## Contents

1. OOP definition and Steve Jobs version of it
2. Class - Blueprint and Objects - implementation of blueprint
3. How to create classes & objects, access values and change values of data members
4. Access Modifiers
5. Exercise: Implement Fraction Class

=====================================================================

## Introduction

**Object Oriented Programming** (OOP) is a way of programming that helps us to write maintainable code. Maintainable code is:

1. Easy to understand
2. Easy to change
3. Not susceptible to errors.

OOP does so by mimicking real life interactions in code. Steve Jobs explains OOP in the best way possible. Following are the excerpts from Steve Jobs interview where he explains OOP. Give it a read:

*Jeff Goodell: Would you explain, in simple terms, exactly what object-oriented software is?*

*Steve Jobs: Objects are like people. They're living, breathing things that have knowledge inside them about how to do things and have memory inside them so they can remember things. And rather than interacting with them at a very low level, you interact with them at a very high level of abstraction, like we're doing right here.*

*Here's an example: If I'm your laundry object, you can give me your dirty clothes and send me a message that says, "Can you get my clothes laundered, please." I happen to know where the best laundry place in San Francisco is. And I speak English, and I have dollars in my pockets. So I go out and hail a taxicab and tell the driver to take me to this place in San Francisco. I go get your clothes laundered, I jump back in the cab, I get back here. I give you your clean clothes and say, "Here are your clean clothes."*

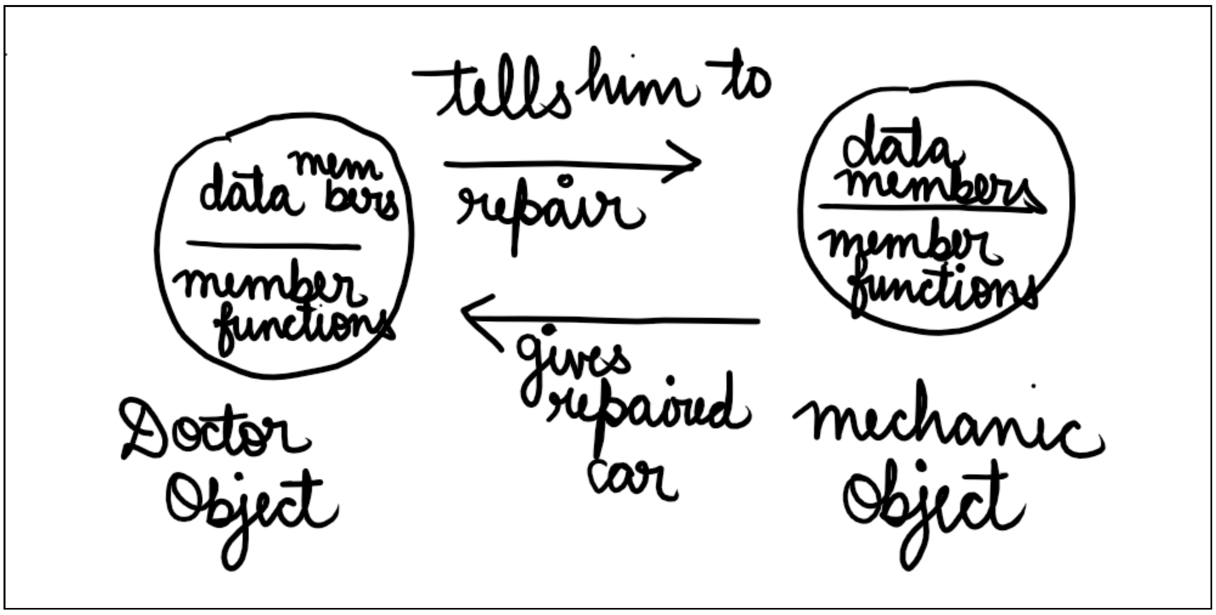
*You have no idea how I did that. You have no knowledge of the laundry place. Maybe you speak French, and you can't even hail a taxi. You can't pay for one, you don't have dollars in your* *pocket. Yet, I knew how to do all of that. And you didn't have to know any of it. All that complexity was hidden inside of me, and we were able to interact at a very high level of abstraction. That's what objects are. They encapsulate complexity, and the interfaces to that complexity are high level.*

## Analogy of real world interactions and OOP interactions

Another example to explain OOP is interaction between a doctor and a mechanic. The doctor's car has broken and he wants it repaired. He will ask the mechanic for the service and in turn of favour, he will pay.

