

Builder

Team 11

Builder

The builder design pattern is a type of **creational design** pattern.

Creational design patterns solve problems related to **object creation**.

Motivation

Have a **complex object** that can be created with **many parameters** (some are mandatory, others are optional)

Want to reduce the number of parameters to a constructor, increase design flexibility, code readability.

Intent

Abstract away the construction of a **complex object** so that many different representations can be created from same construction process

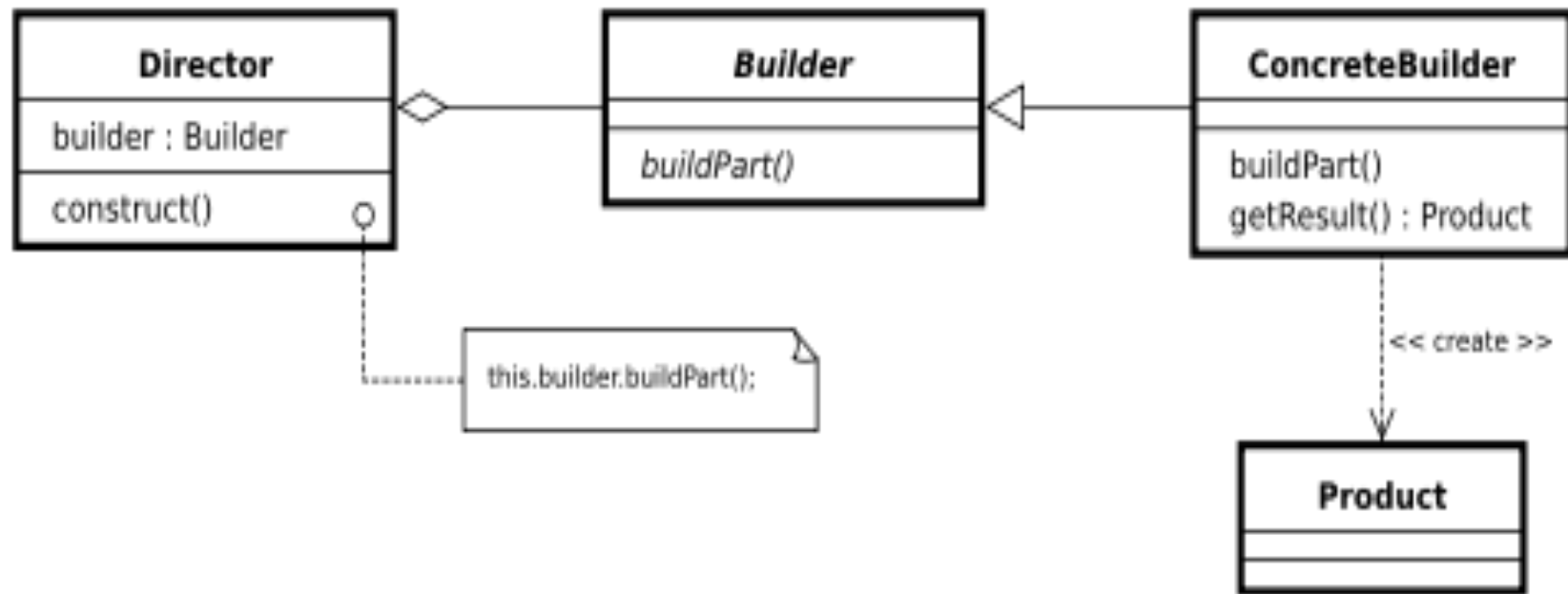
Parse a complex representation and create one of several target products.

Discussion

Separate algorithm for parsing and reading from algorithm for building and representing target products

Have a **director** that invokes **builder** services which create parts of complex object each time it is called and maintains all intermediate states

Components



Components

Director: Construct complex object using Builder interface

Builder: Specify interface for creating product

Concrete builder: Assembles parts of product by implementing builder interface

Product: Complex object that is being built

Example

Object: Canadian Address

Example (Step 1)

In the following implementation
CanadianAddress has **two** responsibilities:

- responsible for both parsing strings representing Canadian addresses
- representing a Canadian address.

