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

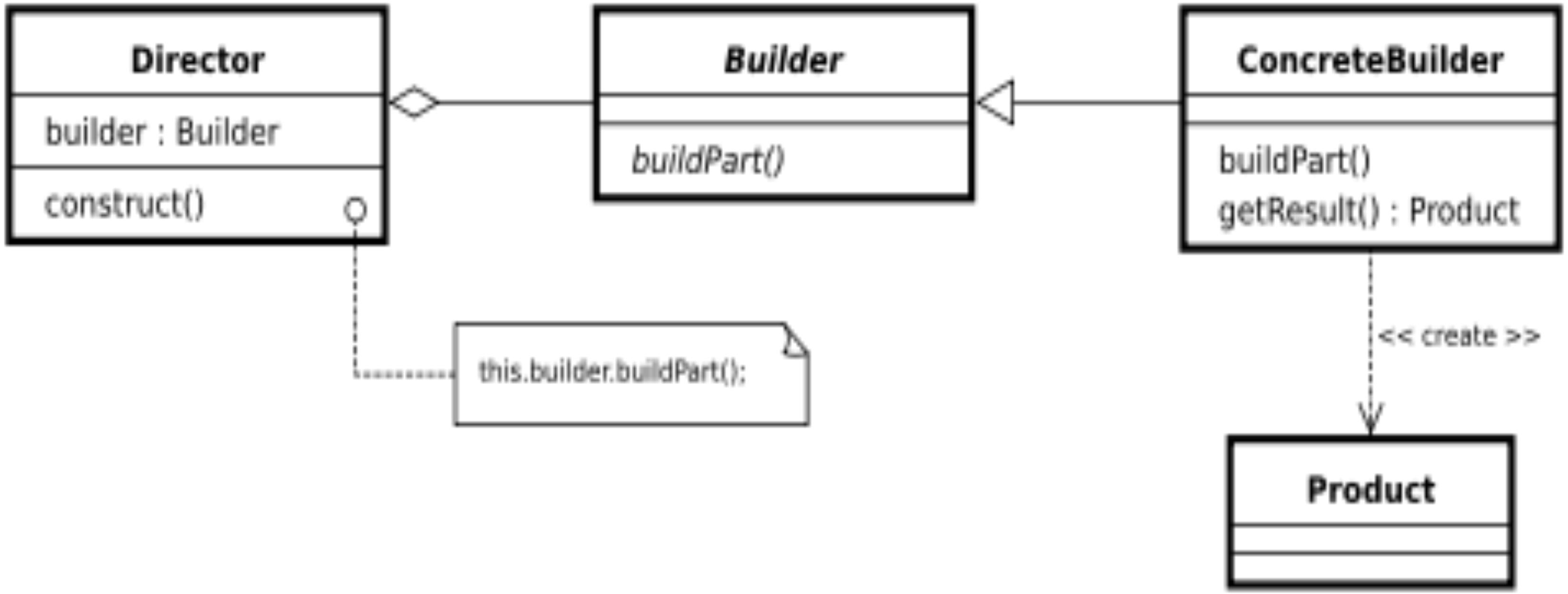
Parse a complex representation and create one of several target products.

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


Director
builder : Builder
construct()
Builder
buildPart()
ConcreteBuilder
buildPart()
getResult() : Product
this.builder.buildPart();
<< create >>
Product

