OBJECT-ORIENTED PROGRAMMING

. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts defined below:

1) Class is a user-defined data type defining its properties and functions. Class is the only logical representation of the data. For example, Human beings are a class.

The body parts of a human being are its properties, and the actions performed by the body parts are known as functions.

The class does not occupy any memory space until an object is instantiated.

In Java Syntax (For class):

```
 Public class student{
          public static void main(String[]args);

                     {

                        Int id; // Data member
                      System.out.println("Hello world !");

                       System.out.println("New World");

                                         }
                                         }
```

2) Object is a run-time entity. It is an instance of the class. An object can represent a person, place, or any other item. An object can operate on both data members and member functions.

Scanner sc=new Scanner (System. in); // Create object

Note: When an object is created using a new keyword, space is allocated for the variable in a heap, and the starting address is stored in the stack memory.

When an object is created without a new keyword, space is not allocated in the heap memory, and the object contains the null value in the stack.

Example:-Pen, chair, Table, computer, watch, etc.

3) Inheritance

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such a way, you can reuse, extend, or modify the attributes and behaviors defined in other classes.

In Java, the class that inherits the members of another class is called the derived class and the class whose members are inherited is called the base class.

The derived class is the specialized class for the base class.

<u>Syntax in java:-</u>

```
   class Subclass-name extends Superclass-name
   {
    //methods and fields
        }
```

<u>Example:-</u>

```
    class Animal
           {
            void sound()
           {
               System.out.println("Animal is making a sound");
                     }
                               }
Class Dog Extends Animal
{
         void sound()
           {
           System.out.println("Dog is barking");
           }
            }
public class Main {
public static void main(String[] args) {
   Dog d = new Dog();
   d.sound();
}
}
```

Output-Dog is barking

<u>Explanation</u>:-The Dog class inherits from the Animal class and overrides the sound method.

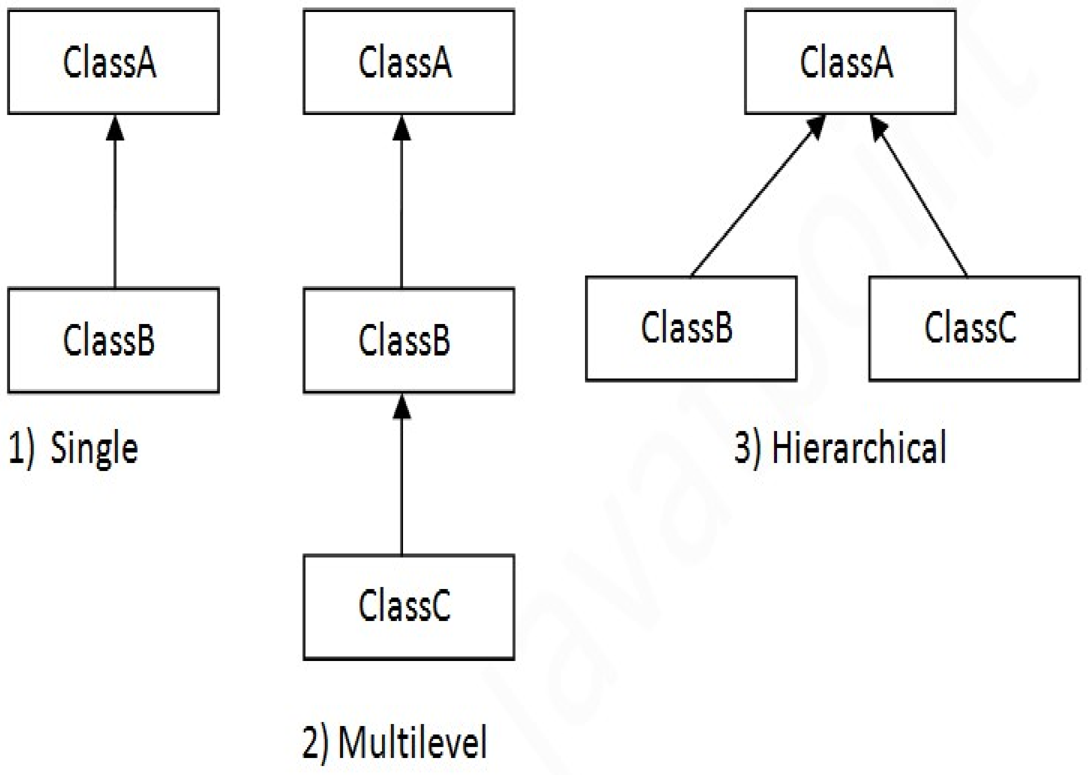
This is an example of both inheritance and polymorphism(method overriding)

