INTERFACE

Before getting directly into the definition of Interface, let us understand it through an example.

Let us consider a software, from the software engineer's view it is internally divided into three parts.

Front-end

Front-end Development *is responsible for implementing visual elements that users see and interact within a web application.*

A front-end is like a dummy, it makes software to look good, you can type things but it is not capable of execution.

To create Front-end few technologies used are:

Back-end

It is in the ***Back-end*** *all the execution and processing happens with the help of programming languages. This is not visible to the user.*

Backend development works in tandem with the front end to deliver the final product to the end user.

To create Back-end few technologies used are:

Database

A **Database** *is a collection of information that is organized so that it can be easily accessed, managed and updated.*

Code written by back end developers is what communicates the database information to the browser.

To take care of databases there are database Management software's.

The few Database Management Systems software's are:

ORACLE Informix

The Front-end interacts with the back-end; the back-end understands the requirement of the front-end and supports it. If there is need of some data then back-end interacts with Database takes the data, processes it and gives back to the front-end. Therefore, software works with the combination of all three.

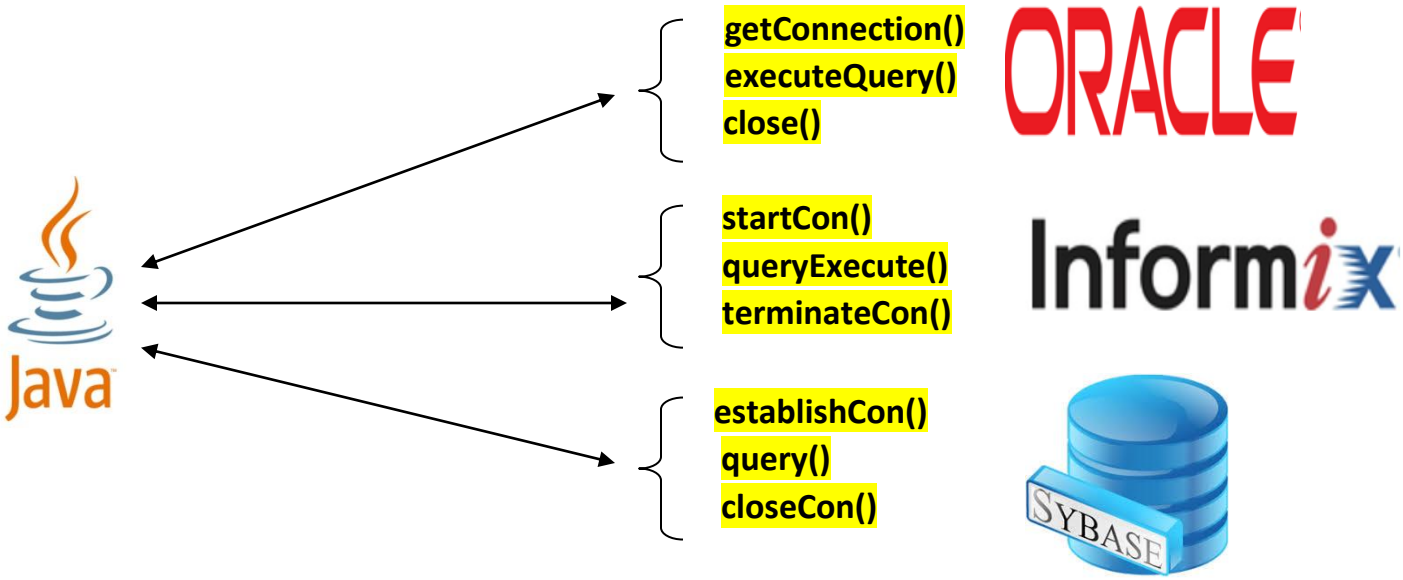
Now you might be wondering what these have to do with Interface??

