What is Object Oriented Programming?

- Java is an Object Oriented programming language.
- **Object-oriented programming (OOP)** is a programming paradigm that uses "objects" and their interactions to design applications and computer programs.
- A language that supports Object Oriented Programming (OOP) concepts is known as **Object Oriented Programming.**
- **Object-Oriented programming=** Data Abstraction

  +Data Encapsulation+ Overloading + polymorphism + Dynamic Binding+Inheritance.

- There are six qualities to be satisfied for a programming language to be pure **Object Oriented**.
- **They are**:
  - Encapsulation/Data Hiding
  - Inheritance
  - Polymorphism
  - All predefined types are objects
  - All operations are performed by sending messages to objects
  - All user defined types are objects.

OO Languages

- Simula (1967)
- Smalltalk (1980)
- C++ (1983, 1990)
- Object Pascal (1988)
- Lisp CLOS (1989)
- Java (1995, 1997, 1998…)
- Python(1992)
- JavaScript (1997)

- **Object-Oriented Programming with Java**
- **The object paradigm**
- Object-oriented (OO) programming
- Encapsulation, inheritance and polymorphism
- OO analysis and design: "Is a" and "Has a" relationships
- Designing an OO application step by step
- Diagramming object structure with Unified Modeling Language (UML)
- **Java's object-oriented features**
- Instantiating objects from classes
- Aggregation and composition
- Extending existing classes
- Overloading and overriding methods.

Other OO Languages

  - Object Based programming (classes, objects, encapsulation)
  - Object Oriented programming (inheritance, polymorphism)
  - Generic programming (class and function templates)
- **Object oriented** : C++,Java, Smalltalk,DotNet….
- **Object based** : Ada, Ada,Simula67, Eifel.
- Smalltalk
  - pure OO language developed at PARC
- Java
  - built on C/C++
  - objects and data types.

Object Terminology

- Objects -- packet containing data and procedures
- Methods – procedure (or) deliver service .
- Message -- request to execute a method
- Class -- template for creating objects
- Instance -- an object that belongs to a class
- Encapsulation – wrapping the data + code
- Inheritance --allowing the reuse of class spec's class
- Hierarchy -- tree structure inheritance relations
- Polymorphism -- to hide different implementations
- Data Abstraction—Ignoring unwanted details
- Method-Overloading-overloading methods in program.
- **Package**: is a grouping of related types providing access protection and name space management.

OOProgramming Concepts

1. Real-world objects contain **state** and **behavior**.
2. A software object's state is stored in **fields**.
3. A software object's behavior is exposed through **methods**.
4. Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as data **encapsulation**.
5. A blueprint for a software object is called a **class**.
6. Common behavior can be defined in a super class and inherited into a **subclass** using the **extends** keyword.
7. A collection of methods with no implementation is called an **interface**.
8. A namespace that organizes classes and interfaces by functionality is called a **package**.
9. The term **API** stands for Application Programming Interface.

- **Features of OOP**
  1. Programs are divided into objects
  2. functions that operate on the data of an object are together in the data structure.
  3. Data is hidden and can’t be accessed by external functions(Data Security).
  4. Objects may communicate with each other through functions.
  5. New data can be easily added to the functions whenever necessary.
  6. It follows bottom-up approach in program designing.
  7. Information hiding: state, autonomous behavior.
  8. Data abstraction: emphasis on what rather than how.
  9. Dynamic binding: binding at runtime, polymorphism.
  10. Inheritance: incremental changes specialization, reusability.

OOP BENEFITS

- OOP is a proven general-purpose technique
- OOP models the real world better
- OOP makes programming more visual
- OOP makes programming easier and faster
- OOP reduces the number of places that require changing
- OOP increases reuse (recycling of code)
- OOP does automatic garbage-collection better
- OO databases can better store large, multimedia data
- OODBMS are overall faster than RDBMS
- OOP better hides persistence mechanisms.
- OOP eliminates the "complexity" of "case" or "switch" statements.