Types of Inheritance :-

1.**Single inheritance:** When one class inherits another class, it is known as single-level inheritance

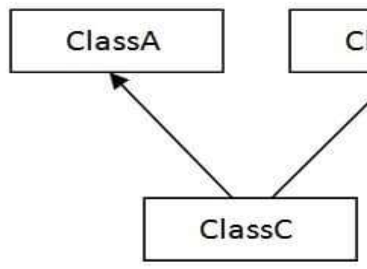
2.Multiple inheritance: Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.

3.Hierarchical inheritance: Hierarchical inheritance is defined as the process of deriving more than one class from a base class.

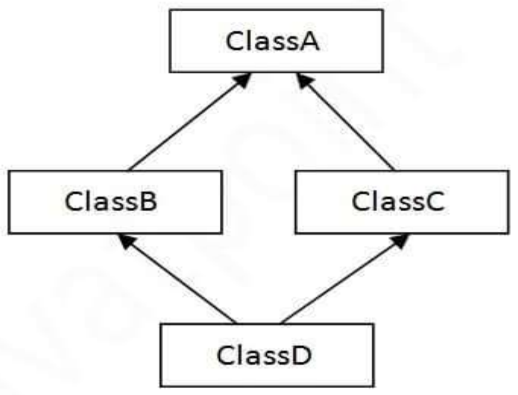


4.Multilevel inheritance: Multilevel inheritance is a process of deriving a class from another derived class.

5.Hybrid inheritance: Hybrid inheritance is a combination of simple, multiple inheritance and hierarchical inheritance.



4) Multiple



5) Hybrid

Multilevel Inheritance Example:-

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, Baby Dog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

Program:-

```java
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:-

```
weeping...

barking...

eating...
```

Hierarchical Inheritance Example:-

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Program:-

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were
```

```

 public static void main(String args[]){
  C obj=new C();
  obj.msg();//Now which msg() method would be invoked?
}
}
```

Output:-

```
Compile Time Error
```

4) Encapsulation:-

Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.(Data hiding: a language feature to restrict access to members of an object, reducing the negative effect due to dependencies.

