ACE Inspiration

Java Web Development Course
Chapter 1 – Core Java

- Java Basics Part-4(OOP)

Contents

ACE Inspiration

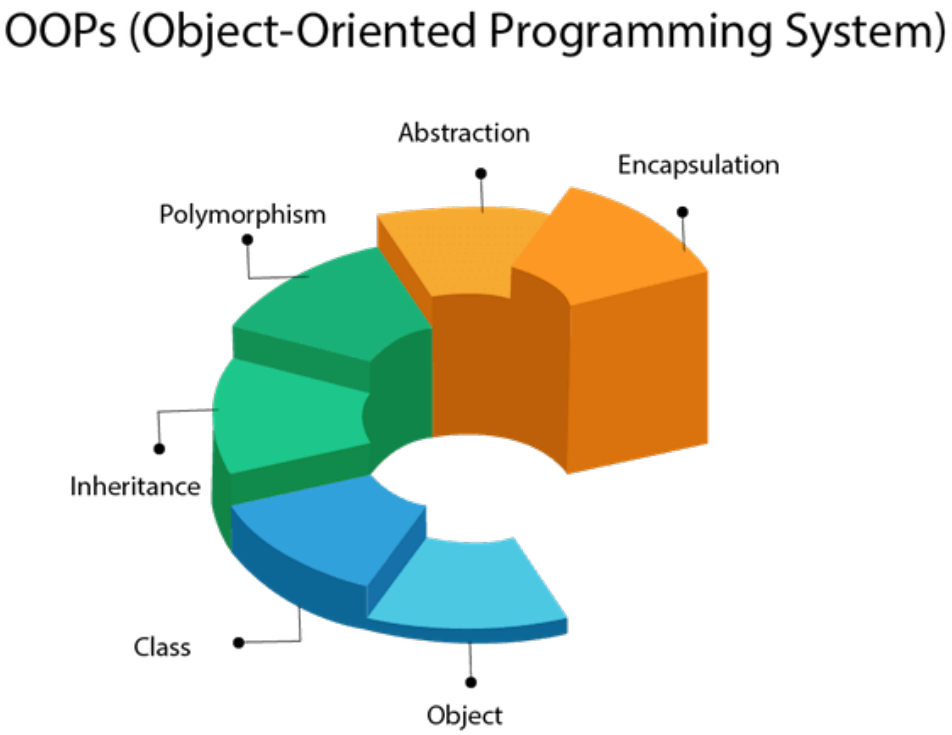
- Object Oriented Programming
- Object & Class
- Inheritance
- Aggregation
- Polymorphism
- Super Keyword
- Abstraction
- Encapsulation
- Workout

Object Oriented Programming

ACE Inspiration

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Class
- Object
- Inheritance
- Polymorphisms
- Abstraction
- Encapsulation



Object & Class

ACE Inspiration

Object

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. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity.

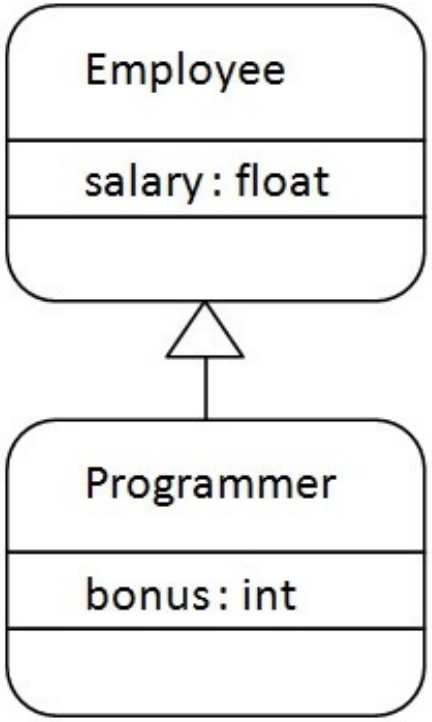
A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

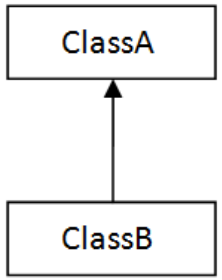
ACE Inspiration

Inheritance uses for Code Reusability

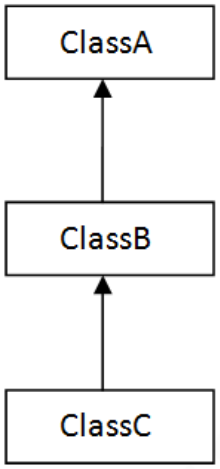
- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.
- The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.



