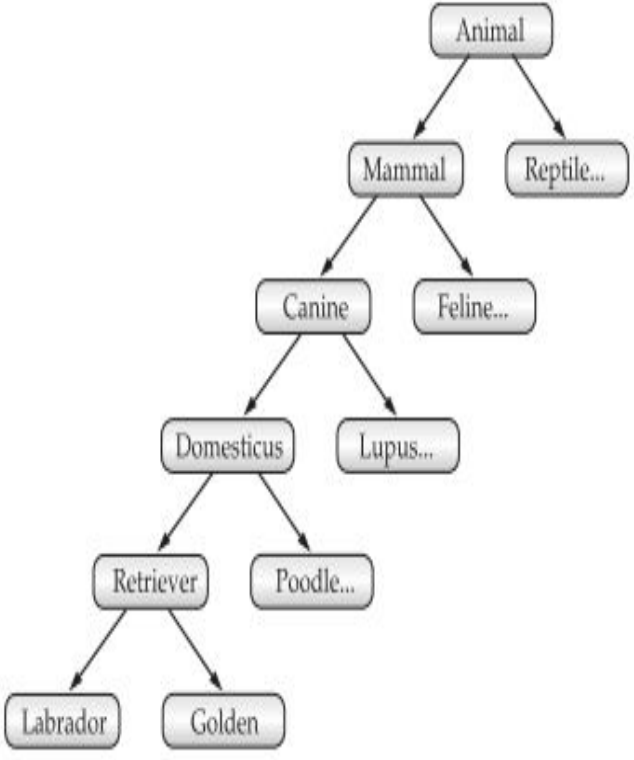
Fig: Encapsulation: public methods can be used to protect private data

Inheritance

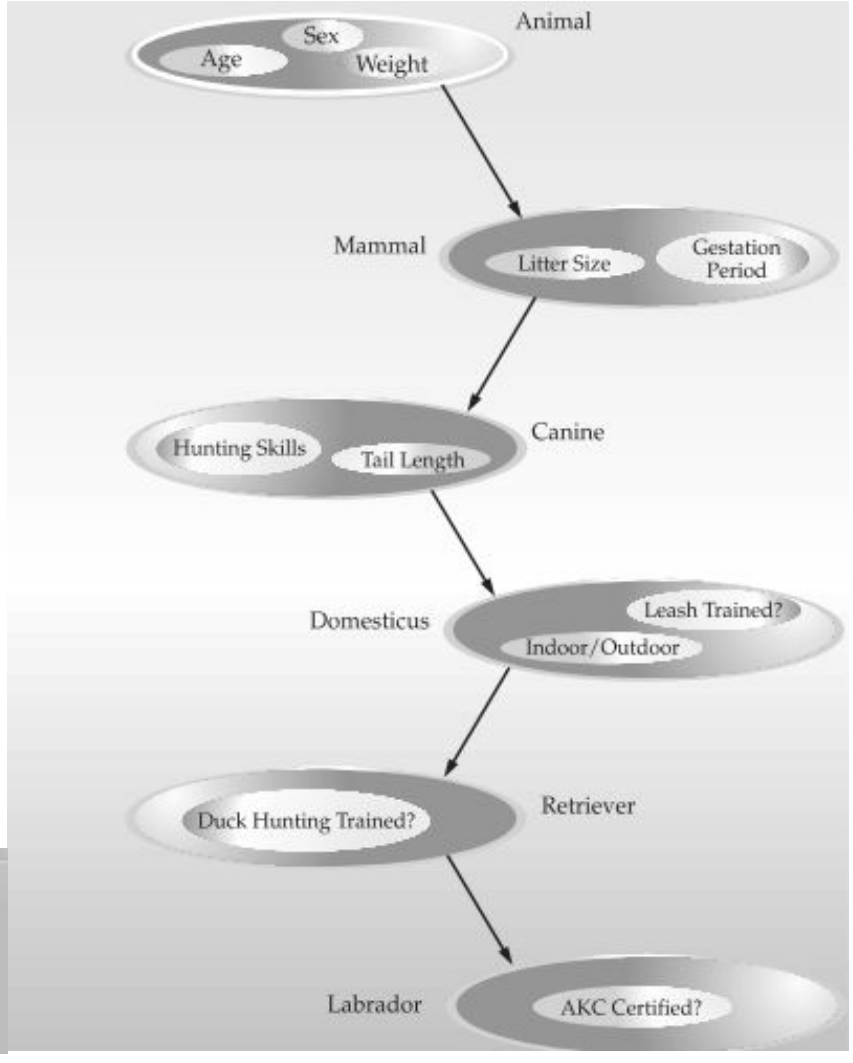
- Inheritance is the process by which one object acquires the properties of another object.
- This is important because it supports the concept of hierarchical classification.
- For example, a Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal.
- Without the use of hierarchies, each object would need to define all of its characteristics explicitly.
- However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent.
- Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