Like:-Private,Protected and Public

Encapsulation

Company

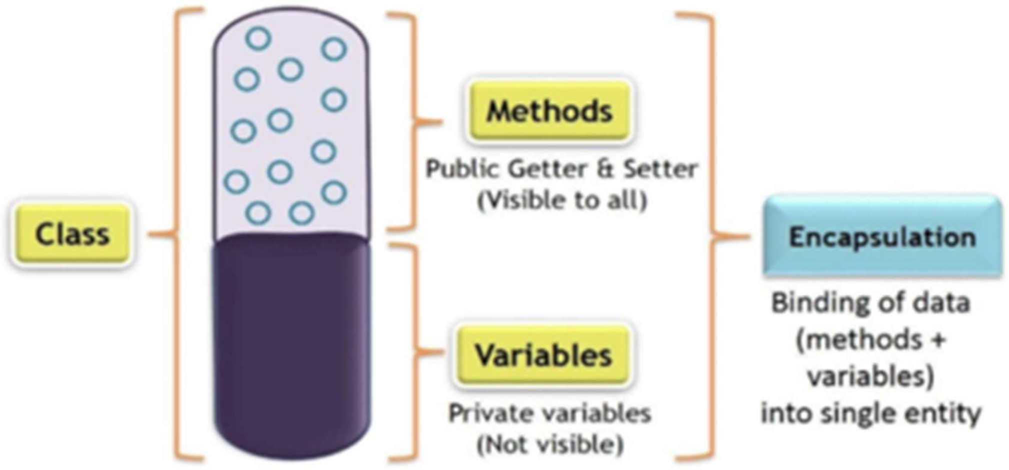
It can have several departments

→ Finance

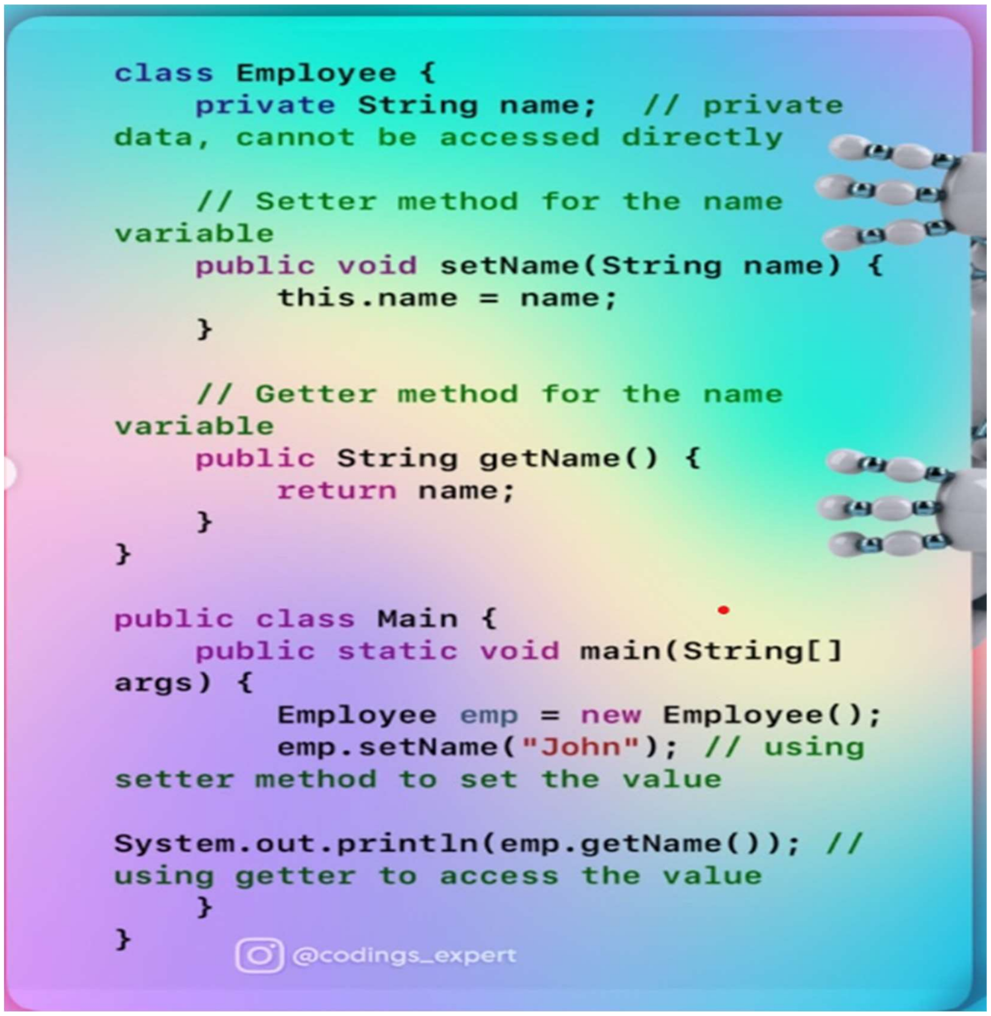
→ Marketing

→ Production

These departments form the company.


Class
Methods
Public Getter & Setter (Visible to all)
Variables
Private variables (Not visible)
Encapsulation
Binding of data (methods + variables) into single entity

Program:-



```
class Employee {
    private String name;  // private
data, cannot be accessed directly

    // Setter method for the name
variable
    public void setName(String name) {
        this.name = name;
    }

    // Getter method for the name
variable
    public String getName() {
        return name;
    }
}

public class Main {
    public static void main(String[]
args) {
        Employee emp = new Employee();
        emp.setName("John"); // using
setter method to set the value

System.out.println(emp.getName()); //
using getter to access the value
    }
}
```

