Object Oriented Programming with Java 8

Akshita Chanchlani

Contents

- Inheritance

Advantages of OOPS

1. To achieve simplicity

2. To achieve data hiding and data security.

3. To minimize the module dependency so that failure in single part should not stop complete system.

4. To achieve reusability so that we can reduce development time/cost/efforts.

5. To reduce maintenance of the system.

6. To fully utilize hardware resources.

7. To maintain state of object on secondary storage so that failure in system should not impact on data.

Major and Minor pillars of oops

- 4 Major pillars
  1. Abstraction
  2. Encapsulation
  3. Modularity
  4. Hierarchy

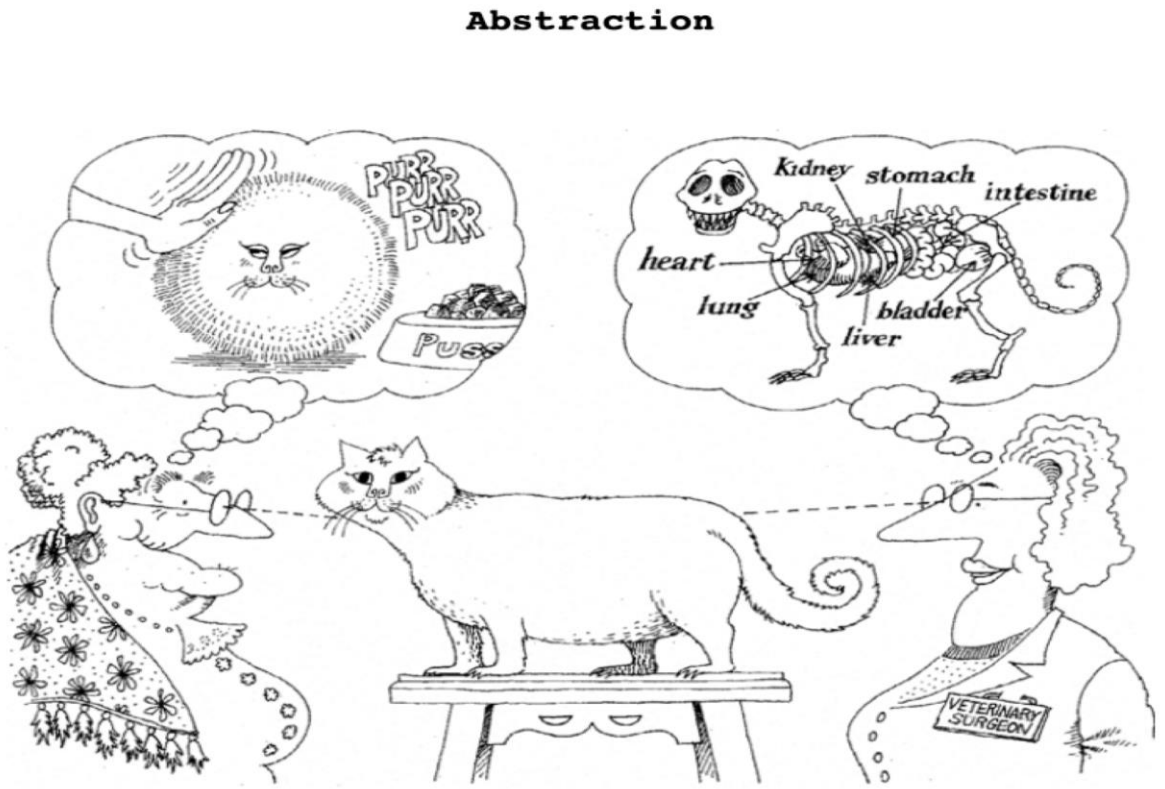
- 3 Minor Pillars
  1. Typing
  2. Concurrency
  3. Persistence

Abstraction

- It is a major pillar of oops.
- **It is a process of getting essential things from object.**
- It describes outer behaviour of the object.
- Abstraction focuses on some essential characteristics of object relative to the perspective of viewer. In other words, abstraction changes from user to user.
- **Using abstraction, we can achieve simplicity.**
- Abstraction in Java

```
Complex c1 = new Complex( );
c1.acceptRecord( );
c1.printRecord( );
```

Abstraction



Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

Encapsulation

- It is a major pillar of oops.