- OOP models human thought better.
- OOP is more "modular"
- OOP divides up work better
- OOP "hides complexity" better
- OOP better models spoken language
- OOP is "better abstraction"
- OOP reduces "coupling"
- OOP does multi-tasking better
- OOP scales better
- OOP is more "event driven"
- OOP manages behavior better
- Only OOP offers automatic initialization
- OOP would have prevented more Y2K problems
- OOP "does patterns" better
- Only OOP can "protect data" .

OOP Characteristics

Data Abstraction: Abstraction is of the process of hiding unwanted details from the user.
  - Information hiding.
  - Objects contain their own data and algorithms.

- **Encapsulation**
  - Information hiding.
  - Objects contain their own data and algorithms.
  - Data Security
- **Inheritance**
  - Writing reusable code.
  - Objects can inherit characteristics from other objects.
- **Polymorphism**
  - A single name can have multiple meanings depending on its context .

Characteristics of JAVA

- Object Oriented
- Reusability
- Extensibility
- Portability
- Modularity
- Reliability
- Adaptability
- Modifiability
- Procedural
- Compatibility...

Pillars of an OOP

- **1.Inheritance**: Inheritance provide the facility to drive one class by another using simple syntax.
- **2.Encapsulation**: Encapsulation is the ability to bundle the property and method of the object and also operate them.
- **3.Polymorphism**: As the name suggest one name multiple form, Polymorphism is the way that provide the different functionality by the functions having the same name based on the signatures of the methods.
- **4.Dynamic binding**: It is the way that provide the maximum functionality to a program for a specific type at runtime.

1. **Encapsulation**: Encapsulation is the process of binding together the methods and data variables as a single entity. It keeps both the data and functionality code safe from the outside world. It hides the data within the class and makes it available only through the methods.

- Java provides different accessibility scopes (public, protected, private ,default) to hide the data from outside.

//Write a Java Program for concept of Encapsulation

```
class Check
{               private int amount=0;
                public int getAmount()
                {               return amount;
                }
                public void setAmount(int amt)
                {               amount=amt;
                }
}
public class Mainclass
{               public static void main(String[ ] args)
                {               int amt=0;
                                Check obj=new Check();
                                obj.setAmount(200);
                                amt=obj.getAmount();
                                System.out.println("Your current amount is :"+amt);
                }
}
```

2. **Inheritance**: Inheritance allows a class (subclass) to acquire the properties and behavior of another class (superclass). In java, a class can inherit only one class (superclass) at a time but a class can have any number of subclasses. It helps to reuse, customize and enhance the existing code. So it helps to write a code accurately and reduce the development time. Java uses extends keyword to extend a class.

//Write a Java Program for concept of Inheritance

```
class A
{       public void fun1(int  x)
        {               System.out.println("Int in A is :" + x);
        }
}
class B extends A
{       public void fun2(int  x, int  y)
        {               fun1(6);  // prints "int in A"
                        System.out.println("Int in B is :" + x + " and " + y);
        }
}
public class inherit
{       public static void main(String[] args)
        {               B obj=new B();
                        obj.fun2(2, 5);
        }
}
```

3. **Polymorphism**: Polymorphism allows one interface to be used for a set of actions i.e. one name may refer to different functionality.

- Polymorphism allows a object to accept different requests of a client (it then properly interprets the request like choosing appropriate method) and responds according to the current state of the runtime system, all without bothering the user.

- **There are two types of polymorphism :**
  - **Compile-time polymorphism**
  - **Runtime Polymorphism**

➢ In **compile time** Polymorphism, method to be invoked is determined at the compile time. Compile time polymorphism is supported through the **method overloading** concept in java.

➢ Method overloading means having multiple methods with same name but with different signature (number, type and order of parameters).