@codings_expert

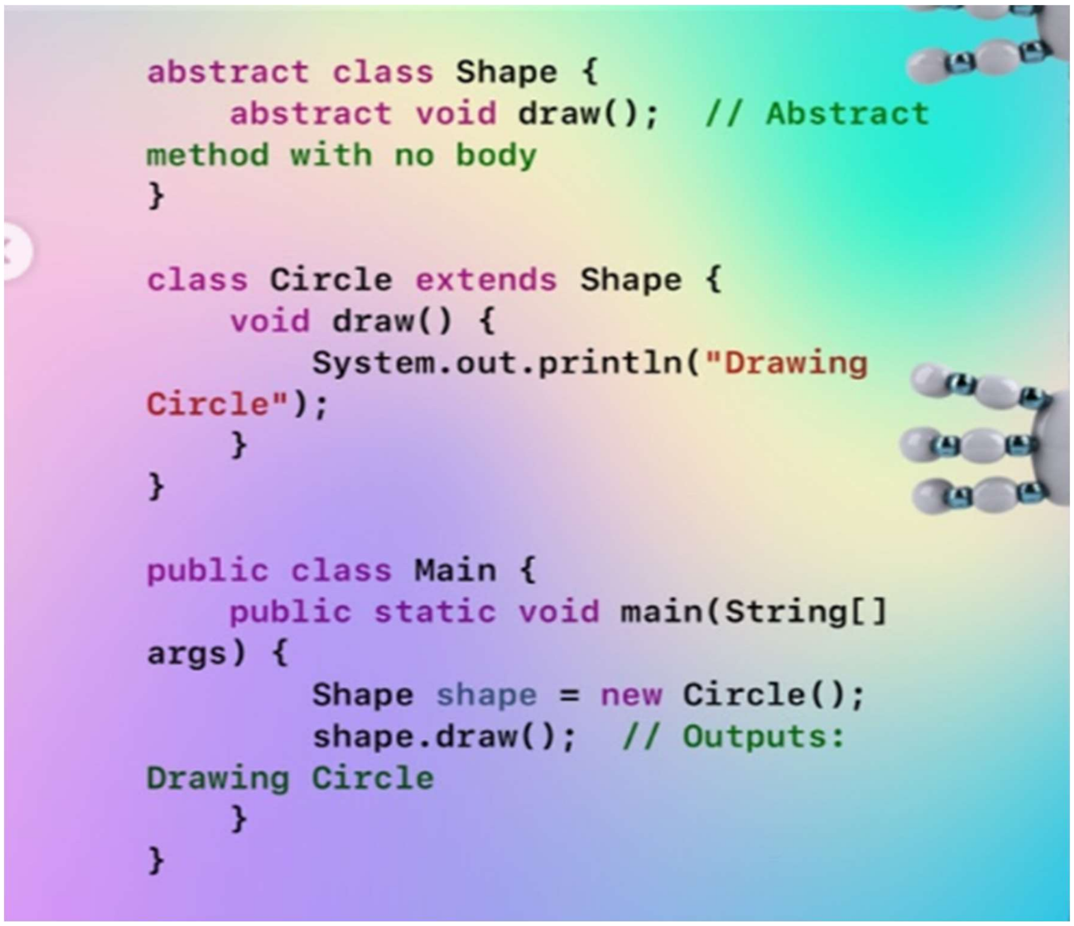
5) Abstraction:-

We try to obtain an abstract view, model or structure of a real life problem, and reduce its unnecessary details. With definition of properties of problems, including the data which are affected and the operations which are identified, the model abstracted from problems can be a standard solution to this type of problems. It is an efficient way since there are nebulous real-life problems that have similar properties.

Example:-(Abstract class)

Data binding : Data binding is a process of binding the application UI and business logic. Any change made in the business logic will reflect directly to the application UI.