In OOP world, each object is seen as combination of two things:

1. Things that object knows [Information or Data members of the object]
2. Things that can be achieved by manipulating the information [Functions or Member functions of the object]



The above communication involves 3 features of OOP:

1. Encapsulation: Keeping information and functions together

   Data members and member functions for doctor and mechanic objects are together combined in one entity or container

2. Abstraction: Hiding unnecessary details

   Doctor doesn't have to worry about how the mechanic repairs the car.

3. Message Passing: Communication between objects

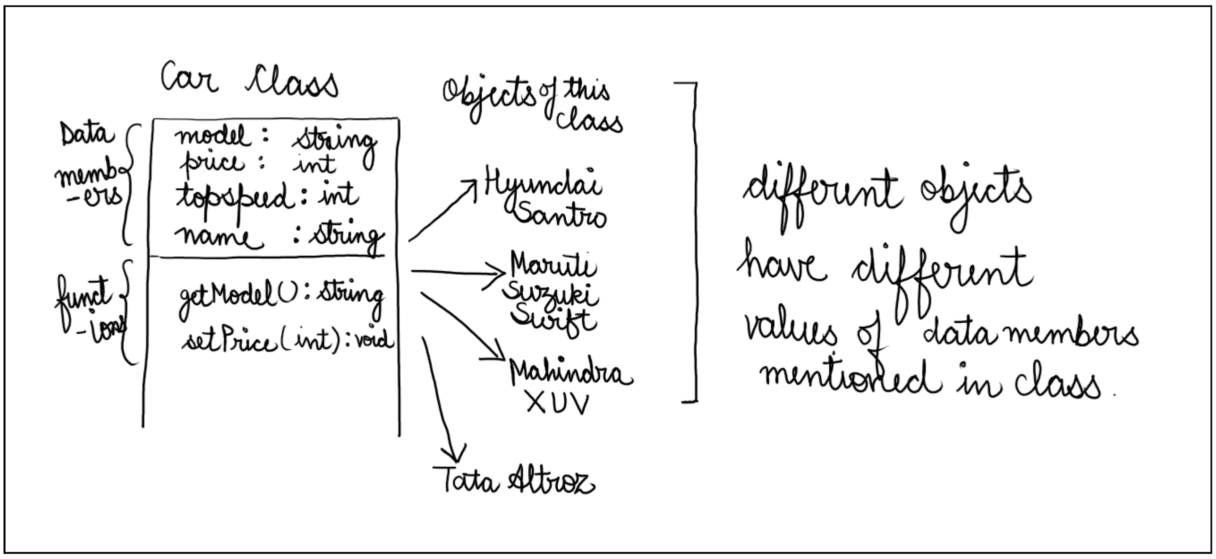
   Doctor objects calls function of mechanic object to get the car repaired.

## Classes and Objects

Now, let us understand about classes. From the above discussion, we can conclude that there can be many doctor as well as mechanic objects. And in real life also, we can observe so many doctors such as orthopaedics, surgeons and many others. Now, there is a lot of commonality between doctor objects. The commonality is called the **class**. All the objects originate from the class.

Class is the blueprint and the objects are instances of the blueprint.

Let us take an example of Car class and its objects:



Hence, class is the blueprint which consists of data members and functions. Objects are instances of class. Each object has a different value of data members mentioned in class.

## How to create Classes and Objects, how to access its variables/ data members and how to change values of data members

Class Code:

```
public class Student {
        // information
        // data members
        private String name;
        private int rollno;
```

```
	// what it can do with this information
	// functions/ methods

}
```

Driver Class to instantiate objects of this class:

```
public class StudentUse {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Student s1 = new Student();

		System.out.println(s1.rollno + " " + s1.name);

		/*
			The output will be 0 null, as these are the default values of int and
		string in Java
		*/

		s1.rollno=1;
		s1.name="Akshima";
		System.out.println(s1.rollno + " " + s1.name);

		/*
			The output will be 1 Akshima, as these are the default values of int
		and string in Java
		*/

	}

}
```

## Access Modifiers

We are going to talk about private, public and default. They are used to restrict access to variables/ data members and member functions of a class.