Initially Java didn't have the facility to connect to the database. It was in 1997, a feature in Java was introduced ***using which a Java program can be connected to the Database Management Software.*** *And the technology which gave Java as a programming language the ability to interact/connect to Database Management Software is what called as* ***"Java Database Connectivity"****.*

If a programmer through a Java program wants to connect to the database what should be the method name? if he/she wants to send a query to the database what should be method name? Once all the execution is done, to terminate the connection what should be the method name??

Query ***is a request for data or information from a database.***
Databases store data in a structured format, which can be accessed using queries and the special type of language which is used for this purpose is called as Structured Query Language(SQL).

So as a solution for above question each Database Management Software came up with their own method names.



But this is not the good approach of programming.

***Well, let me explain you why*?**

Imagine a Java programmer has got a project which had to be worked in Oracle Database, he remembers the method names for establishing the connection, execute the query, many other method names as per the requirement and finally closes the connection and completes the project.
Now assume, he got another project but now in which he has to work with Informix Software.
Now don't you think the method names for this database software is different and he has to remember all those methods too.

Therefore, Java giving the flexibility to the Database Management Software to decide their own methods the, Java programmer were put into trouble. Because every time the Database software changes the programmer had to remember different method names.

So as a solution Java developed a common methods names irrespective of what the Database software it is.

But how??? 

Java came up with a contract in which the method names would be mentioned and the Database software's had to use the same method names as in the contract.

(Contract)