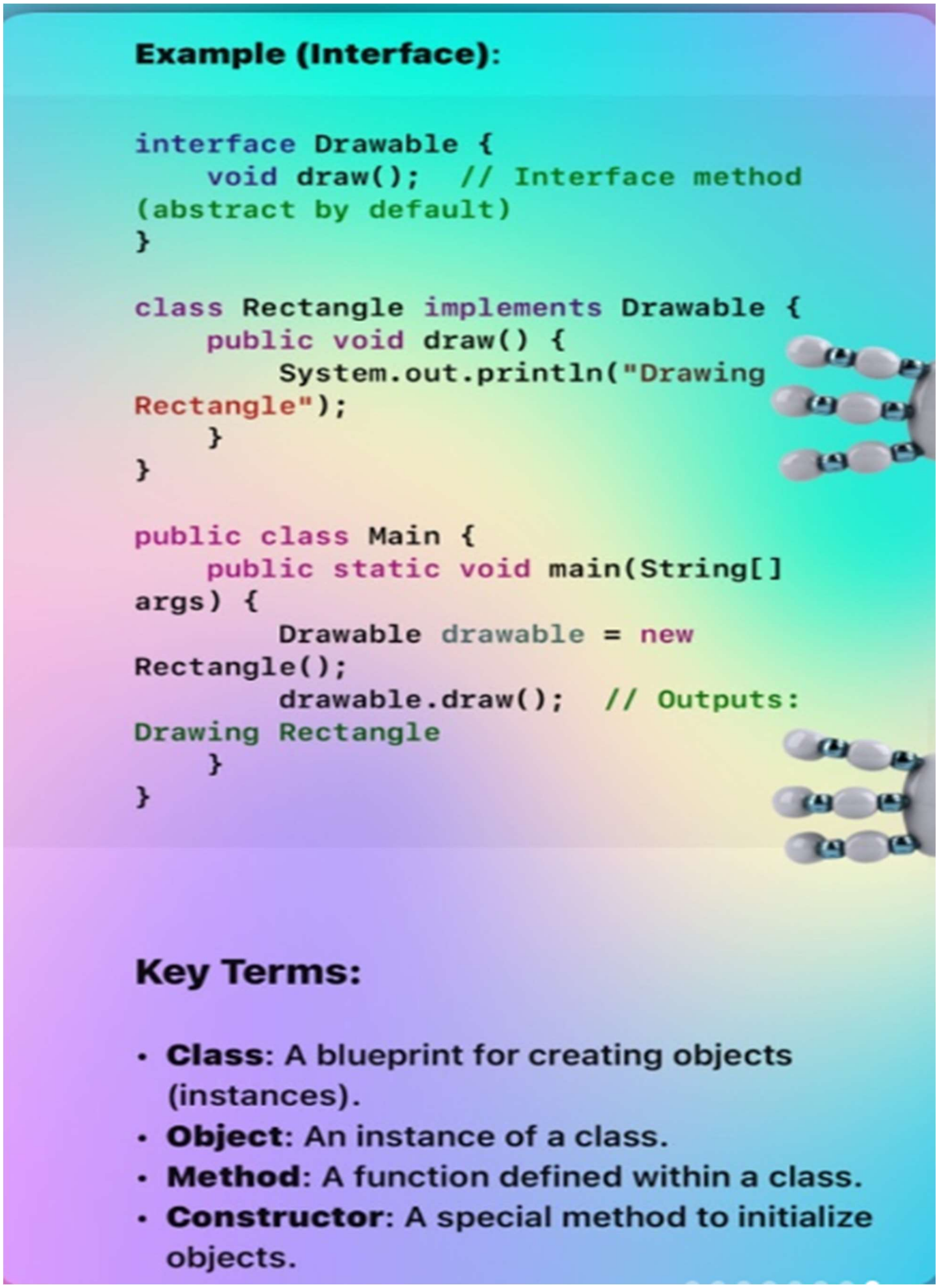
Program:-



```
abstract class Shape {
    abstract void draw();  // Abstract
method with no body
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing
Circle");
    }
}

public class Main {
    public static void main(String[]
args) {
        Shape shape = new Circle();
        shape.draw();  // Outputs:
Drawing Circle
    }
}
```



Example (Interface):

```
interface Drawable {
    void draw();  // Interface method
(abstract by default)
}

class Rectangle implements Drawable {
    public void draw() {
        System.out.println("Drawing
Rectangle");
    }
}

public class Main {
    public static void main(String[]
args) {
        Drawable drawable = new
Rectangle();
        drawable.draw();  // Outputs:
Drawing Rectangle
    }
}
```

Key Terms:

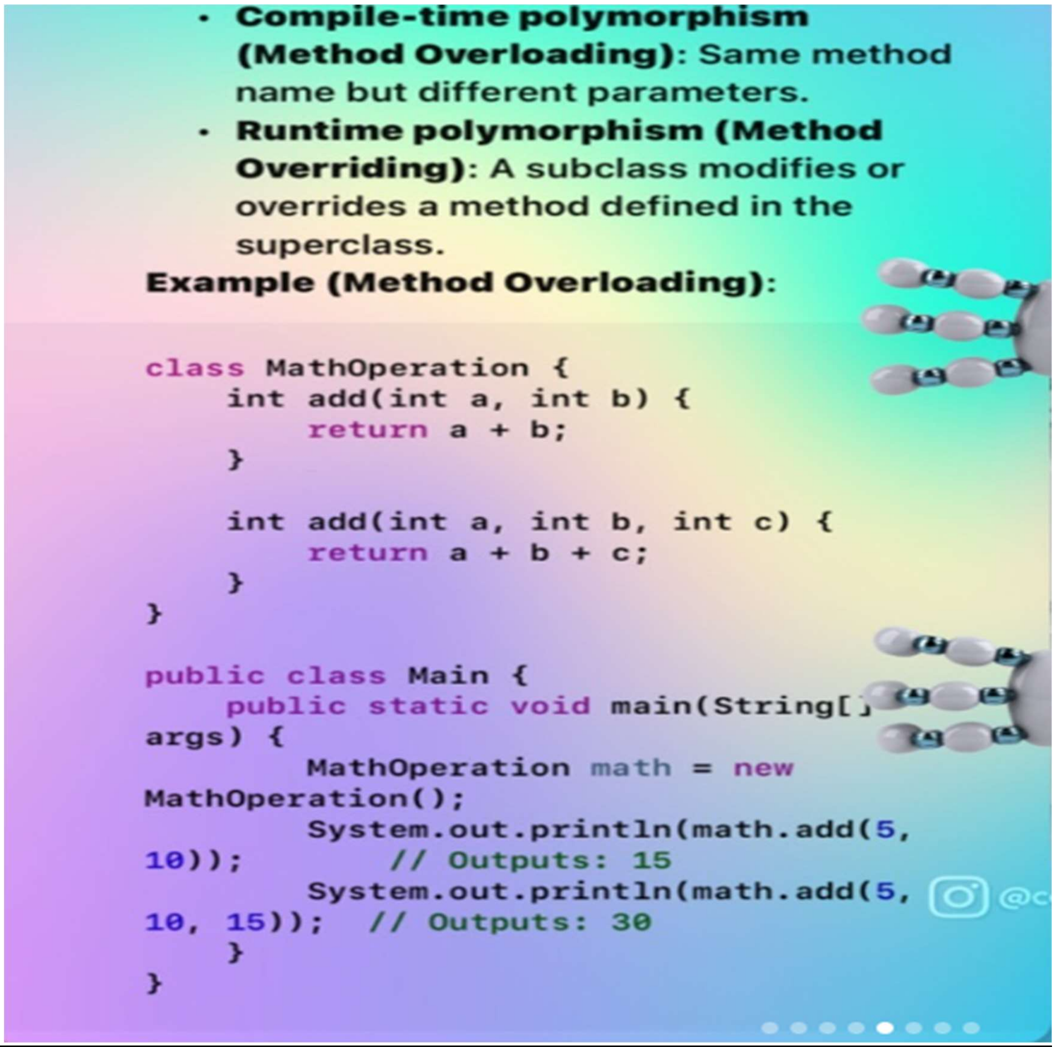
- **Class**: A blueprint for creating objects (instances).
- **Object**: An instance of a class.
- **Method**: A function defined within a class.
- **Constructor**: A special method to initialize objects.

6) Polymorphism:-

Polymorphism is the ability to present the same interface for differing underlying forms (data types). With polymorphism, each of these classes will have different underlying data. Apoint shape needs only two coordinates (assuming it's in a two-dimensional space of course). Acircle needs a center and radius. Asquare or rectangle needs two coordinates for the top left and bottom right corners and (possibly) a rotation. An irregular polygon needs a series of lines. Precisely, Poly means 'many' and morphism means 'forms'.