Code of Student Class using access modifiers:

```
/*
	This code is used to restrict illegal setting of roll numbers. This ensures that roll
number is never set to negative.
```

```
*/

public class Student {
	// information
	// data members
	private String name;
	private int rollno;

	// what it can do with this information
	// functions/ methods

	public void setrollno(int rollno) {
		if (rollno < 0) {
			return;
		}

		this.rollno = rollno;
	}

	public void setname(String name) {
		this.name = name;
	}

	public int getrollno() {
		return this.rollno;
	}

	public String getname() {
		return this.name;
	}

}
```

Driver Class:

```
public class StudentUse {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Student s1 = new Student();

		System.out.println(s1.getrollno() + " " + s1.getname());

		s1.setrollno(1);
		s1.setname("Akshima");
```

```
                System.out.println(s1.getrollno() + " " + s1.getname());
        }

}
```

**Thing to remember**:
Private access modifier restricts access to the same class. Variables and functions with private access modifier can be accessed within the same class.

Default access modifier restricts access to the same package. Variables and functions with default access modifier can be accessed within the same package.

Public access modifier can be accessed from anywhere. Variables and functions with public access modifier can be accessed from anywhere.

## Exercise:

Complete this Fraction Class:

```
public class Fraction {
        private int numerator;
        private int denominator;

        public Fraction(int numerator, int denominator) {
                this.numerator = numerator;
                if (denominator == 0) {
                        // TODO error out
                }
                this.denominator = denominator;
                simplify();
        }


        public int getDenominator() {
                return denominator;
        }
```

```
public int getNumerator() {
        return numerator;
}

public void setNumerator(int n) {
        this.numerator = n;
        simplify();
}


public void setDenominator(int d) {
        if (d == 0){
                // TODO error out
                return;
        }
        this.denominator = d;
        this.simplify();
}

public void print() {
        if (denominator == 1) {
                System.out.println(numerator);
        } else {
                System.out.println(numerator + "/" + denominator);
        }


}

private void simplify() {

}

public static Fraction add(Fraction f1, Fraction f2) {

}

public void add(Fraction f2) {

}

public void multiply(Fraction f2) {

}
```

```
}
```

**Solution**:

```
public class Fraction {
	private int numerator;
	private int denominator;

	public Fraction(int numerator, int denominator) {
		this.numerator = numerator;
		if (denominator == 0) {
			// TODO error out
		}
		this.denominator = denominator;
		simplify();
	}

	public int getDenominator() {
		return denominator;
	}

	public int getNumerator() {
		return numerator;
	}

	public void setNumerator(int n) {
		this.numerator = n;
		simplify();
	}

	public void setDenominator(int d) {
		if (d == 0) {
			// TODO error out
			return;
		}
		this.denominator = d;
		this.simplify();
	}

	public void print() {
		if (denominator == 1) {
			System.out.println(numerator);
		} else {
```

```
                System.out.println(numerator + "/" + denominator);
            }
        }

        private void simplify() {
            int gcd = 1;
            int smaller = Math.min(numerator, denominator);
            for (int i = 2; i <= smaller; i++) {
                if (numerator % i == 0 && denominator % i == 0) {
                    gcd = i;
                }
            }
            numerator = numerator / gcd;
            denominator = denominator / gcd;
        }
        public static Fraction add(Fraction f1, Fraction f2) {
            int newNum = f1.numerator * f2.denominator + f2.numerator * f1.denominator;
            int newDen = f1.denominator * f2.denominator;
            Fraction f = new Fraction(newNum, newDen);
            return f;
        }

        public void add(Fraction f2) {
            this.numerator = this.numerator * f2.denominator + this.denominator *
    f2.numerator;
            this.denominator = this.denominator * f2.denominator;
            simplify();
        }

        public void multiply(Fraction f2) {
            this.numerator = this.numerator * f2.numerator;
            this.denominator = this.denominator * f2.denominator;
            simplify();
        }

    }
```