- Definition:
  1. **Binding of data and code together is called encapsulation.**
  2. To achieve abstraction, we should provide some implementation. It is called encapsulation.

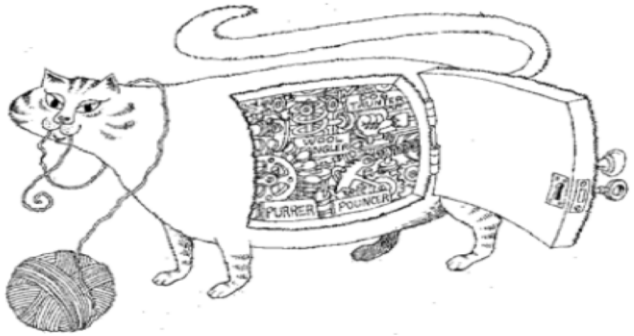
- Encapsulation represents, internal behaviour of the object.

- **Using encapsulation we can achieve data hiding.**

- Abstraction and encapsulation are complementary concepts: **Abstraction focuses on the observable behaviour** of an object, whereas **encapsulation focuses on the implementation** that gives rise to this behaviour.

Encapsulation

Encapsulation



Encapsulation hides the details of the implementation of an object.

Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.

Modularity

- It is a major pillar of oops.
- It is the process of developing complex system using small parts.
- **Using modularity, we can reduce module dependency.**
- We can implement modularity by creating library files.
  - .lib/.a, .dll / .so files
  - .jar/.war/.ear in java

Hierarchy

- It is a major pillar of oops.
- **Level / order / ranking of abstraction is called hierarchy.**
- Main purpose of hierarchy is **to achieve reusability.**
- Advantages of code reusability
    1. We can reduce development time.
    2. We can reduce development cost.
    3. We can reduce developers effort.

- Types of hierarchy:
    1. **Has-a** / Part-of => Association
    2. **Is-a** / Kind-of => Inheritance / Generalization
    3. **Use-a** => Dependency
    4. **Creates-a** => Instantiation

Typing

- It is a minor pillar of oops.
- Typing is also called as polymorphism.
- Polymorphism is a Greek word. Polymorphism = Poly(many) + morphism(forms).
- **An ability of object to take multiple forms is called polymorphism.**
- **Using polymorphism, we can reduce maintenance of the system.**
- Types of polymorphism:
  - **Compile time polymorphism**
    - It is also calling static polymorphism / **Early binding** / Weak Typing / False polymorphism.
    - We can achieve it using:
      1. **Method Overloading**
  - **Run time polymorphism**
    - It is also calling dynamic polymorphism / **Late binding** / Strong Typing / True polymorphism.
    - We can achieve it using:
      1. **Method Overriding.**

Concurrency

- It is a minor pillar of oops.
- In context of operating system, it is called as multitasking.
- It is the process of executing multiple task simultaneously.
- Main purpose of concurrency is to utilise CPU efficiently.
- In Java, we can achieve concurrency using thread.

Persistence

- It is a minor pillar of oops.
- It is process of maintaining state of object on secondary storage.
- In Java, we can achieve Persistence using file and database.