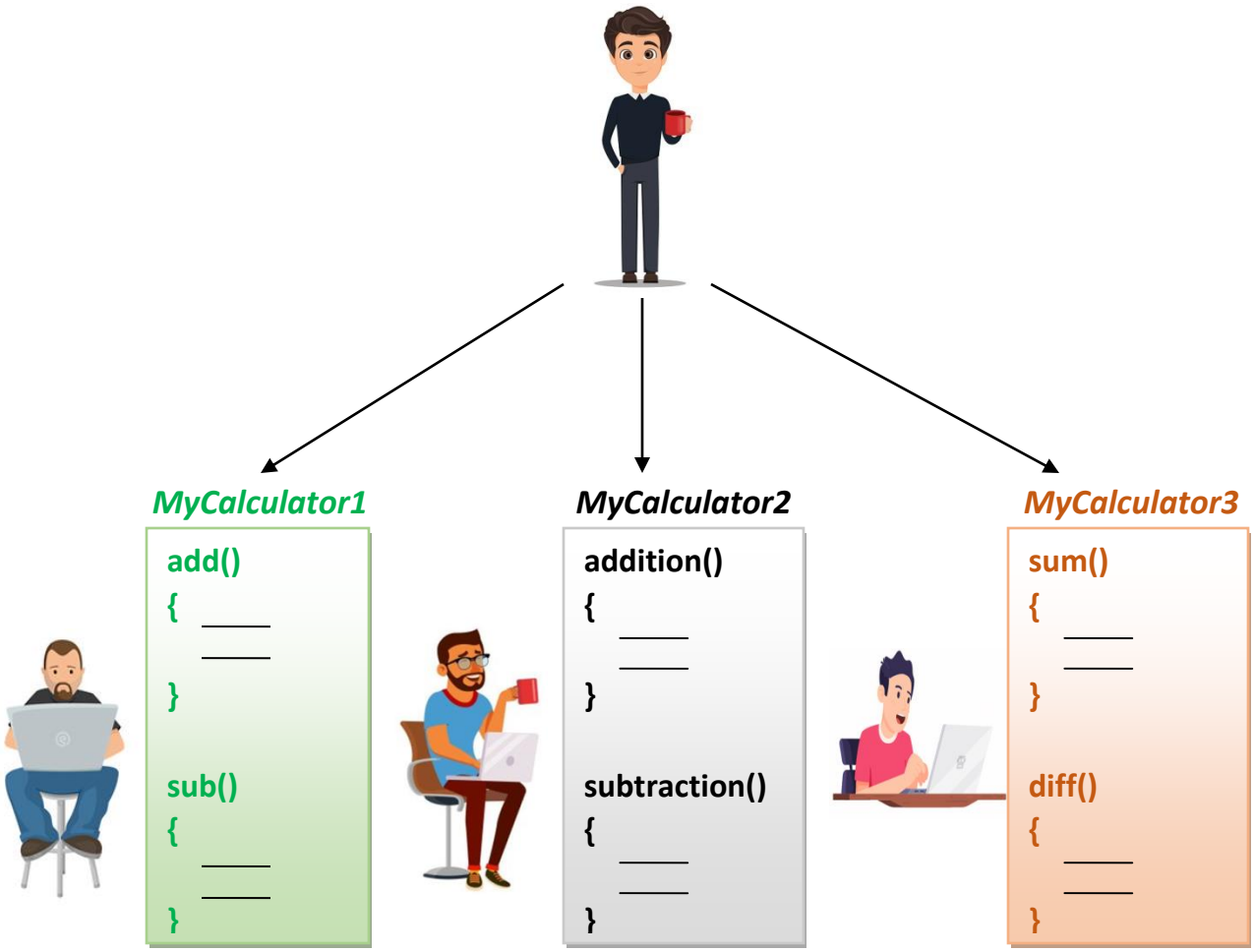
Interface Connection

< >
getConnection();
executeQuery();
close();

An Interface is like a contract which when implemented helps achieves standardization. Whatever mentioned in the contract must be used as it is without any changes. An interface can contain any number of methods.

Let us see a scenario which explains Interface.

Let us consider that a function of addition and subtraction is to be performed for a user. Therefore, let us imagine that there are three developers who came up with three different classes which has different method names to perform the same function of addition and subtraction, like has shown in the below representation.



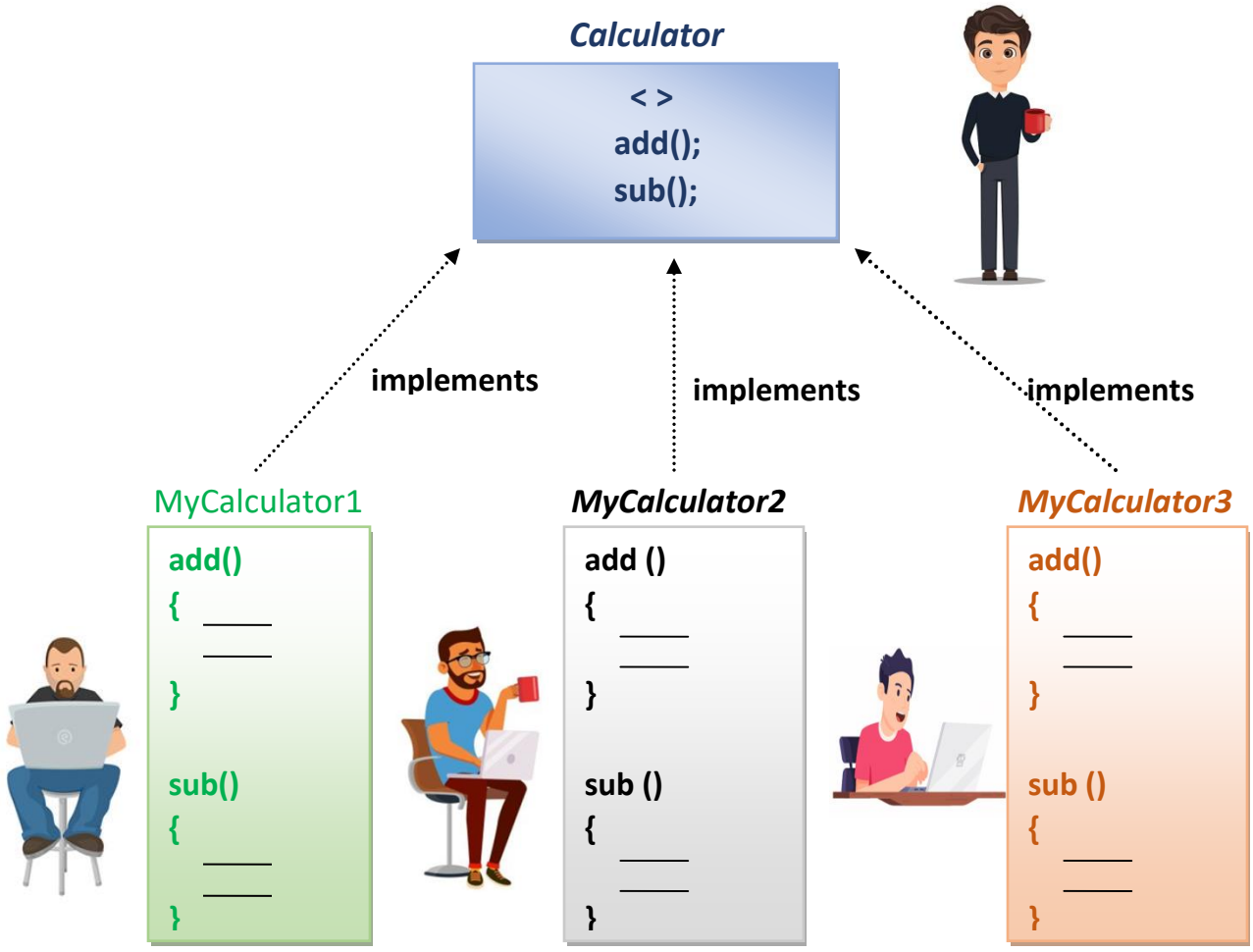
The problem with this approach was, to perform the same function as addition and subtraction three different types of methods have been used. The difficulty here is for the user to remember different names of the methods and the user friendliness misses.

