Object-Oriented Programming (OOP)

Abstraction, Encapsulation, Inheritance & Polymorphism

OOP

- Classes and objects are the two main aspects of object-oriented programming.
- **Object**: Object means a real-world entity.
- Any entity that has state and behavior is known as an object.
- For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- An object can be defined as an instance of a class.
- An object contains an address and takes up some space in memory.
- **class**: Collection of objects is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object.
- class doesn't consume any space.

OOP

- All object-oriented programming languages provide mechanisms that help you implement the object-oriented model.
  - ✓Abstraction
  - ✓Encapsulation
  - ✓Inheritance and
  - ✓Polymorphism.

Abstraction

- Abstraction is an essential element of object-oriented programming.
- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- It is the act of representing the essential features without knowing the background details. It is always related to the purpose or user.
- Hide the unwanted details from user.
- A powerful way to manage abstraction is through the use of hierarchical classifications.
- Hierarchical abstractions of complex systems can also be applied to computer programs.

Encapsulation

- Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
- It is the system of wrapping data and functions into a single unit (called class).
- 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.
- Code in Java must be encapsulated in classes.

Inheritance

- Inheritance is the process by which one object acquires the properties of another object.
- It means to link and share some common properties of one class with other class.
- This is important because it supports the concept of hierarchical classification.
- A new subclass inherits all of the attributes of all of its ancestors.

Polymorphism

- Polymorphism is derived from two Greek words: **poly** and **morphs**. The word **poly** means *many* and **morphs** means *forms*. So polymorphism means *many forms*.
- It is the process of using a function/method for more than one purpose.
- There are two types of polymorphism in Java:
- Compile-time polymorphism (overloading)
- Runtime polymorphism (overriding)

class Declaration

- Any concept you wish to implement in a Java program must be encapsulated within a class.
- A class is a template for an object, and an object is an instance of a class.
- A class is declared by use of the **class** keyword.

<class modifiers> class <class name> <extends clause> <implements clause>
{
    <field declarations>
    <method declarations>
<nested class/interface declarations>
<constructor declarations>
}

class Declaration

- The data, or variables, defined within a class are called instance variables.
- Collectively, the methods and variables defined within a class are called members of the class.
- Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables.
- Each object has its own copies of the instance variables.
- Changes to the instance variables of one object have no effect on the instance variables of another.

Declaring Objects

- When we create a class, you are creating a new data type. We can use this type to declare objects of that type.
- However, obtaining objects of a class is a two-step process.
- First, you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object.

Example: Abc obj; // declare reference to object

- Second, you must acquire an actual, physical copy of the object and assign it to that variable. We can do this using the **new** operator.
- The **new** operator dynamically allocates memory for an object and returns a reference to it.
- This reference is then stored in the variable.

Example: obj=new Abc(); // allocate a Abc object

- Thus, in Java, all class objects must be dynamically allocated

Introducing Methods

- Classes usually consist of two things: instance variables and methods.

```
type name(parameter-list) {
// body of method
}
```

- Here, type specifies the type of data returned by the method.
- If the method does not return a value, its return type must be **void**.
- The name of the method can be any legal identifier other than those already used by other items within the current scope.
- The parameter-list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the arguments passed to the method when it is called.
- If the method has no parameters, then the parameter list will be empty.
- Methods that have a return type other than void return a value to the calling routine using the following form of the return statement:

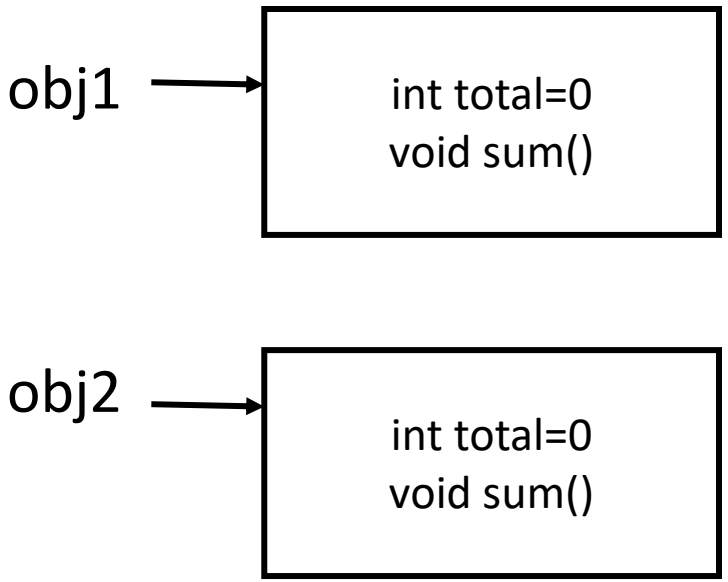
```
return value;
```

- Here, value is the value returned.

Example

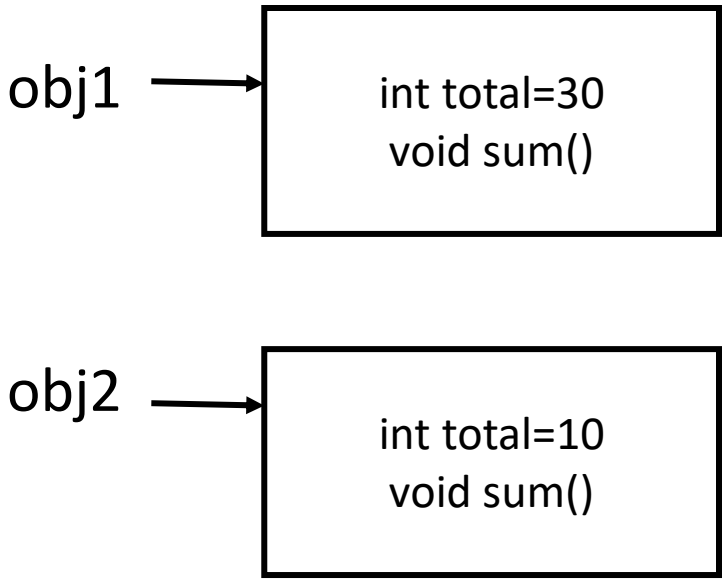
- Example:

```
class Abc
{
      int total;
      void sum(int a, int b){
                  total=a+b;
       }
public static void main(String[] args) {
      Abc obj1, obj2;
      obj1=new Abc();
      obj2=new Abc();
      obj1.sum(10,20);
      obj2.sum(4,6);
      System.out.println("value of total for obj1= "+obj1.total);
      System.out.println("value of total for obj2= "+obj2.total);
}
```



Example

- Example:




```
class Abc
{
      int total;
      void sum(int a, int b){
                  total=a+b;
       }
public static void main(String[] args) {
      Abc obj1=new Abc();
      Abc obj2=new Abc();
      obj1.sum(10,20);
      obj2.sum(4,6);
      System.out.println("value of total for obj1= "+obj1.total);
      System.out.println("value of total for obj2= "+obj2.total);
}
```

Overloading Methods

- Method overloading is one of the ways that Java supports polymorphism.
- Defining two or more methods within the same class that share the same name, as long as their parameter declarations are different.
- Overloaded methods must differ in the type and/or number of their parameters.
- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- Overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method.
- Return types do not play a role in overloading

Overloading Methods

- Example:

```
class Abc{
        void test() {
                System.out.println("No parameters");
        }
        void test(int a) {
                System.out.println("a: " + a);
        }
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
        public static void main(String[] args) {
                Abc ob=new Abc();
                double result;
                ob.test();
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(11.10);
                System.out.println("Result of ob.test(11.10): " + result);
        }
}
```

Overloading Methods

- Example:

```
class Abc{
        void test() {
                System.out.println("No parameters");
        }
        void test(int a) {
                System.out.println("a: " + a);
        }
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
        public static void main(String[] args) {
                Abc ob=new Abc();
                double result;
                ob.test();
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(11.10);
                System.out.println("Result of ob.test(11.10): " + result);
        }
}
```

Output:

No parameters
a: 10
a and b: 10 20
double a: 11.1
Result of ob.test(11.10): 123.21

Overloading Methods

- When an overloaded method is called, Java looks for a match between the arguments used to call the method and the method's parameters.
- However, this match need not always be exact.