Should only have **one**!

Example (Step 1)

```
1 package csc301.builderExample;
2
3
4 public class CanadianAddress {
5
6     private String apartmentNumber;
7     private String streetNumber;
8     private String streetName;
9     private String streetType;
10    private String city;
11    private String province;
12    private String postalCode;
13
14
15    public CanadianAddress(String fullAddress) {
16        parseAddress(fullAddress);
17    }
18
19
20
21    private void parseAddress(String fullAddress) {
22        // Parse the address somehow, and populate the various
23        // instance variables ...
24    }
25
26
27
28
29
30    public String getApartmentNumber() {
31        return apartmentNumber;
32    }
33
34    public String getStreetNumber() {
35        return streetNumber;
36    }
37
38    public String getStreetName() {
39        return streetName;
40    }
41
42    public String getStreetType() {
43        return streetType;
44    }
45
46    public String getCity() {
47        return city;
48    }
49
50    public String getProvince() {
51        return province;
52    }
53
54    public String getPostalCode() {
55        return postalCode;
56    }
57
58 }
```

Example (Step 2)

In next implementation, CanadianAddress only one responsibility but the constructor has **too many arguments and all of same type**.

Developers need to remember order of arguments every time code is modified.

Result: Slows down developers and more error prone code

Example (Step 2)

```
1 package csc301.builderExample;
2
3
4 public class CanadianAddress {
5
6     private String apartmentNumber;
7     private String streetNumber;
8     private String streetName;
9     private String streetType;
10    private String city;
11    private String province;
12    private String postalCode;
13
14
15
16
17    public CanadianAddress(String apartmentNumber, String streetNumber,
18        String streetName, String streetType, String city, String
19        province,
20        String postalCode) {
21        this.apartmentNumber = apartmentNumber;
22        this.streetNumber = streetNumber;
23        this.streetName = streetName;
24        this.streetType = streetType;
25        this.city = city;
26        this.province = province;
27        this.postalCode = postalCode;
28    }
29
```

```
30
31    public String getApartmentNumber() {
32        return apartmentNumber;
33    }
34
35    public String getStreetNumber() {
36        return streetNumber;
37    }
38
39    public String getStreetName() {
40        return streetName;
41    }
42
43    public String getStreetType() {
44        return streetType;
45    }
46
47    public String getCity() {
48        return city;
49    }
50
51    public String getProvince() {
52        return province;
53    }
54
55    public String getPostalCode() {
56        return postalCode;
57    }
58
59 }
60
```

Example (Step 3)

In next, no more telescoping constructor problem

- provided default constructor (0 arguments)
- setters for all properties

New problem: too easy to create **invalid/illegal instances**.
Should throw exception if constructor is provided with
invalid arguments

Example (Step 3)

```
1 package csc301.builderExample;
2
3
4 public class CanadianAddress {
5
6
7     private String apartmentNumber;
8     private String streetNumber;
9     private String streetName;
10    private String streetType;
11    private String city;
12    private String province;
13    private String postalCode;
14
15
16
17    public void setApartmentNumber(String apartmentNumber) {
18        this.apartmentNumber = apartmentNumber;
19    }
20
21    public void setStreetNumber(String streetNumber) {
22        this.streetNumber = streetNumber;
23    }
24
25    public void setStreetName(String streetName) {
26        this.streetName = streetName;
27    }
28
29    public void setStreetType(String streetType) {
30        this.streetType = streetType;
31    }
32
33    public void setCity(String city) {
34        this.city = city;
35    }
36
37    public void setProvince(String province) {
38        this.province = province;
39
40
41    public void setPostalCode(String postalCode) {
42        this.postalCode = postalCode;
43    }
44
45
46
47    public String getApartmentNumber() {
48        return apartmentNumber;
49    }
50
51    public String getStreetNumber() {
52        return streetNumber;
53    }
54
55    public String getStreetName() {
56        return streetName;
57    }
58
59    public String getStreetType() {
60        return streetType;
61    }
62
63    public String getCity() {
64        return city;
65    }
66
67    public String getProvince() {
68        return province;
69    }
70
71    public String getPostalCode() {
72        return postalCode;
73    }
74
75 }
76
```

