***object-oriented programming (OOP)* :** is a programming methodology that helps organize complex programs through the use of inheritance, encapsulation, and polymorphism.

All computer programs consist of two elements: code and data. Furthermore, a program can be conceptually organized around its code or around its data.

- If a program organized around its code and are written around "what is happening" this methodology called **Procedure Oriented programming.**
- If a program organized around its data and are written around "who is being affected.", this methodology called **Object Oriented programming.**

The following table summarizes the main differences between the two methodologies that govern how a program is constructed:

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	Program is divided into small parts called **functions**.	Program is divided into parts called **objects**.
Importance	Importance is not given to data but to **functions**	Importance is given to the **data** rather than functions(methods)
Approach	Follows **Top Down approach**.	Follows **Bottom Up approach**.
Access Specifiers	does not have any access specifier.	has access specifiers named Public, Private, Protected, etc.
Data Moving	data can move freely from function to function	objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	provides an easy way to add new data and function.
Data Access	most function uses global data for sharing that can be accessed freely from function to function.	data cannot move easily from function to function, it can be kept public or private so we can control the access of data.
Data Hiding	does not have any proper way for hiding data so it is **less secure**.	provides data hiding so provides **more security**.
Overloading	overloading is not possible.	overloading is possible
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Abstraction

An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. A powerful way to manage abstraction is through the use of **hierarchical classifications.**

hierarchical classifications : allows you to layer the semantics of complex systems, breaking them into more manageable pieces.
Hierarchical abstractions of complex systems can also be applied to computer programs. The **data** from a traditional process-oriented program can be **transformed by abstraction** into its **component objects**. A sequence of process steps can become a collection of messages between these objects. Thus, each of these objects describes its own unique behavior. You can treat these objects as concrete entities that respond to messages telling them to do something. This is the essence of object-oriented programming.

The Three OOP Principles

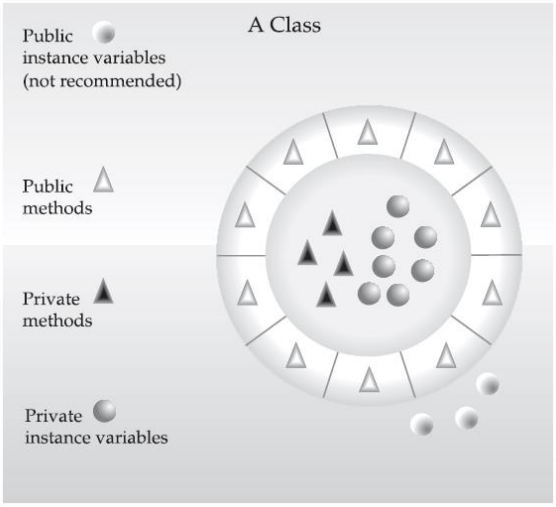
1. Encapsulation,
2. Inheritance,
3. Polymorphism.

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.**

The power of encapsulated code is that everyone knows how to access it and thus can use it regardless of the implementation details—and without fear of unexpected side effects. **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, as if it were stamped out by a mold in the shape of 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.**

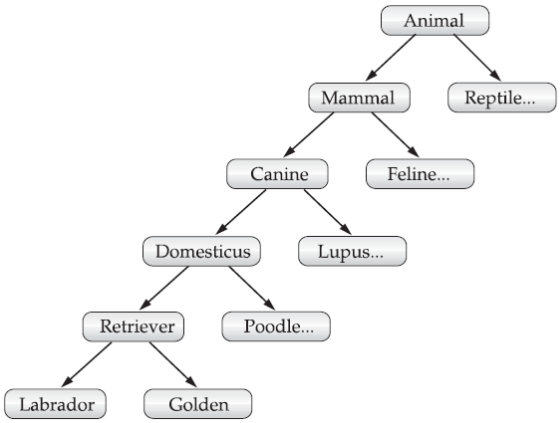
When you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called members of the class. Specifically, **the data** defined by the class are referred to as **member variables or instance variables**. The code that operates on that data is referred 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. Since the purpose of a class is to encapsulate complexity, there are mechanisms for hiding the complexity of the implementation inside the class. 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. Since the private members of a class may only be accessed by other parts of your program through the class' public methods, you can ensure that no improper actions take place. This means that the public interface should be carefully designed not to expose too much of the inner workings of a class.



Example:
Most people naturally view the world as made up of objects that are related to each other in a **hierarchical way**, such as animals, mammals, and dogs. If you wanted to describe animals in an abstract way, you would say they have some attributes, such as size, intelligence, and type of skeletal system. Animals also have certain behavioral aspects; they eat, breathe, and sleep. This description of attributes and behavior is the class definition for animals.
If you wanted to describe 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.

Inheritance interacts with **encapsulation** as well. If a given class encapsulates some attributes, then any subclass will have the same attributes plus any that it adds as part of its specialization.

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.**

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.**

Example :

Consider a stack (which is a last-in, first-out list). You 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, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names.

The concept of **polymorphism** is often expressed by the phrase **"one interface, multiple methods."** This means design a generic interface to a group of related activities, that 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. You, the programmer, do not need to make this selection manually.

Polymorphism, Encapsulation, and Inheritance Work Together

A well-designed **hierarchy of classes** is the basis for reusing the code in which you have invested time and effort developing and testing. **Encapsulation** allows you to migrate your implementations over time without breaking the code that depends on the public interface of your classes. **Polymorphism** allows you to create clean, sensible, readable, and resilient مرن code.

