Design Pattern: Builder

This solves the problem of having to create a constructor with a lot of parameters. This is solved by having a builder class that has a lot of setters that returns the builder object. This is a creation pattern.

This also hides the creation of the object from the client. The client only has to call the builder class and the builder class will create the object for you.

Say you have a car class that has a lot of parameters, you could do this:

```
public class CompanyEmployee{
    private int id;
    private String name;
    private int age;
    private String address;
    private String phoneNumber;


    public CompanyEmployee(int id, String name, int age, String address,
String phoneNumber){
        this.id = id;
        this.name = name;
        this.age = age;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }

}
```

There are three issues with this:

1. You have to remember the order of the parameters
2. You have to pass in a lot of parameters that you might not need
3. The constructor is not very readable.

The builder makes it so you can customize what you want to define in the constructor.

The builder pattern suggest that we extract the object construction code out of the object's own class and move it to separate object, called the builder.

```
public class CompanyEmployeeBuilder{

    public CompanyEmployee e = new CompanyEmployee();

    public CompanyEmployeeBuilder id(int id){
        e.setId(id);
        return this;
    }
```

```
    public CompanyEmployeeBuilder name(String name){
        e.setName(name);
        return this;
    }

    public CompanyEmployeeBuilder age(int age){
        e.setAge(age);
        return this;
    }

    public CompanyEmployeeBuilder address(String address){
        e.setAddress(address);
        return this;
    }

    public CompanyEmployeeBuilder phoneNumber(String phoneNumber){
        e.setPhoneNumber(phoneNumber);
        return this;
    }

    public CompanyEmployee build(){
        return e;
    }


}
```

```
public class CompanyEmployee{
    private int id;
    private String name;
    private int age;
    private String address;
    private String phoneNumber;

    public CompanyEmployee(){

    }

    public CompanyEmployee(int id, String name, int age, String address,
String phoneNumber){
        this.id = id;
        this.name = name;
        this.age = age;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }


    public void setId(int id){
        this.id = id;
```

```
    }

}
```

```
public class Main{
    public static void main(String[] args){
        CompanyEmployeeBuilder builder = new CompanyEmployeeBuilder()
            .id(1);

        CompanyEmployee e = builder.build(); // returns a CompanyEmployee
object

    }
}
```

The setters return the builder object. This is so we can chain the setters together. The `this` keyword is used to refer to the current object. Implicit return of (current) instance when leaving the method. Each method should return the object itself allowing the calls to be chained on that same object. This allows you to call multiple methods on the same object in a chain, without the need for separate statements.

The `build()` method creates an instance of the object the build is meant to build.

The Car class has a private constructor that takes in all the parameters. This is so we can't create a Car object without using the builder.