Example (Step 4)

In next implementation, we introduce the `CanadianAddressBuilder` class which separates argument collection and creation of instance of `CanadianAddress`

Problem: Want to prevent the ability to directly create new instances using “`new CanadianAddress()`”

Example (Step 4)

```
1 package csc301.builderExample;
2
3 public class CanadianAddressBuilder {
4
5     private String apartmentNumber;
6     private String streetNumber;
7     private String streetName;
8     private String streetType;
9     private String city;
10    private String province;
11    private String postalCode;
12
13
14
15    public CanadianAddressBuilder apartmentNumber(String apartmentNumber)
16    {
17        this.apartmentNumber = apartmentNumber;
18        return this;
19    }
20
21    public CanadianAddressBuilder streetNumber(String streetNumber){
22        this.streetNumber = streetNumber;
23        return this;
24    }
25
26    public CanadianAddressBuilder streetName(String streetName){
27        this.streetName = streetName;
28        return this;
29    }
30
31    public CanadianAddressBuilder streetType(String streetType){
32        this.streetType = streetType;
33        return this;
34    }
35
36    public CanadianAddressBuilder city(String city){
37        this.city = city;
38        return this;
39    }
40
41    public CanadianAddressBuilder province(String province){
42        this.province = province;
43        return this;
44    }
45
46    public CanadianAddressBuilder postalCode(String postalCode){
47        this.postalCode = postalCode;
48        return this;
49    }
50
51    public CanadianAddress build(){
52        // At this point, we can validate the arguments
53        CanadianAddress a = new CanadianAddress();
54        a.setApartmentNumber(apartmentNumber);
55        a.setStreetNumber(streetNumber);
56        a.setStreetName(streetName);
57        a.setStreetType(streetType);
58        a.setCity(city);
59        a.setProvince(province);
60        a.setPostalCode(postalCode);
61        return a;
62    }
63
64 }
```


Example (Step 4 Cont.)

```
1 package csc301.builderExample;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         CanadianAddress a = new CanadianAddressBuilder().apartmentNumber
8             ("1A")
9             .streetNumber("123-2").streetName("Main")
10            .streetType("Ave").city("Toronto").province("ON")
11            .postalCode("M2K 7R1").build();
12
13        // Do something with our address instance ...
14        System.out.println(a);
15    }
16
17 }
18
```

Example (Step 5)

- Builder class inside CanadianAddress class
- Changed constructor of CanadianAddress to be private and takes single argument of type CanadianAddress.Builder
- Cannot create new instances of CanadianAddress since constructor is private
- Can validate arguments inside build method

Example (Step 5)

```
1 package csc301.builderExample;
2
3
4 public class CanadianAddress {
5
6     public static class Builder {
7
8         private String apartmentNumber;
9         private String streetNumber;
10        private String streetName;
11        private String streetType;
12        private String city;
13        private String province;
14        private String postalCode;
15
16
17        public Builder apartmentNumber(String apartmentNumber){
18            this.apartmentNumber = apartmentNumber;
19            return this;
20        }
21
22        public Builder streetNumber(String streetNumber){
23            this.streetNumber = streetNumber;
24            return this;
25        }
26
27        public Builder streetName(String streetName){
28            this.streetName = streetName;
29            return this;
30        }
31
32        public Builder streetType(String streetType){
33            this.streetType = streetType;
34            return this;
35        }
36    }
37
38    public Builder city(String city){
39        this.city = city;
40        return this;
41    }
42
43    public Builder province(String province){
44        this.province = province;
45        return this;
46    }
47
48    public Builder postalCode(String postalCode){
49        this.postalCode = postalCode;
50        return this;
51    }
52
53    public CanadianAddress build(){
54        // We can validate the arguments, before creating a
55        // CanadianAddress instance.
56        return new CanadianAddress(this);
57    }
58
59
60
61
62
63
64
65    private String apartmentNumber;
66    private String streetNumber;
67    private String streetName;
68    private String streetType;
69    private String city;
70    private String province;
71    private String postalCode;
72
73
74
```