Example of Inheritance

Consider objects that are related to each other in a hierarchical way such as animals, mammals and dogs. Animals have some attributes such as size, intelligence and type of skeletal system. Animals also have certain behavioral aspects like they eat, breathe and sleep. This description of attributes and behavior is the class definition for animals. A more specific class of animals, such as mammals, they would have more specific attributes, such as type of teeth and mammary glands. This is known as a subclass of animals, where animals are referred to as mammals' superclass. Since mammals are simply more precisely specified animals, they inherit all of the attributes from animals. A deeply inherited subclass inherits all of the attributes from each of its ancestors in the class hierarchy.



Labrador		
Age	Gestation Period	Leash Trained?
Sex	Hunting Skills	Duck Hunting Trained?
Weight	Tail Length?	AKC Certified?
Litter Size	Indoor / Outdoor?	



Polymorphism

- Polymorphism (from Greek, meaning ‘many forms’) is a feature that allows one interface to be used for a general class of actions.
- The specific action is determined by the exact nature of the situation.
- Consider a stack (which is a last-in, first-out list). We might have a program that requires three types of stacks. One stack is used for integer values, one for floating-point values and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs.
- In a non-object-oriented language, we would be required to create three different sets of stack routines, with each set using different names.
- However, because of polymorphism, in Java we can specify a general set of stack routines that all share the same names.

- Polymorphism is expressed by the phrase “one interface, multiple methods”
- This means that it is possible to design a generic interface to a group of related activities.
- This helps reduce complexity by allowing the same interface to be used to specify a general class of action.
- It is the compiler’s job to select the specific action (that is, method) as it applies to each situation. The programmer, do not need to make this selection manually.
- A dog’s sense of smell is polymorphic. If the dog smells a cat, it will bark and run after it. If the dog smells its food, it will salivate and run to its bowl.
- The same sense of smell is at work in both situations. The difference is what is being smelled, that is, the type of data being operated upon by the dog’s nose.

Simple program……

```
/* the first program.*/

class Example
{
        public static void main(String args[])
        {
                System.out.println("Hello, World!");
        }
}
```

Compiling the Example program……

- Compile theprogram using thejava compiler. The compiler will produce an output file Example.class

To compile the program:

```
C:\> javac Example.java
```

Running the Example program…..

- Run the example program by:

```
C:\> java Example
```

- This will run the Example program and generate

```
Hello, World!
```

as the output

A closer look at the Example program…….

- // → This is a single line comment.
- /* This is a multi-line comment. */
- Class Example {

  This line states that the lines between the {} pair defines a class named Example

.....

public static void main(String args[]) {

- This defines the main method.
- This method will be called by the JVM when the class Example is specified on the command line.
- 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.

.....

System.out.println("Hello, World!");

- This calls the println() method in the System class.
- It is used to output text on the console.
- We can use println() to output text and have a new line appended or use print() to output text without the new line.
- The println method is a member of the out object, which is a static data member of System class.
- The statement ends with a semicolon.

.....

Public :

- The keyword public is an access specifier that declares the main method as unprotected and therefore making it accessible to all other classes.

Static :

- The keyword static 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.

.....

Void :

- The type modifier void states that the main method does not return any value (but simply prints some text to the screen).

Main :

- main() is the method called when a java application begins.
- main is different from Main.
- Java compiler also compiles the class that do not contain main() method.

.....

String args[]

- Only one parameter in main(), but complicated one.
- String args[] declares a parameters named args, which is an array of instances of the class String.
- args receive any command-line arguments information, when the program is executed.