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

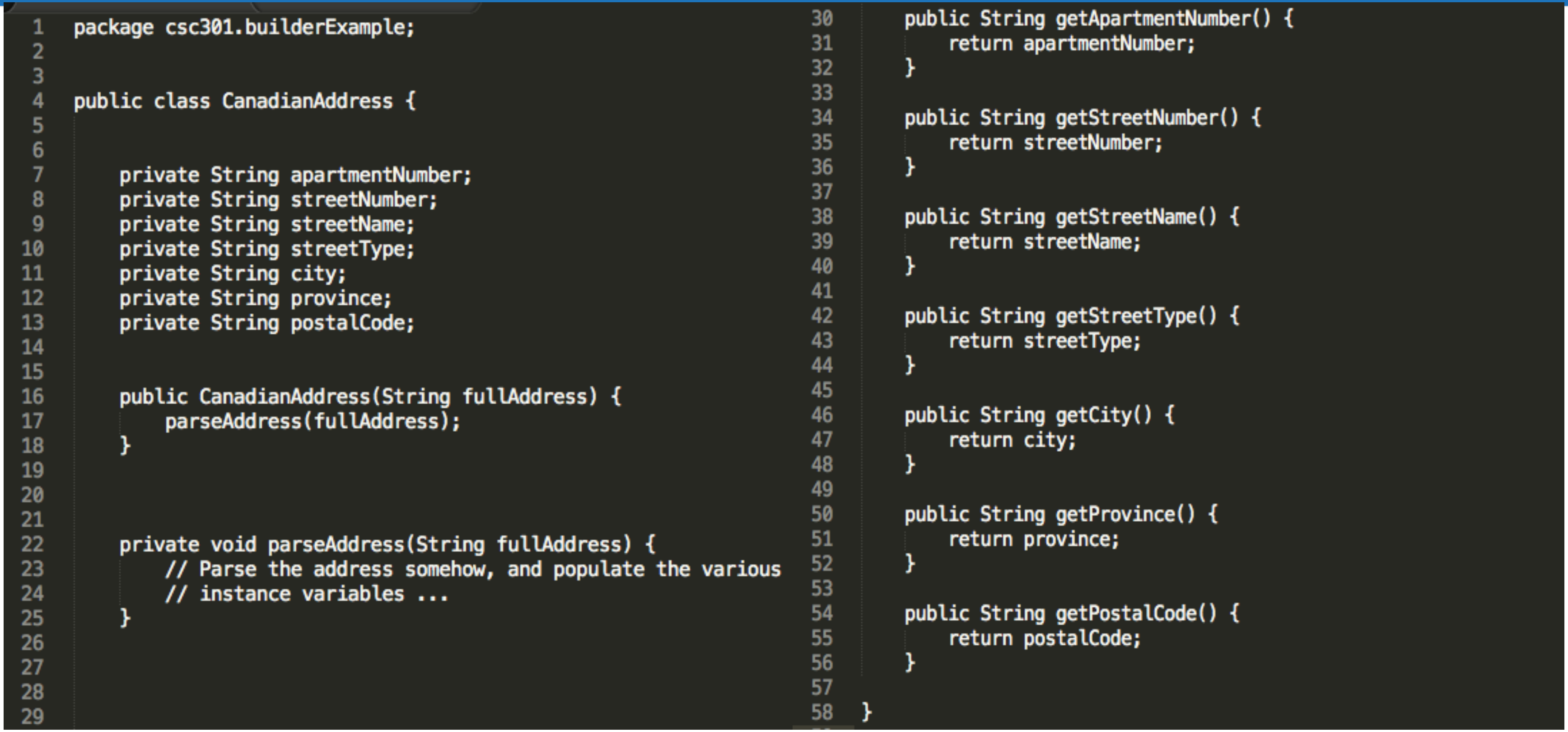
Example (Step 1)

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)



```java
package csc301.builderExample;


public class CanadianAddress {


    private String apartmentNumber;
    private String streetNumber;
    private String streetName;
    private String streetType;
    private String city;
    private String province;
    private String postalCode;


    public CanadianAddress(String fullAddress) {
        parseAddress(fullAddress);
    }



    private void parseAddress(String fullAddress) {
        // Parse the address somehow, and populate the various
        // instance variables ...
    }




    public String getApartmentNumber() {
        return apartmentNumber;
    }

    public String getStreetNumber() {
        return streetNumber;
    }

    public String getStreetName() {
        return streetName;
    }

    public String getStreetType() {
        return streetType;
    }

    public String getCity() {
        return city;
    }

    public String getProvince() {
        return province;
    }

    public String getPostalCode() {
        return postalCode;
    }

}
```