//Write a Java Program for concept of Compile-time polymorphism

```
class A
{	public void fun1(int x)
	{		System.out.println("The value of class A is : " + x);
	}
	public void fun1(int x,int y)
	{		System.out.println("The value of class B is : "+x +" and "+ y);
	}
}
public class polyone
{	public static void main(String[] args)
	{		A obj=new A();
			// Here compiler decides that fun1(int) is to be called and "int" will be printed.
			obj.fun1(2);
			// Here compiler decides that fun1(int,int) is to be called and "int and int" will be printed.
			obj.fun1(2,3);	}
}
```

- **Rumtime polymorphism**: The method to be invoked is determined at the run time. The example of run time polymorphism is **method overriding**. When a subclass contains a method with the same name and signature as in the super class then it is called as method overriding.

//Write a Java Program for concept of Runtime Polymorphism

```
class A
{	public void fun1(int x)
	{		System.out.println(""int in Class A is : " + x : " + x);
	}
}
class B extends A
{	public void fun1(int x)
	{		System.out.println("int in Class B is : " + x);
	}
}
public class polytwo
{	public static void main(String[ ] args)
	{		A obj;
			obj=new A(); // line 1
			obj.fun1(2);  // line 2 (prints ""int in Class A is : 2")
			obj=new B();  // line 3
			obj.fun1(5);  // line 4 (prints "int in Class A is : 5")
	}
}
```

Basic building blocks of OOP

- **Class**: A collection of fields (instance and class variables) and methods.
- Instance variable (field variable or member variable
  - An instance variable is a variable that is defined in a class, but outside of a method. There is one copy of the variable for every instance (object) created from that class.
  - A common problem is trying to reference an instance variable from a static method. A static method (Ex: main) can only reference static variables in its own class (or its own local variables).
- **Class variable** (static variable)
  - A class variable or static variable is defined in a class, but there is only one copy regardless of how many objects are created from that class. It's common to define static final variables (constants) that can be used by all methods, or by other classes. Color. blue is an example of a static final variable.

- **Constructor**: When an object is created, a constructor for that class is called. If no constructor is defined, a default constructor is called. It's common to have multiple constructors taking different numbers of parameters. Before a constructor is executed, the constructor for the parent class is called implicitly if there is no parent constructor called explicitly on the first line.
- **Inner class**: If a class is defined within another class, it is an inner class. There are two common reasons to do this: to attach an anonymous listener to a control at the point where the control is being built, and to have access to the fields and methods of the enclosing class.

- **Override** (applies to methods): If a subclass redefines a method defined in a superclass, the method in the superclass is overridden.
- **Overload** (applies to methods): A method in a class is overloaded if there is more than one method by the same name.
- **Abstract class**: A class which doesn't define all it's methods is called an abstract class. To be useful, there must be a subclass which defines the missing methods.
- **Interface** : An interface is a list of methods that must be implemented. A class that implements an interface must define all those methods.

Object Oriented Programming: Benefits

1. Benefits of data abstraction + programming methodology
2. Code reuse and maintenance, extension of applications
3. Component-oriented software development and integration
4. Language power: inheritance, polymorphism
5. Reflect concepts of problem domains
6. Easier to use.

- **Advantages of OOP**

1. OOP provides a clear modular structure for programs which makes it good for defining abstract data types where implementation details are hidden and the unit has a clearly defined interface.

1. OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.

1. OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interfaces.

4. OOP provides a clear modular structure for programs which makes it good for defining abstract datatypes where implementation details are hidden and the unit has a clearly defined interface.
5. OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
6. OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interfaces.

- **Disadvantages**
  - Initial development requires higher expertise.
  - More upfront development effort required.
- **Advantages**
  - Code re-use
  - Simplified application design
  - Lower cost of ownership.

Applications

- Applications (standalone, Web apps, components)
- Active desktop (Dynamic HTML, inclination Web)
- Create graphical applications
- Data access (e-mail, files, ODBC)
- Integrate components w/ other languages
- Simulation and Modeling
- User interface Design
- AI and Expert System
- Neural Networks.
- Web Designing.
- CAD/CAM System.