Types of Polymorphism:-

1) **Compile Time Polymorphism(Static) Ex-Method Overloading.**
2) **Runtime Polymorphism(Dynamic) Ex-Method Overriding.**



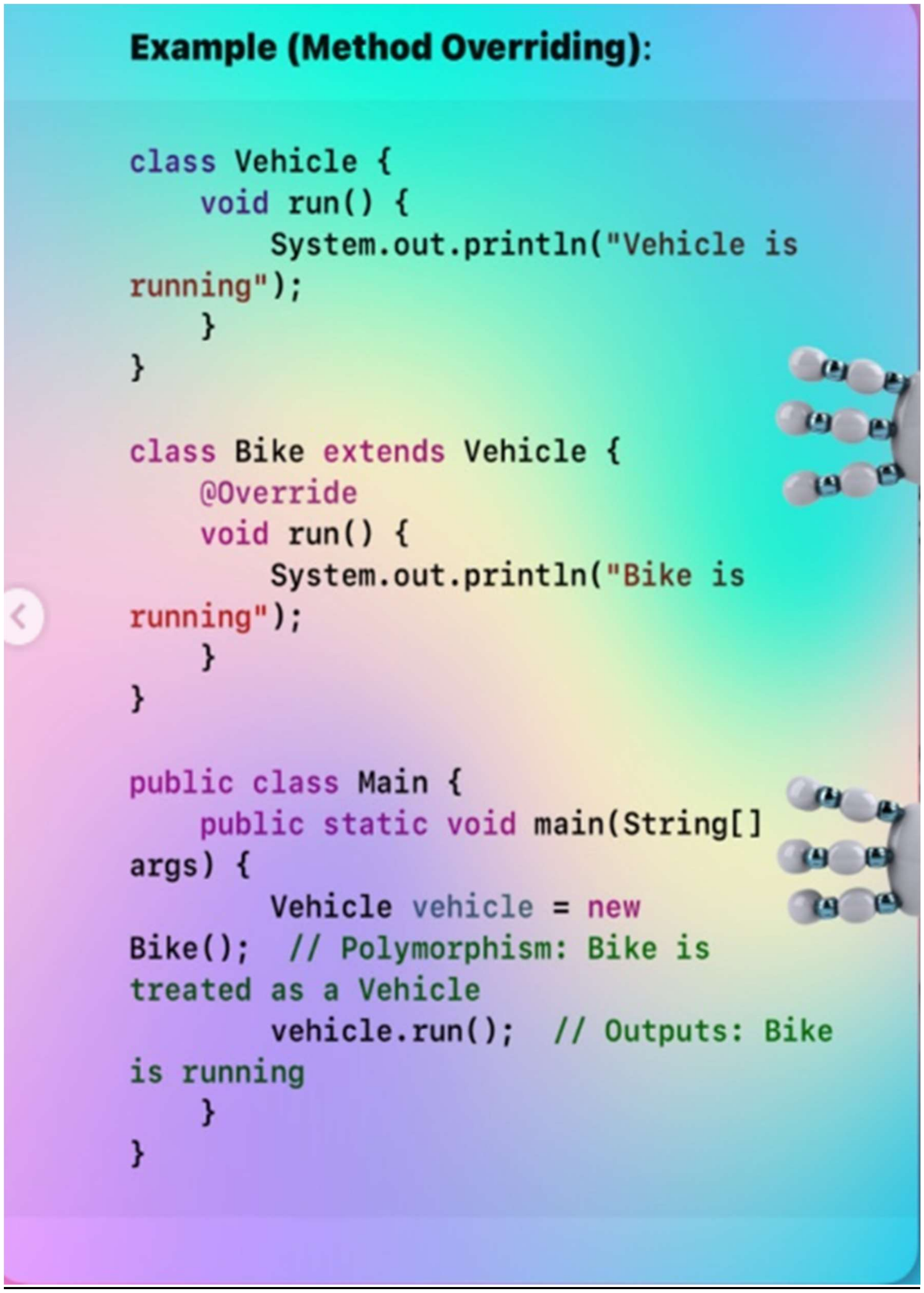
- **Compile-time polymorphism (Method Overloading)**: Same method name but different parameters.
- **Runtime polymorphism (Method Overriding)**: A subclass modifies or overrides a method defined in the superclass.