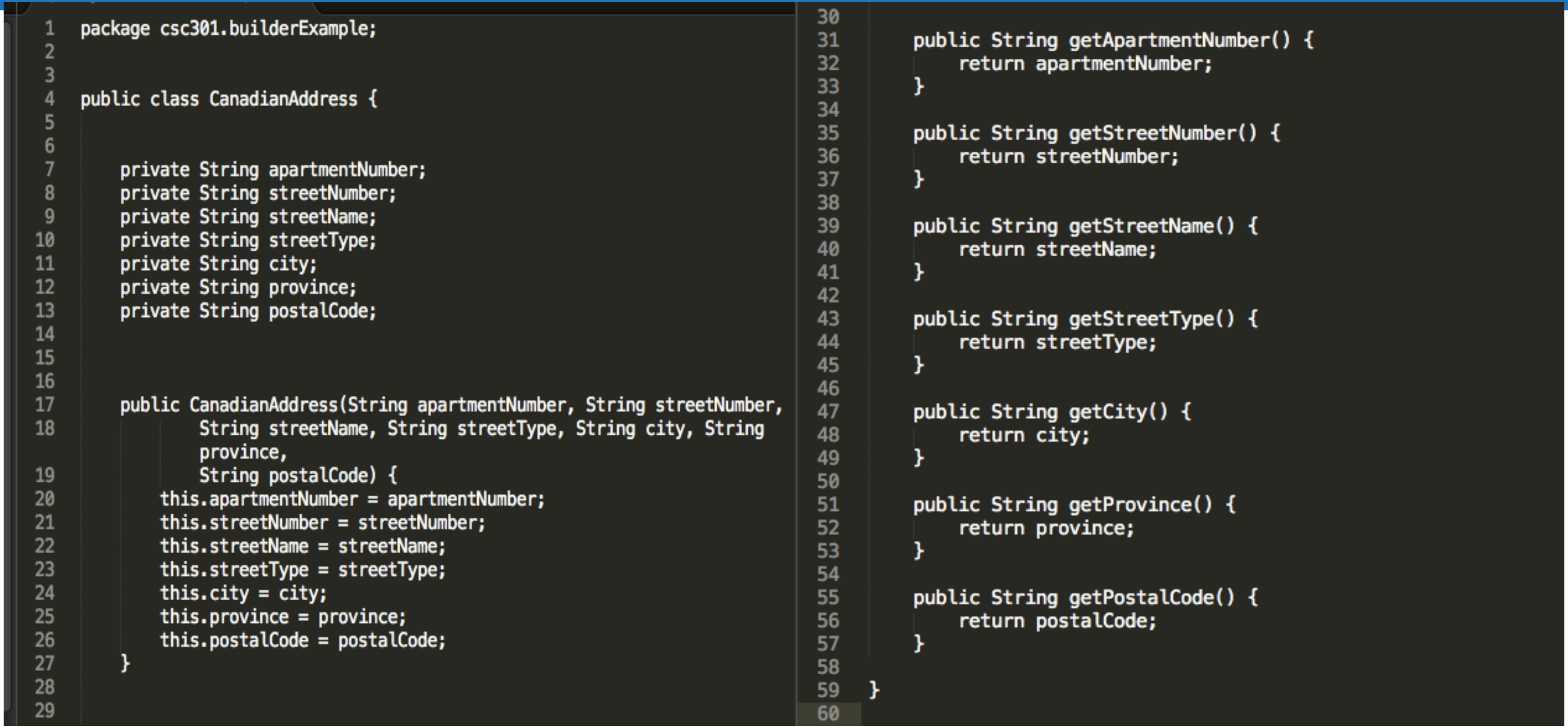
Example (Step 2)

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)



```java
package csc301.builderExample;


public class CanadianAddress {


    private String apartmentNumber;
    private String streetNumber;
    private String streetName;
    private String streetType;
    private String city;
    private String province;
    private String postalCode;



    public CanadianAddress(String apartmentNumber, String streetNumber,
            String streetName, String streetType, String city, String
            province,
            String postalCode) {
        this.apartmentNumber = apartmentNumber;
        this.streetNumber = streetNumber;
        this.streetName = streetName;
        this.streetType = streetType;
        this.city = city;
        this.province = province;
        this.postalCode = postalCode;
    }



    public String getApartmentNumber() {
        return apartmentNumber;
    }

    public String getStreetNumber() {
        return streetNumber;
    }

    public String getStreetName() {
        return streetName;
    }

    public String getStreetType() {
        return streetType;
    }

    public String getCity() {
        return city;
    }

    public String getProvince() {
        return province;
    }

    public String getPostalCode() {
        return postalCode;
    }

}
```