That is, if a developer creates the method names it is a problem so the best would be to create a contract and mention the method names that have to be used by all three developers with the body of the method written as per their requirement.

Hence they came up with the contract/ Interface.

Interfaces are represented with rectangular box with angular braces inside it.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.



implements means MyCalculator classes will give body to the methods of add() and sub() of Calculator interface but will never change the name of the methods.

The logic of operations of three developers may be different but the method names must be same.

Note:

Interface promotes polymorphism. An interface type reference can point to implementing class objects. This achieves loose coupling and hence code reduction and code flexibility.

Methods within an interface are automatically public abstract.

Let us now write code to understand interface:

```
import java.util.Scanner;
interface Calculator
{
    void add();
    void sub();
}

class MyCalculator1 implements Calculator
{
    public void add()
    {
        int a = 20;
        int b = 10;
        int c = a+b;
        System.out.println(c);
    }
    public void sub()
    {
        int a = 20;
        int b = 10;
        int c = a-b;
        System.out.println(c);
    }
}
class MyCalculator2 implements Calculator
{
    public void add()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the first number");
        int a = scan.nextInt();
        System.out.println("Enter the second number");
        int b = scan.nextInt();
        int c = a+b;
        System.out.println(c);
    }
    public void sub()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the first number");
        int a = scan.nextInt();
        System.out.println("Enter the second number");
        int b = scan.nextInt();
        int c = a-b;
        System.out.println(c);
    }
}
```

```java
class MyCalculator3 implements Calculator
{
    public void add()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the first number");
        int a = scan.nextInt();
        System.out.println("Enter the second number");
        int b = scan.nextInt();
        if(b == 0)
        {
            System.out.println("Second number is zero");
        }
        else
        {
            int c = a+b;
            System.out.println(c);
        }
    }
    public void sub()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the first number");
        int a = scan.nextInt();
        System.out.println("Enter the second number");
        int b = scan.nextInt();
        if(b == 0)
        {
            System.out.println("Second number is zero");
        }
        else
        {
            int c = a-b;
            System.out.println(c);
        }
    }
}
class Demo
{
    public static void main(String[] args)
    {
        MyCalculator1 mc1 = new MyCalculator1();
        MyCalculator2 mc2 = new MyCalculator2();
        MyCalculator3 mc3 = new MyCalculator3();

        mc1.add();
        mc1.sub();
        mc2.add();
        mc2.sub();
        mc3.add();
        mc3.sub();
    }
}
```