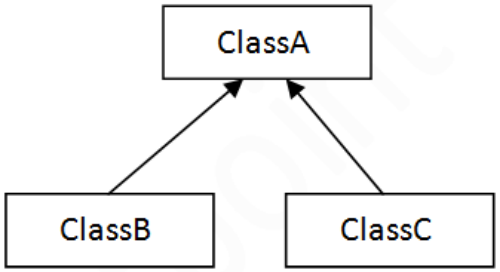
Inheritance



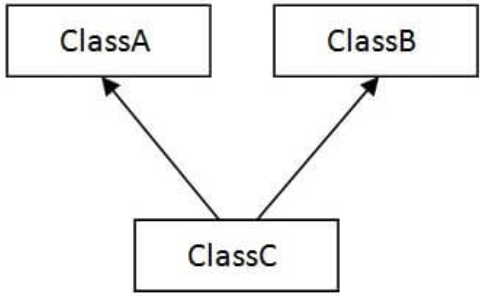
1) Single



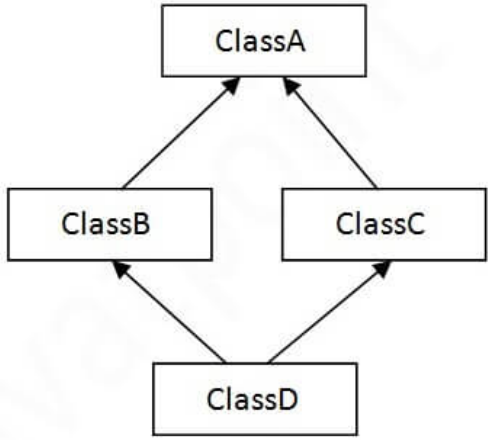
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Single Inheritance

ACE Inspiration

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

Multilevel Inheritance

ACE Inspiration

```
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...");}
}
 public class SingleInheritance{
        public static void main(String args[]){
                 BabyDog  d=new  BabyDog();
                d.bark();
                d.eat();
        }
}
```

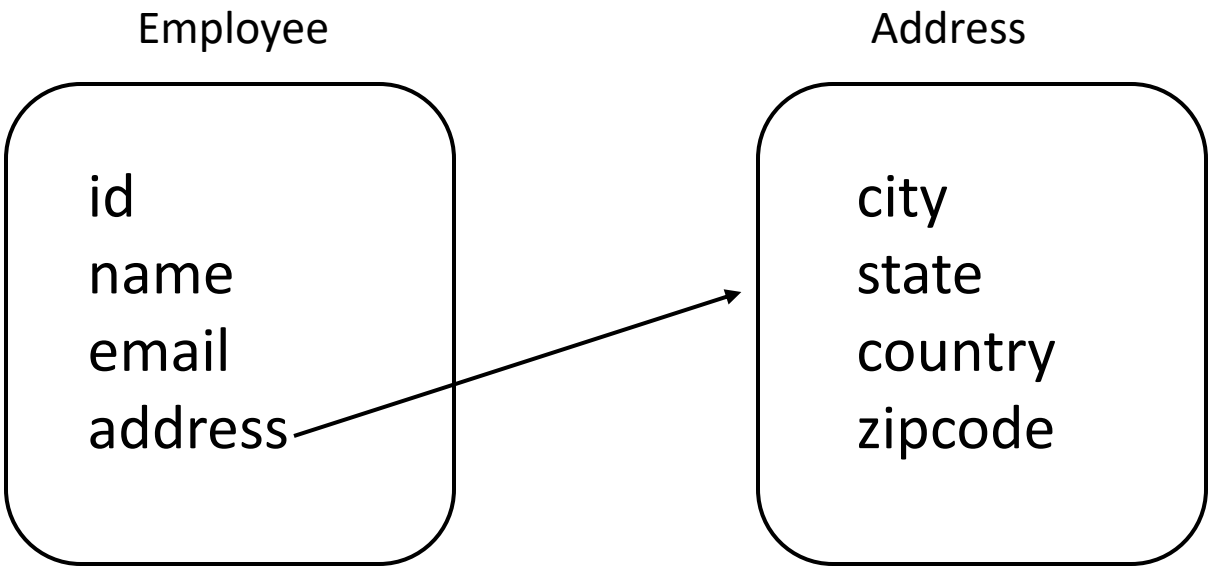
Hierarchical Inheritance

ACE Inspiration

```
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...");}
}
 public class HierarchicalInheritance{
        public static void main(String args[]){
                Dog d=new Dog();
                d.bark();               d.eat();
                Cat c=new Cat();
                c.meow();               c.eat();
        }
}
```

Aggregation(HAS-A)

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship. Consider a situation, Employee object contains many information such as id, name, emailId etc. It contains one more object named address, which contains its own information such as city, state, country, zip code etc. as given below.