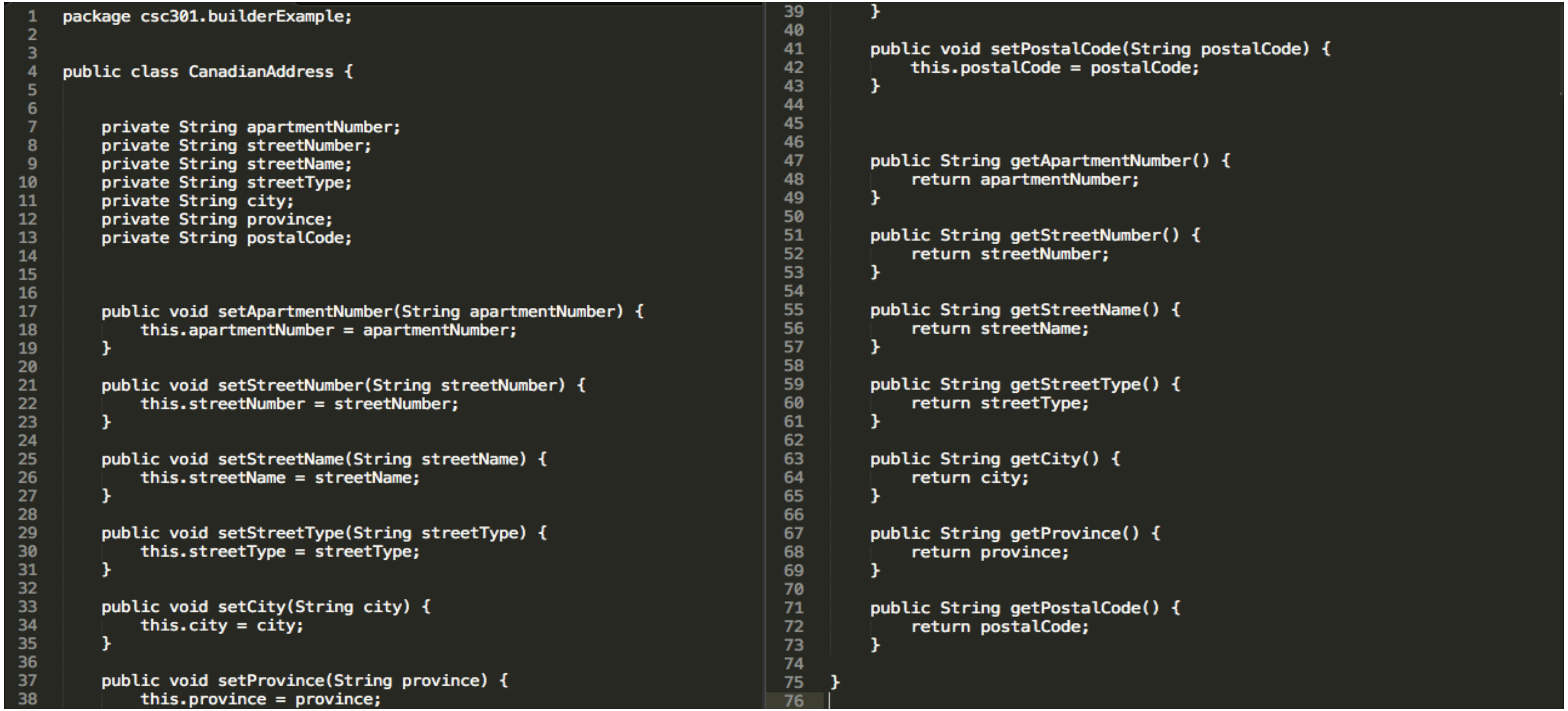
Example (Step 3)

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)