Overloading Methods

- Example:

```
class Abc{
        void test() {
                System.out.println("No parameters");
        }
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
        public static void main(String[] args) {
                Abc ob=new Abc();
                double result;
                ob.test();
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(11.10);
                System.out.println("Result of ob.test(11.10): " + result);
        }
}
```

Overloading Methods

- Example:

```
class Abc{
        void test() {
                System.out.println("No parameters");
        }
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }
        double test(double a) {
                System.out.println("double a: " + a);
                return a*a;
        }
        public static void main(String[] args) {
                Abc ob=new Abc();
                double result;
                ob.test();
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(11.10);
                System.out.println("Result of ob.test(11.10): " + result);
        }
}
```

Output:

```
No parameters
double a: 10.0
a and b: 10 20
double a: 11.1
Result of ob.test(11.10): 123.21
```

Overloading Methods

- As you can see, this version of example does not define test(int).
- When test( ) is called with an integer argument inside Overload, no matching method is found.
- Java can automatically convert an integer into a double, and this conversion can be used to resolve the call.
- Therefore, after test(int) is not found, Java elevates i to double and then calls test(double).
- Of course, if test(int) had been defined, it would have been called instead. Java will employ its **automatic type conversions** only if **no exact match** is found.

Objects as Parameters

- So far, we have only been using simple types as parameters to methods.
- However, it is both correct and common to pass objects to methods.

```
class Abc
{
	int a,b;
	void test(Abc o)
	{
		System.out.println("a="+ o.a +", b="+ o.b);
	}
public static void main(String[] args) {
	Abc ob=new Abc();
		ob.a=10;
		ob.b=20;
	Abc ob1=new Abc();
		ob1.a=50;
		ob1.b=100;
ob.test(ob);
ob1.test(ob1);
}
}
```

Objects as Parameters

- So far, we have only been using simple types as parameters to methods.
- However, it is both correct and common to pass objects to methods.

```
class Abc
{
	int a,b;
	void test(Abc o)
	{
		System.out.println("a="+ o.a +", b="+ o.b);
	}
public static void main(String[] args) {
	Abc ob=new Abc();
		ob.a=10;
		ob.b=20;
	Abc ob1=new Abc();
		ob1.a=50;
		ob1.b=100;
ob.test(ob);
ob1.test(ob1);
}
}
```

Output:

a=10, b=20

a=50, b=100

Objects as return type

- A method can return any type of data, including class types that you create.

```
class Abc
{
        int a,b;
        Abc create_obj()
        {
                Abc o=new Abc();
                return o;
        }

  public static void main(String[] args) {
          Abc obj=new Abc();
                obj.a=5;
                obj.b=10;
                Abc obj_new;
                obj_new=obj.create_obj();
                System.out.println("a="+ obj.a +", b="+ obj.b);
                System.out.println("a="+ obj_new.a +", b="+ obj_new.b);
}
}
```

Objects as return type

- A method can return any type of data, including class types that you create.

```
class Abc
{
        int a,b;
        Abc create_obj()
        {
                Abc o=new Abc();
                return o;
        }

  public static void main(String[] args) {
          Abc obj=new Abc();
                obj.a=5;
                obj.b=10;
                Abc obj_new;
                obj_new=obj.create_obj();
                System.out.println("a="+ obj.a +", b="+ obj.b);
                System.out.println("a="+ obj_new.a +", b="+ obj_new.b);
}
}
```

Output:

a=5, b=10

a=0, b=0