JAVA FLUENT API DESIGN

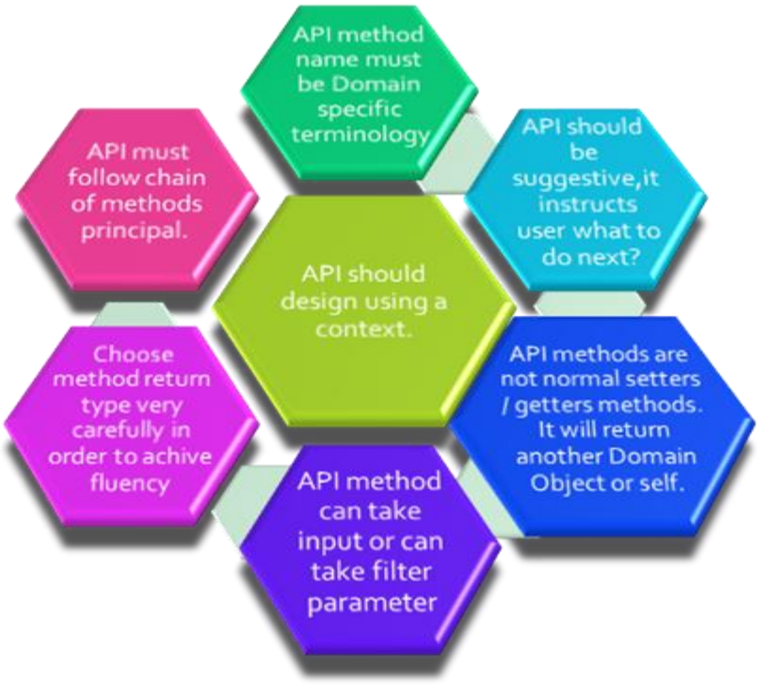
JAVA FLUENT API

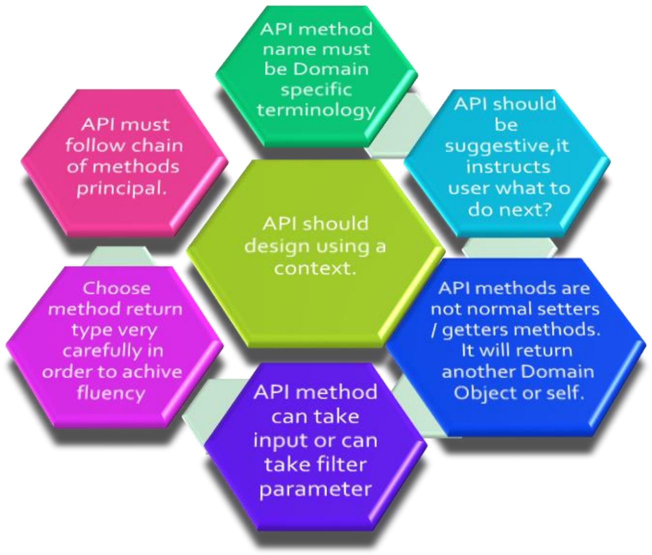
In this article we will discuss about how to design fluent API in Java. The term *Fluent interface* is coined by *Martin Fowler* and *Eric Evans*. Fluent API means, build an API in such way so that same meets following criteria.

a. API user can understand API very easily.
b. API can perform a series of actions in order to finish a task, in java we can do it by series of method calls (Chaining of methods).
c. Each Methods name should be Domain specific terminology.
d. API should suggestive enough to guide API Users, what to do next and What are the possible operations users can take at a particular moment.

Suppose You want to Design an API for a Domain, say (Retail) so There should be some common terminology exists in Retail domain and for a certain context(Task) it will take a series of actions to finish this task. Say for an invoice generation it has to follow certain steps. Now when you design API, you should design it such a way, when API Users call Billing Service for invoice generation, API User can fluently perform each step in order to complete invoice generation and API will assist user to perform those steps upon invoking Billing Service. When a API method invokes by an user, method will perform its task and returns a Domain Object which will assist what to do next, until all steps are executed. Unlike Standard API it is API user job to call API methods in a sequential way to successfully performs a task. So API Users has to know about the service steps very well.