Example (Method Overloading):

```
class MathOperation {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}

public class Main {
    public static void main(String[]
args) {
        MathOperation math = new
MathOperation();
        System.out.println(math.add(5,
10));       // Outputs: 15
        System.out.println(math.add(5,
10, 15));  // Outputs: 30
    }
}
```



Example (Method Overriding):

```
class Vehicle {
    void run() {
        System.out.println("Vehicle is
running");
    }
}

class Bike extends Vehicle {
    @Override
    void run() {
        System.out.println("Bike is
running");
    }
}

public class Main {
    public static void main(String[]
args) {
        Vehicle vehicle = new
Bike();  // Polymorphism: Bike is
treated as a Vehicle
        vehicle.run();  // Outputs: Bike
is running
    }
}
```

Constructor: Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as class or structure.

There are be two types of constrctor in java.

1).Default constructor: A constructor which has no argument is known as default constructor. It is invoked at the time of creating an object.

2.Parameterized constructor: Constructor which has parameters is called a parameterized constructor. It is used to provide different values to distinct objects.

3.Copy Constructor: A Copy constructor is an overloaded constructor used to declare and initialize an object from another object. It is of two types - default copy constructor and user defined copy constructor.

● Destructor: A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like a constructor. It must have the same name as class, prefixed with a tilde sign (~).

● Friend Function: Friend function acts as a friend of the class. It can access the private and protected members of the class. The friend function is not a member of the class, but it must be listed in the class definition. The non-member function cannot access the private data of the class. Sometimes, it is necessary for the non-member function to access the data. **The friend function is a non-member function and has the ability to access the private data of the class.**

Note: 1. A friend function cannot access the private members directly, it has to use an object name and dot operator with each member name.

2. Friend function uses objects as arguments.

Declaration of friend function in C++

```
class class_name
{
   friend data_type function_name(argument/s);           // syntax of friend function.
};
```

Program:-

```
#include <iostream>
using namespace std;
class Box
{
    private:
        int length;
    public:
        Box(): length(0) { }
        friend int printLength(Box); //friend function
};
```

```
int printLength(Box b)
{
   b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

● **Aggregation:** It is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents the HAS-A relationship.
● **Virtual Function** IMP: A virtual function is used to replace the implementation provided by the base class. The replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer.

1. A virtual function is a member function which is present in the base class and redefined by the derived class.

2. When we use the same function name in both base and derived class, the function in base class is declared with a keyword virtual.

3. When the function is made virtual, then determines at run-time which function is to be called based on the type of the object pointed by the base class pointer.

Thus, by making the base class pointer to point to different objects, we can execute different versions of the virtual functions.

Key Points:-

1. Virtual functions cannot be static.

2. A class may have a virtual destructor but it cannot have a virtual constructor.

Parent.Java:

```
  class Parent {
void v1() //Declaring function
{
System.out.println("Inside the Parent Class");
}
}
```

Child.java:

```
public class Child extends Parent{
            void v1() // Overriding function from the Parent class
            {
            System.out.println("Inside the Child Class");
            }
```

```
        public static void main(String args[]){
        Parent ob1 = new Child(); //Refering the child class object using the parent cl
ass
        ob1.v1();
        }
        }
```

Output:

```
Inside the Child Class
```

● **Pure Virtual Function:**

1. Apure virtual function is not used for performing any task. It only serves as a placeholder.

2. Apure virtual function is a function declared in the base class that has no definition relative to the base class.

3. A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.

4. The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.

● Access Specifiers IMP: The access specifiers are used to define how functions and variables can be accessed outside the class. There are three types of access specifiers:

1. Private: Functions and variables declared as private can be accessed only within the same class, and they cannot be accessed outside the class they are declared.

2. Public: Functions and variables declared under public can be accessed from anywhere.

3. Protected: Functions and variables declared as protected cannot be accessed outside the class except a child class. This specifier is generally used in inheritance.

Key Notes

● **Delete** : is used to release a unit of memory, delete[] is used to release an array.

● **Virtual inheritance**: facilitates you to create only one copy of each object even if the object appears more than one in the hierarchy.

● **Function overloading:** Function overloading is defined as we can have more than one version of the same function. The versions of a function will have different signatures meaning that they have a different set of parameters.

Operator overloading: Operator overloading is defined as the standard operator can be redefined so that it has a different meaning when applied to the instances of a class.

● **Overloading** is static Binding, whereas Overriding is dynamic Binding. Overloading is nothing but the same method with different arguments, and it may or may not return the same value in the same class itself.

Overriding: is the same method name with the same arguments and return types associated with the class and its child class.