A second Short Example……

```
class Example2
{
        public static void main(String args[])
        {       int num;    // This declares an integer variable
                num = 100;  // now initialize num to 100

                System.out.println("num = " + num);

                num = num * 2;// multiply num by 2

                System.out.println("Now num = " + num);

        }
}
```

```
class IfSample
{
  public static void main(String args[])
  {
    int x = 10, y = 20;     // declare and initialize x and y
     if (x < y)
       System.out.println("x is less than y");
    else if(x == 10)
       System.out.println("x is equal to 10");
    else
       System.out.println("x is more than y");
  }
  }
```

for example….

```
class ForDemo
{
        public static void main(String args[])
        {
            int count;
            for(count = 0; count < 5; count = count+1)
            System.out.println("This is count: " +count);
            System.out.println("Done!");
        }
}
```

Using blocks of code…

- ***Code blocks*** are enclosed in { } curly braces.

```
if (w < h)
{
        v = w * h;
        w = 0;
}
```

Whitespace

- Java is a free-form language. This means we do not need to follow any special indentation rules.
- For ex. A program can be written in a single line.
- Whitespace characters are space, tab or newline.

Identifiers

- Identifiers are used for **class names**, **method names** and **variables**.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- Identifiers cannot begin with a number.
- Identifiers are case sensitive, so VALUE is different from Value.
- Valid identifiers : AvgTemp, count, a4, $test….
- Invalid identifiers : 2count, high-temp, Not/ok…

Literals...

- There are many different types of literals or constants.
- **Integer** – numbers like 45.
- **Floating point** – numbers like 98.6
- **Character** – character literals are represented with opening and closing single quote characters. For example: 'a' or 'Z'
- **String** – String literals are represented with opening and closing double quotes.
  For example: "This is a String!"

Comments…

- // indicates a single line comment.
- /* begins a comment that must be terminated with */
- Documentation comments begin with /** and end with */
- Documentation comments areused to insert documentation into code.
- These comments are then used to produce documentation by javadoc .

Separators....

- ( ) – parenthesis, used to enclose parameters for methods or to define precedence.
- { } – braces, used to contain blocks of code such as classes, methods and any group of statements that must act a single unit.
- [ ] – brackets, declare and operate on arrays.
- ; - used to terminate statements.
- , - used to separate identifiers in a variable declaration.
- . – used to separate package names, used to separate a variable or method name from its class.

Java Class Libraries…

- Java has a wide array of standard classes that are available for every programmer to use.
- The Java class libraries provide facilities for IO, string handling, networking and graphics.
- The class Libraries save you from having to write base functionality yourself.

The simple Types…..

- Java defines **eight** simple types of data: byte, short, int, long, char, float, double & Boolean. These can be **put in four groups**……..

- Integers : This group includes byte(8), short(16), int(32) & long(64), which are for whole valued signed numbers.

- Floating-point numbers : Includes float & double, which represents numbers with fractional precision.

- Characters : Includes char, which represents symbols in a character set, like letters and numbers.

- Boolean : Includes Boolean, which represents true/false.

Variables…..

- The variable is thebasic unit of storage in a java program.
- A variable is defined by the combination of an identifier, a type, and an optional initializer.
- All variables have a scope, which defines their visibility and lifetime.

Variable declarations….

- int intvar; // simple variable declaration
- int i = 0; // declaration with initialization
- int i = 0, j = 0, k = 23; // multiple declarations and initializations on one line
- double PI = 3.14159; // double precision floating point variable declaration with initialization.
- double sqrtOfTwo = Math.sqrt(2); // dynamic initialization, initialization will be performed when the program runs.
- char x = 'x'; // declare and init a character variable.

Dynamic Initialization…

- Java allows variables to be initialized dynamically.

```
class DynInit
{
    public static void main(String args[])
    {
            double a = 3.0, b = 4.0;
                        // c is dynamically initialized
            double c = Math.sqrt(a * a + b * b);
            System.out.println("Hypotenuse is " + c);
    }
}
```

Arrays…..

- **An array is a group of like - typed variable that are referred to by a common name.**

- Arrays of any type can be created and may have one or more dimensions.

- A specific element in an array is accessed by its **index**.

- The principal advantage of an array is that it organizes data in such a way that it can be easily manipulated.

One-Dimensional Arrays….