Aggregation(HAS-A)

ACE Inspiration

```
public class Address {
        String city,state,country;

        public Address(String city, String state, String country) {
                this.city = city;
                this.state = state;
               this.country = country;
        }

}
```

Aggregation(HAS-A)

ACE Inspiration

```
public class Emp {
        int id;
        String name;
        Address address;
        public Emp(int id, String name,Address address) {
                this.id = id;
                this.name = name;
        this.address=address;
        }
        void display(){
                System.out.println(id+" "+name);
        System.out.println(address.city+" "+address.state+" "+address.country);
        }
        public static void main(String[] args) {
                Address address1=new Address("gzb","UP","india");
                Address address2=new Address("gno","UP","india");
                Emp e=new Emp(111,"varun",address1);
                Emp e2=new Emp(112,"arun",address2);
                e.display();
                e2.display();
        }
}
```

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use **method overloading and method overriding** to achieve polymorphism. Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Method Overloading

ACE Inspiration

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**. It is also call static polymorphism and compile time polymorphism because it resolve compile time.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

By changing number of arguments

By changing the data type

Method Overloading

ACE Inspiration

```
class Adder{
        static int add(int a,int b){return a+b;}
        static int add(int a,int b,int c){return a+b+c;}
        static double add(double a,double b){return a+b;}
}
 public class TestOverloading{
        public static void main(String[] args){
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(11,11,11));
        System.out.println(Adder.add(1.1,1.1));
        }
}
```

Method Overloading

ACE Inspiration

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
class MainMethodOverloading{
        public static void main(String[] args){
                System.out.println("main with String[]");
        }
        public static void main(String args){
                System.out.println("main with String");
        }
        public static void main(){
                System.out.println("main without args");
        }
}
```

Method Overriding

ACE Inspiration

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

The method must have the same name as in the parent class

The method must have the same parameter as in the parent class.

There must be an IS-A relationship (inheritance).

Method Overriding

ACE Inspiration

```
class Bank{
        int getRateOfInterest(){return 0;}
}
class KBZ extends Bank{
        int getRateOfInterest(){return 8;}
}
class AYA extends Bank{
        int getRateOfInterest(){return 7;}
}
class TestOverriding{
        public static void main(String args[]){
        KBZ s=new KBZ();
        AYA i=new AYA();

        System.out.println("KBZ Rate of Interest: "+s.getRateOfInterest());
        System.out.println("AYA Rate of Interest: "+i.getRateOfInterest());
        }
}
```

Super Keyword

ACE Inspiration

- Super can be used to refer immediate parent class instance variable. We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.
- Super can be used to invoke immediate parent class method. The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.
- Super() can be used to invoke immediate parent class constructor. The super keyword can also be used to invoke the parent class constructor.

Super Keyword

```
class Animal {
String color="white";
Animal(){
System.out.println("Animal Constructor...");
}
void eat() {
System.out.println("eating...");
}
}
class Dog extends Animal {
String color="black";
Dog(){
super();//calls animal constructor
System.out.println("Dog Constructor...");
}
void eat() {
super.eat();//calls eat method from Animal class
System.out.println("eating bread...");
}
void printColor(){
System.out.println(super.color);//prints color of Animal class
System.out.println(color);//prints color of Dog class
}
}
public class TestSuper {
public static void main(String args[]) {
Dog d = new Dog();
d.eat();
d.printColor();
}
}
```

Oupput :

```
Animal Constructor...
Dog Constructor...
eating...
eating bread...
white
black
```

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing. In Java, we use abstract class and interface to achieve abstraction. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

Abstract class (0 to 100%)

Interface (100%)

Abstract class

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have constructor and static methods also.
5. It can have final methods which will force the subclass not to change the body of the method.

Abstract class