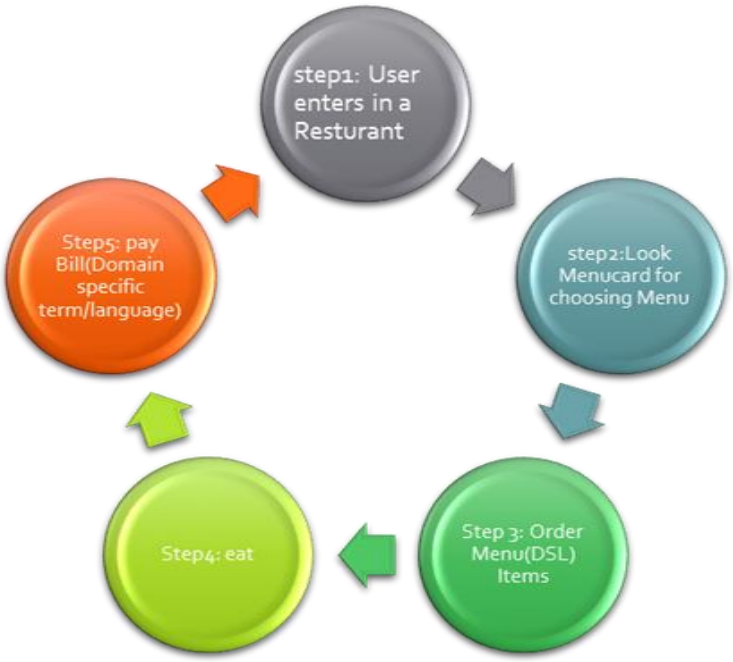
Design A Fluent API:

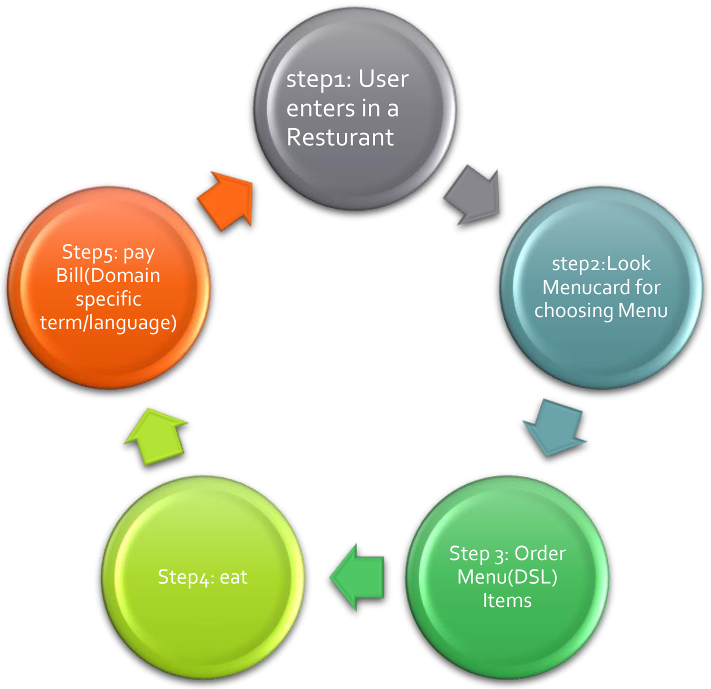

API method name must be Domain specific terminology
API must follow chain of methods principal.
API should be suggestive,it instructs user what to do next?
API should design using a context.
Choose method return type very carefully in order to achive fluency
API methods are not normal setters / getters methods. It will return another Domain Object or self.
API method can take input or can take filter parameter


API method name must be Domain specific terminology
API must follow chain of methods principal.
API should be suggestive,it instructs user what to do next?
API should design using a context.
Choose method return type very carefully in order to achive fluency
API methods are not normal setters / getters methods. It will return another Domain Object or self.
API method can take input or can take filter parameter

Example: Suppose we want to design a API for Restaurant.

As a customer of this Restaurant, one should follow below steps


step1: User enters in a Resturant
step2:Look Menucard for choosing Menu
Step 3: Order Menu(DSL) Items
Step4: eat
Step5: pay Bill(Domain specific term/language)


step1: User enters in a Resturant
step2:Look Menucard for choosing Menu
Step 3: Order Menu(DSL) Items
Step4: eat
Step5: pay Bill(Domain specific term/language)

In a Standard API design, we should do the following:

1. Create a Restaurant interface.
2. Create an Implementation class of Restaurant interface. Compose Menucard class into it.
3. Create getters and setters for restaurant properties like name, address.
4. In MenuCard class maintain a List of menu Items . Expose some methods like showmenu(). Ordermenu(); etc.
5. Each menu Items has name and cost properties and corresponding getters/setters.
6. When API User call this API he/she will call a sequence of methods(Enter Resturent,call showMenu() then Ordermenu() etc.) to perform above Steps shown in the picture.