- A one-dimensional array is a list of related variables.
- Syntax to declare a one-dimensional array:

  type var_name[ ]; OR type[ ] var-name;

  array-var = new *type[size];*

new is a special operator that allocates memory

Ex : int a[ ];

a = new int[10];

type array-name[ ] = new type[size];

Ex: int a[ ] = new int[10];

Example program…..

```
class Arraytrials
{
        public static void main(String args[])
        {
         int sample[] = new int[10];
         int i;
         for(i = 0; i < 10; i = i+1)
           sample[i] = i;
        for(i = 0; i < 10; i = i+1)
         System.out.println("This is sample[" + i + "]:" +sample[i]);
        }
}
```

- Arrays can be **initialized when they are declared**.
- The process is much the same as that used to initialize the simple types.
- An array initializeris a listof comma – separated expressions surrounded by curly braces.
- The commas separate the values of the array elements.
- The array will automatically be created large enough to hold the number of elements you specify in the array initializer.

Example program……

```
class Average
{
        public static void main(String args[])
        {
                double nums[ ] = { 10.1, 11.2, 12.3, 23.4, 22.4};
                double result = 0;
                int i;
                for(i = 0; i < 5; i++)
                   result = result +nums[i];
                System.out.println(" Average is " + result/5
                );
        }
}
```

Multi-dimensional Arrays….

- **In Java multidimensional arrays are actually arrays of array.**
- These look and act like regular multidimensional arrays, but there are a couple of subtle differences.
- To declare a multidimensional array

```
int twoD[ ][ ];
twoD = new int[4][5];
```

or

```
int twoD[ ][ ] = new int[4][5];
```

Example program…

```
class TwoDArray
{
        public static void main(String args[])
        {
                int twoD [ ] [ ] = new int [ 4] [ 5];
                int i, j, k = 0;
                for(i=0; i<4; i++)
                   for(j=0; j<5; j++)
                   {
                      twoD[i] [j] = k;
                      k++;
                   }
                for(i=0; i<4; i++)
                {
                   for(j=0; j<5; j++)
                      System.out.print(twoD[i][j] + " ") ;
                   System.out.println( );
                }
        }
}
```

CONTROL STATEMENTS

```
// Demonstrate the switch.
class SwitchDemo {
      public static void main(String args[]) {
            int i;
            for(i=0; i<10; i++)
                  switch(i) {
                        case 0: System.out.println("i is zero");
                        break;
                        case 1: System.out.println("i is one");
                        break;
                        case 2: System.out.println("i is two");
                        break;
                        case 3: System.out.println("i is three");
                        break;
                        case 4: System.out.println("i is four");
                        break;
                        default: System.out.println("i is five or more");
            }
      }
}
```

```
// Demonstrate the while loop.
class While {
  public static void main(String args[]) {
    int n = 10;

    while(n > 0) {
      System.out.println("tick " + n);
      n--;
    }
  }
}
```

Demonstrate for loop

```
// Test for primes.
class FindPrime {
  public static void main(String args[]) {
    int num;
    boolean isPrime = true;

    num = 14;
    for(int i=2; i <= num/i; i++) {
      if((num % i) == 0) {
        isPrime = false;
        break;
      }
    }
    if(isPrime) System.out.println("Prime");
    else System.out.println("Not Prime");
  }
}
```

The For-Each version of the for loop….

- The for-each version of the for is also refered to as the *enhanced* for loop.
- Syntax:

  for(type itr-var : collection) statement-block

- ***type*** specifies the type.
- ***itr-var*** specifies the name of an iteration variable that will receive the elements from a collection.
- The collection being cycled through is specified by ***collection***.

- There are various types of collections that can be used with the for, ex. is array.
- **With each iteration of the loop, the next element in the collection is retrieved and stored in *itr-var*.**
- **The loop repeats until all elements in the collection have been obtained.**

```
int nums[ ]={1,2,3,4,5,6,7,8,9,10};
int sum=0;
for(int x: nums) sum=sum+x;
```

```
class ForEach
{
  public static void main(String args[])
  {
        int nums[]={1,2,3,4,5,6,7,8,9,10};
        int sum=0;
        for(int x: nums)
        {
          System.out.println("Value is :"+x);
          sum=sum+x;
        }
        System.out.println("Summation : "+sum);
  }
}
```