```
abstract class Bike {
        Bike() {
                System.out.println("bike is created");
        }
        abstract void run();
        void changeGear() {
                System.out.println("gear changed");
        }
}
class Honda extends Bike {
        void run() {
                System.out.println("running safely..");
        }
}
public class TestAbstractClass {
        public static void main(String args[]) {
                Bike obj = new Honda();
                obj.run();
                obj.changeGear();
        }
}
```

Interface

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There **can be only abstract methods** in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**. It **cannot be instantiated just like the abstract class**. Since Java 8, we can have **default and static methods** in an interface. Since Java 9, we can have **private methods** in an interface.

Why use Java interface?

It is used to achieve abstraction.

By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling.

Interface

```
interface Printable{
        void print();
}
interface Showable{
        void show();
}
 public class TestInterface implements Printable,Showable{
        public void print(){System.out.println("Hello");}
        public void show(){System.out.println("Welcome");}

        public static void main(String args[]){
                TestInterface  obj = new  TestInterface();
                obj.print();
        obj.show();
         }
}
```

Abstract class VS Interface

ACE Inspiration

Abstract	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Encapsulation

ACE Inspiration

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines. We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it**. The Java Bean class is the example of a fully encapsulated class.**


Capsule

Advantage of Encapsulation in Java

- By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.
- It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
- It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
- The encapsulate class is **easy to test**. So, it is better for unit testing.
- The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

Encapsulation

ACE Inspiration

```
class Student{
        private int id;
        private String name;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
}
```

```
public class TestEncapsulation {
public static void main(String []args) {
        Student s=new Student();
        s.setId(1);
        s.setName("John");
        System.out.println("ID
"+s.getId());
        System.out.println("Name
"+s.getName());
}
}
```

Workout

ACE Inspiration

- Create a Abstract class Vehicles,
  Write abstract methods of getType, getColor, getBrandName
- Create a class Cars, Bicycles, Motorbikes
  Extend the Vehicles abstract class.
  Implement methods of getType, getColor, getBrandName by printing the different values.
- Create a class MyVehicles and write a main methods.
  public static void main(String args[]){

```
Vehicles obj = new Cars();
obj.getType();
obj.getColor();
obj.getBrandName();
obj = new Bicycles();
obj.getType();
obj.getColor();
obj.getBrandName();
obj = new Motorbytes();
obj.getType();
obj.getColor();
obj.getBrandName();
}
```

Workout

ACE Inspiration

- **Create a super class called Car. The Car class has the following fields and methods.**
  - **◦int speed;**
  - **◦double regularPrice;**
  - **◦String color;**
  - **◦Car(speed,regularPrice,color);**
  - **◦double getSalePrice();**
- **Create a sub class of Car class and name it as Truck. The Truck class has the following fields and methods.**
  - **◦int weight;**
  - **◦Truck(speed,regularPrice,color,weight); //call super class constructor**
  - **◦double getSalePrice();//Ifweight>2000,10%discount.Otherwise,20%discount.**
- **Create a subclass of Car class and name it as Ford. The Ford class has the following fields and methods**
  - **◦int year;**
  - **◦int manufacturerDiscount;**
  - **◦Ford(speed,regularPrice,color,year,manufaacturerDiscount); //call super class constructor**
  - **◦double getSalePrice(); //From the sale price computed from Carcclass, subtract the manufacturer Discount.**

Workout

ACE Inspiration

- **Create a subclass of Car class and name it as Sedan. The Sedan class has the following fields and methods.**
  - **int length;**
  - **Sedan(speed,regularPrice,color,length); //call super class constructor**
  - **double getSalePrice(); //Iflength>20feet,5%discount,Otherwise,10%discount.**
- **Create MyOwnAutoShop class which contains the main() method. Perform the following within the main() method.**
  - **Create an instance of Sedan class and initialize all the fields with appropriate values. Use super(...) method in the constructor for initializing the fields of the superclass.**
  - **Create two instances of the Ford class and initialize all the fields with appropriate values. Use super(...) method in the constructor for initializing the fields of the super class.**
  - **Create an instance of Car class and initialize all the fields with appropriate values.**
  - **Display the sale prices of all instance.**

ACE Inspiration

Thank you!!

Q&As

References

- http://www.tutorialspoint.com/java/
- http://www.javatpoint.com/java/
- http://stackoverflow.com/questions/4014535/differences-in-boolean-operators-vs-and-vs
- https://examples.javacodegeeks.com/