A First Simple Program

```
/*
This is a simple Java program.
Call this file "Example.java".
*/
class Example {
        // Your program begins with a call to main().
    public static void main(String args[]) {
          System.out.println("This is a simple Java program.");
        }
}
```

The output generated by this program is shown here:

```
This is a simple Java program.
```

Important Notes

About Java programs, it is very important to keep in mind the following points.

- Case Sensitivity - Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.
- Class Names - For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Example: class MyFirstJavaClass.
- Method Names - All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. Example: public void myMethodName() .
- Main class – every Java program must contain in a main class, all code must reside inside this class, it compose a block of statements, so, it must begin with { and end with } .
- Program File Name - Name of the program file should exactly match the class name. When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile). Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'
- public static void main(String args[]) - Java program processing starts from the main() method which is a mandatory part of every Java program. Main method must begin with { and end with }, because it contains a block of statements in it.
- Every statement in java program must end with ;. And any block must surrounded with { and }.
- The **public** keyword is **an access specifier**, which allows the programmer to control the visibility of class members. Public means that member may be accessed by code outside the class in which it is declared. (The opposite of public is private, which prevents a member from being used by code defined outside of its class.)
- The keyword **static** allows main( ) to be called without having to instantiate a particular instance of the class.

A Second Short Program

```
/*
Here is another short example.
Call this file "Example2.java".
*/
class Example2 {
        public static void main(String args[]) {
            int num;            // this declares a variable called num
            num = 100;          // this assigns num the value 100
            System.out.println("This is num: " + num);
            num = num * 2;
            System.out.print("The value of num * 2 is ");
            System.out.println(num);
        }
}
```

When you run this program, you will see the following output:

```
This is num: 100
The value of num * 2 is 200
```

A Third Program (IF Statement)

```
/*
Demonstrate the if.
Call this file "IfSample.java".
*/
class IfSample {
        public static void main(String args[]) {
           int x, y;
           x = 10;
           y = 20;
           if(x < y) System.out.println("x is less than y");
           x = x * 2;
           if(x == y) System.out.println("x now equal to y");
           x = x * 2;
           if(x > y) System.out.println("x now greater than y");
          // this won't display anything
          if(x == y) System.out.println("you won't see this");
        }
}
```

The output generated by this program is shown here:

x is less than y
x now equal to y
x now greater than y

A Fourth Program (For Statement)

```
/*
Demonstrate the for loop.
Call this file "ForTest.java".
*/
class ForTest {
        public static void main(String args[]) {
            int x;
            for(x = 0; x<10; x = x+1)
            System.out.println("This is x: " + x);
        }
}
```

This program generates the following output:

This is x: 0
This is x: 1
This is x: 2
This is x: 3
This is x: 4
This is x: 5
This is x: 6
This is x: 7
This is x: 8
This is x: 9

Using Blocks of Code

Java allows two or more statements to be grouped into blocks of code, also called code blocks. This is done by enclosing the statements between opening and closing curly braces.

Example:

Single statement (Non-block)	Code block

if(x < y) x = x * 2;
System.out.println("this is isolated");
	if(x < y) { // begin a block
	  x = y;
	  y = 0;
	} // end of block

A Fifth Program (Block of codes)

```
/*
Demonstrate a block of code.
Call this file "BlockTest.java"
*/
class BlockTest {
        public static void main(String args[]) {
          int x, y;
          y = 20;
          // the target of this loop is a block
         for(x = 0; x<10; x++) {
            System.out.println("This is x: " + x);
            System.out.println("This is y: " + y);
            y = y - 2;
           }
        }
}
```

The output generated by this program is shown here:

```
This is x: 0
This is y: 20
This is x: 1
This is y: 18
This is x: 2
This is y: 16
This is x: 3
This is y: 14
This is x: 4
This is y: 12
This is x: 5
This is y: 10
This is x: 6
This is y: 8
This is x: 7
This is y: 6
This is x: 8
This is y: 4
This is x: 9
This is y: 2
```

Lexical Issues

Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

Whitespace

In Java, whitespace is a space, tab, or newline.

Identifiers

1. Identifiers are used for class names, method names, and variable names.
2. All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
3. After the first character, identifiers can have any combination of characters.
4. They must not begin with a number, lest they be confused with a numeric literal.
5. Most importantly, Java is case-sensitive, so VALUE is a different identifier than Value.

Some examples of valid identifiers are

AvgTemp	count	$test	a4	__1_value	_value	this_is_ok

Invalid identifier names include these:

123abc	2count	-salary	high-temp	Not/ok

Literals

A constant value in Java is created by using a literal representation of it. For example, here are some literals:

100	98.6	'X'	"This is a test"

A literal can be used anywhere a value of its type is allowed.

Comments

A comment describes or explains the operation of the program to anyone who is reading its source code, explain how some part of the program works or what a specific feature does.
Java supports three styles of comments:

- ***Single-line comment.*** begins with a // and ends at the end of the line.
- ***A multiline comment.*** It may be several lines long, it must begin with /* and end with */. Anything between these two comment symbols is ignored by the compiler.
- ***A documentation comment.*** This type of comment is used to produce an HTML file that documents your program.

Separators

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

The Java Keywords

There are 50 keywords currently defined in the Java language (see the following table). These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language. These keywords cannot be used as names for a variable, class, or method.

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

- The keywords **const** and **goto** are reserved but not used.
- Java reserves the following value: **true**, **false**, and **null**. You may not use these words for the names of variables, classes, and so on.

Java Modifiers

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers:

- **Access Modifiers:** default, public , protected, private
- **Non-access Modifiers:** final, abstract, strictfp

Java Variables Types

Following are the types of variables in Java:

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static Variables)