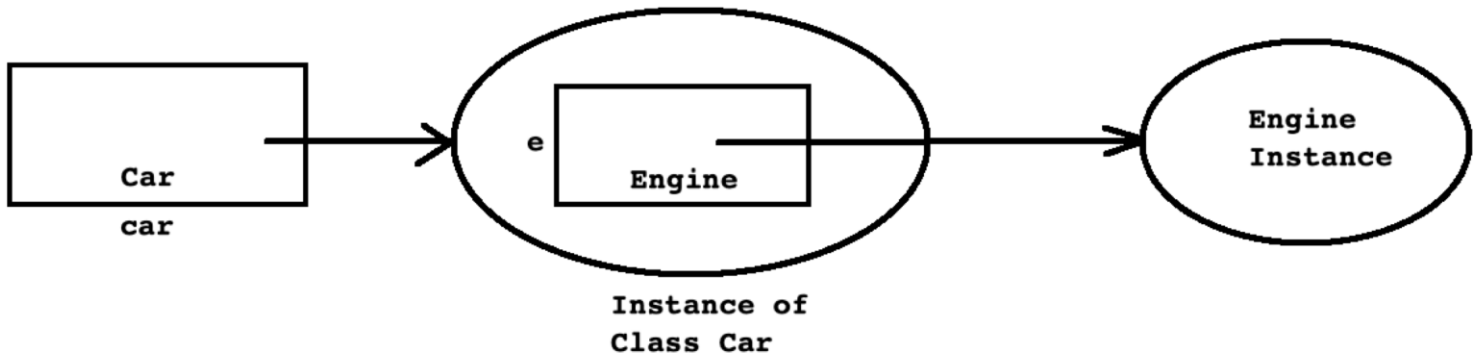
Association

- If has-a relationship is exist between the types then we should use association.
- Example
    1. Car has a engine
    2. Room has a chair
- Let us consider example of car and engine:
    1. Car has a engine
    2. Engine is part of Car.
- If object/instance is a part/component of another instance then it is called as association.
- To implement association, we should declare instance of a class as a field inside another class.

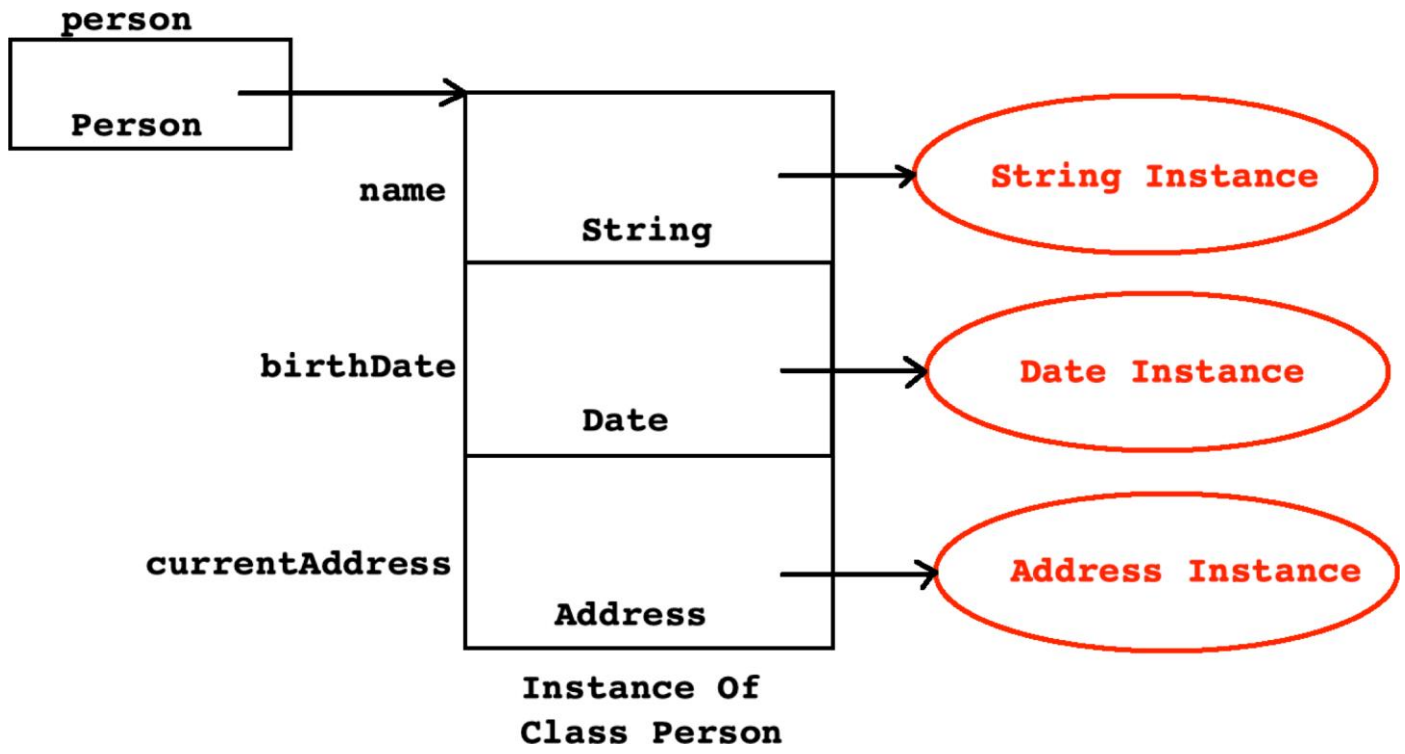
Association In Java

- Engine is part of Car

```
class Engine{
  //TODO
}
class Car{
  Engine e = new Engine( ); //Association
}
Car c = new Car( );
```