Example (Step 5 Cont)

```
74  
75  
76     private CanadianAddress(Builder builder) {  
77         this.apartmentNumber = builder.apartmentNumber;  
78         this.streetNumber = builder.streetNumber;  
79         this.streetName = builder.streetName;  
80         this.streetType = builder.streetType;  
81         this.city = builder.city;  
82         this.province = builder.province;  
83         this.postalCode = builder.postalCode;  
84     }  
85  
86  
87  
88     public String getApartmentNumber() {  
89         return apartmentNumber;  
90     }  
91  
92     public String getStreetNumber() {  
93         return streetNumber;  
94     }  
95  
96     public String getStreetName() {  
97         return streetName;  
98     }  
99  
100    public String getStreetType() {  
101        return streetType;  
102    }  
103  
104    public String getCity() {  
105        return city;  
106    }  
107  
108    public String getProvince() {  
109        return province;  
110    }  
111  
112    public String getPostalCode() {  
113        return postalCode;  
114    }  
115 }
```

Example (Step 5 Cont)

```
1 package csc301.builderExample;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         CanadianAddress a = new CanadianAddress.Builder().
8             apartmentNumber("1A")
9             .streetNumber("123-2").streetName("Main")
10            .streetType("Ave").city("Toronto").province("ON")
11            .postalCode("M2K 7R1").build();
12
13        // Do something with our address instance ...
14        System.out.println(a);
15    }
16
17 }
18
```

Example (Complete, Builder Class)

```
6
7 public static class Builder {
8
9     private String apartmentNumber;
10    private String streetNumber;
11    private String streetName;
12    private String streetType;
13    private String city;
14    private String province;
15    private String postalCode;
16
17
18    public Builder apartmentNumber(String apartmentNumber){
19        this.apartmentNumber = apartmentNumber;
20        return this;
21    }
22
23    public Builder streetNumber(String streetNumber){
24        this.streetNumber = streetNumber;
25        return this;
26    }
27
28    public Builder streetName(String streetName){
29        this.streetName = streetName;
30        return this;
31    }
32
33    public Builder streetType(String streetType){
34        this.streetType = streetType;
35        return this;
36    }
37
38    public Builder city(String city){
39        this.city = city;
40        return this;
41    }
42
43    public Builder province(String province){
44        this.province = province;
45        return this;
46    }
47
48    public Builder postalCode(String postalCode){
49        this.postalCode = postalCode;
50        return this;
51    }
52
53
54    public CanadianAddress build(){
55        // We can validate the arguments, before creating a CanadianAddress instance.
56        return new CanadianAddress(this);
57    }
58
59 }
60
```

Example (Complete, Canadian Address Class)

```
1 package csc301.builderExample;
2
3
4 public class CanadianAddress {
5
6     public static class Builder {
7         ...
8     }
9
10    private String apartmentNumber;
11    private String streetNumber;
12    private String streetName;
13    private String streetType;
14    private String city;
15    private String province;
16    private String postalCode;
17
18    private CanadianAddress(Builder builder) {
19        this.apartmentNumber = builder.apartmentNumber;
20        this.streetNumber = builder.streetNumber;
21        this.streetName = builder.streetName;
22        this.streetType = builder.streetType;
23        this.city = builder.city;
24        this.province = builder.province;
25        this.postalCode = builder.postalCode;
26    }
27
28
29
30
31    public String getApartmentNumber() {
32        return apartmentNumber;
33    }
34
35    public String getStreetNumber() {
36        return streetNumber;
37    }
38
39    public String getStreetName() {
40        return streetName;
41    }
42
43    public String getStreetType() {
44        return streetType;
45    }
46
47    public String getCity() {
48        return city;
49    }
50
51    public String getProvince() {
52        return province;
53    }
54
55    public String getPostalCode() {
56        return postalCode;
57    }
58
59 }
60
```

Conclusion

Final notes on Builder Design Pattern

Conclusion

The builder design pattern builds complex objects step by step and returns a product at the end.

Use when you have complex object with common input and many possible representations

THE END