So it is not fluent lot of sequential statement needs to perform to complete task and API user has to know the sequence .

Now I will show you How we will design a Fluent API.

Java code:

```
package com.example.fluentapi.contract;

public interface IResturant {
        public IResturant name(String name);
        public IMenu show();


}




package com.example.fluentapi.contract;

public interface IMenu{

        public IMenu order(int index);
        public IMenu eat();
        public IMenu pay();
        public IItem get(int index);

}


package com.example.fluentapi.contract;

public interface IItem {

        public IItem name();
        public Integer cost();


}
```

Implementation:

```
package com.example.fluentapi.impl;



import com.example.fluentapi.contract.IMenu;

import com.example.fluentapi.contract.IResturant;



public class Arsalan implements IResturant{



        String name;

        String IMenu;
```

```
	public IResturant name(String name) {

		this.name=name;

		System.out.println("Enter to hotel :: " + name);

		return this;

	}




	public IMenu show() {

		// TODO Auto-generated method stub

		ArsalanMenuHandler handler = new ArsalanMenuHandler();

		handler.showMenu();

		return handler;

	}


}


package com.example.fluentapi.impl;


import java.util.ArrayList;

import java.util.List;


import com.example.fluentapi.contract.IItem;

import com.example.fluentapi.contract.IMenu;


public class ArsalanMenuHandler implements IMenu{
```

```
List<IItem> menuList = new ArrayList<IItem>();

List<IItem> selectedList = new ArrayList<IItem>();


public ArsalanMenuHandler()

{

        IItem biriyani = new IItem(){

                public IItem name()

                {

                        System.out.println("Mutton Biriyani");

                        return this;

                }

                public Integer cost()

                {

                        return 180;

                }

        };

        IItem muttonChap = new IItem(){

                public IItem name()

                {

                        System.out.println("Mutton Chap");

                        return this;

                }

                public Integer cost()

                {

                        return 160;
                }

            };

            IItem firni = new IItem(){

                public IItem name()

                {

                    System.out.println("Firni");

                    return this;

                }

                public Integer cost()

                {

                    return 100;

                }



            };

            menuList.add(biriyani);

            menuList.add(muttonChap);

            menuList.add(firni);


        }

        public IMenu order(int index) {

            // TODO Auto-generated method stub

            IItem item =get(index);

            selectedList.add(item);

            System.out.println("Order given ::");

            item.name();
        return this;
}



public IMenu eat() {
        for(IItem item : selectedList)
        {
                System.out.println("eating ");
                item.name();
        }
        return this;
}



public IMenu pay() {
        int cost=0;
        for(IItem item : selectedList)
        {
                cost = cost + item.cost();
        }
        System.out.println("Paying Ruppes" + cost);
        return this;
}
@Override
public IItem get(int index) {
        // TODO Auto-generated method stub
```

```
                if(index <3)

                {

                        return menuList.get(index);

                }

                return null;


        }


        public void showMenu(){

                System.out.println("MENU IN ARSALAN");

                for(IItem item : menuList)

                {


                        item.name();


                }


        }


}
```

Test Fluent API:

```
package com.example.fluentapi.impl;

public class FluentApiTest {

        public static void main(String[] args) {

                new Arsalan().name("ARSALAN").show().order(0).order(1).eat().pay();
        }
}
```

Output :

```
Enter to hotel :: ARSALAN
MENU IN ARSALAN
Mutton Biriyani
Mutton Chap
Firni
Order given ::
Mutton Biriyani
Order given ::
Mutton Chap
eating
Mutton Biriyani
eating
Mutton Chap
Paying Ruppes340
```

Look How we perform the steps fluently by calling series of methods

```
new Arsalan().name("ARSALAN").show().order(0).order(1).eat().pay();
```

This is a test example, to make it more polished we need to work on following things

1. Order method should take MenuItem instead of position also needs to handle Exception scenarios. If a MenuItem not found.
2. Order can be cancelled.
3. Pay must have payment mode. (creditcard, debitcard, Cash)
4. May user can pay tips and it should be optional.
5. In course of eating, he/she can Order more items, so make an order functionality should be available in course of eating.
6. Tax calculation should be added in pay method.

Java 8 using Fluent API.