Association In Java



Inheritance

- If "is-a" relationship is exist between the types then we should use inheritance.
- Inheritance is also called as generalization.
- Example
    1. Manager is a employee
    2. Book is a product
    3. Triangle is a shape
    4. SavingAccount is a account.

```
class Employee{ //Parent class
    //TODO
}
class Manager extends Employee{ //Child class
    //TODO
}
//Here class Manager is extended from class Employee.
```

Inheritance

- If we want to implement inheritance then we should use extends keyword.
- In Java, parent class is called as super class and child class is called as sub class.
- Java do not support private and protected mode of inheritance
- If Java, class can extend only one class. In other words, multiple class inheritance is not allowed.
- Consider following code:

```
class A{    }
class B{    }
class C extends A, B{    //Not OK
}
```

Inheritance

- During inheritance, if super type and sub type is class, then it is called as implementation inheritance.

Single implementation Inheritance
`class A{ }`
`class B extends A{ } //OK`
	Hierarchical implementation Inheritance
	`class A{ }`
	`class B extends A{ } //OK`
	`class C extends A{ } //OK`

Multiple implementation Inheritance
`class A{ }`
`class B{ }`
`class C extends A, B{ } //Not OK`
	Multilevel implementation inheritance
	`class A{ }`
	`class B extends A{ } //OK`
	`class C extends B{ } //OK`

Multiple Inheritance In Java

```
class A{    }
class B{    }
class C extends A, B{   //Not OK : Multiple implementation inheritance
    //TODO
}
```

```
interface A{    }
interface B{    }
interface C extends A, B{   //OK : Multiple interface inheritance
    //TODO
}
```

```
interface A{    }
interface B{    }
class C implements A, B{   //OK : Multiple interface implementation inheritance
    //TODO
}
```

Inheritance

- If we create instance of sub class then all the non static fields declared in super class and sub class get space inside it. In other words, non static fields of super class inherit into sub class.
- Static field do not get space inside instance. It is designed to share among all the instances of same class.
- Using sub class, we can access static fields declared in super class. In other words, static fields of super class inherit into sub class.
- All the fields of super class inherit into sub class but only non static fields gets space inside instance of sub class.
- Fields of sub class, do not inherit into super class. Hence if we create instance of super class then only non static fields declared in super class get space inside it.
- If we declare field in super class static then, all the instances of super class and sub class share single copy of static field declared in super class.

Inheritance

- We can call/invoke, non static method of super class on instance of sub class. In other words, non static method inherit into sub class.

- We can call static method of super class on sub class. In other words, static method inherit into sub class.

- Except constructor, all the methods of super class inherit into sub class.

Inheritance

- If we create instance of super class then only super class constructor gets called. But if we create instance of sub class then JVM first give call to the super class constructor and then sub class constructor.
- From any constructor of sub class, by default, super class's parameterless constructor gets called.
- Using super statement, we can call any constructor of super class from constructor of sub class.
- Super statement, must be first statement inside constructor body.

Inheritance

- According to client's requirement, if implementation of super class is logically incomplete / partially complete then we should extend the class. In other words we should use inheritance.
- According to client's requirement, if implementation of super class method is logically incomplete / partially complete then we should redefine method inside sub class.
- Process of redefining method of super class inside sub class is called as method overriding.

Regarding this and super

this(...) implies invoking constructor from the same class.
super(...) implies invoking constructor from the immediate super class

1. Only a constr can use this() or super()
2. Has to be 1st statement in the constructor
3. Any constructor can never have both ie. this() & super()
4. super & this (w/o brackets) are used to access (visible) members of super class or the same class.

Thank You.

akshita.chanchlani@sunbeaminfo.com