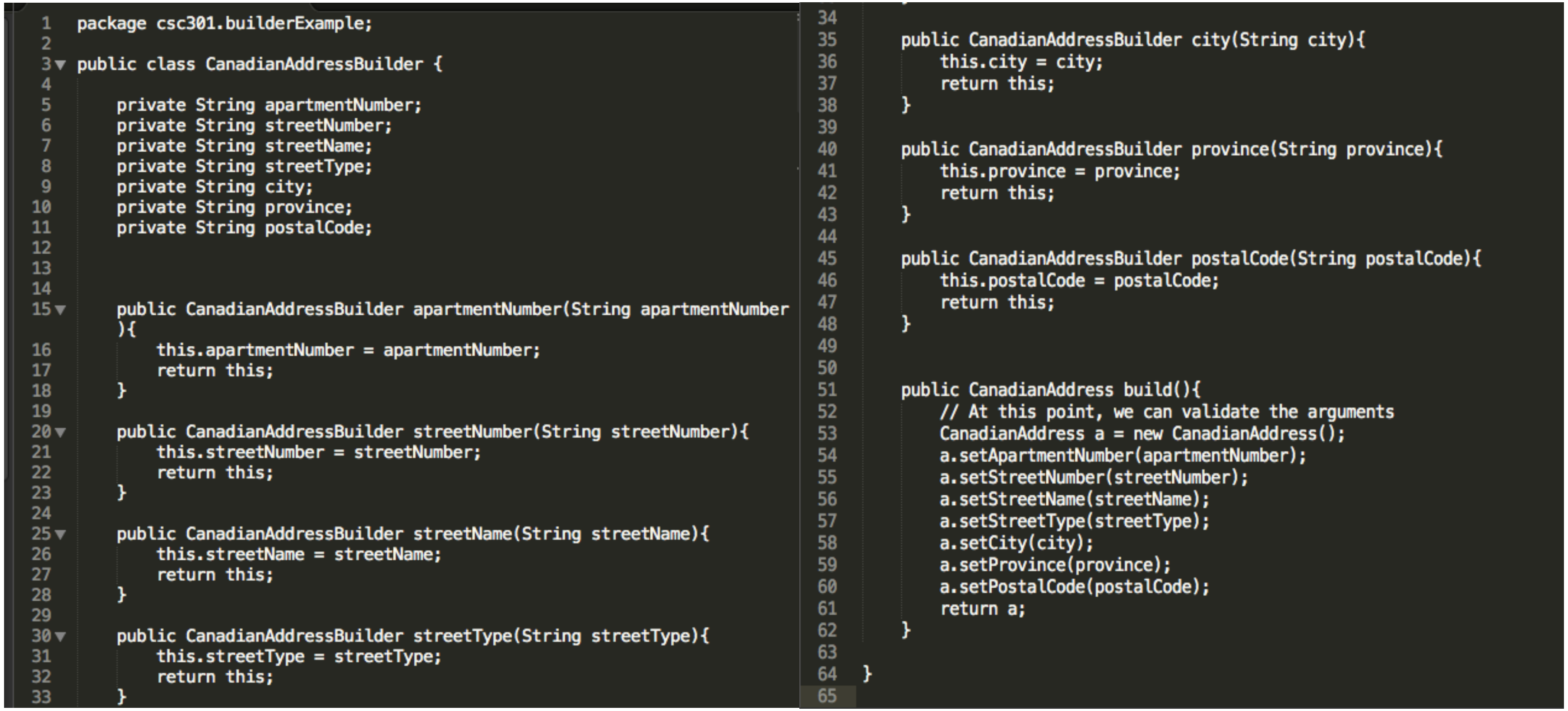
```java
package csc301.builderExample;


public class CanadianAddress {


    private String apartmentNumber;
    private String streetNumber;
    private String streetName;
    private String streetType;
    private String city;
    private String province;
    private String postalCode;



    public void setApartmentNumber(String apartmentNumber) {
        this.apartmentNumber = apartmentNumber;
    }

    public void setStreetNumber(String streetNumber) {
        this.streetNumber = streetNumber;
    }

    public void setStreetName(String streetName) {
        this.streetName = streetName;
    }

    public void setStreetType(String streetType) {
        this.streetType = streetType;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public void setProvince(String province) {
        this.province = province;
    }

    public void setPostalCode(String postalCode) {
        this.postalCode = postalCode;
    }



    public String getApartmentNumber() {
        return apartmentNumber;
    }

    public String getStreetNumber() {
        return streetNumber;
    }

    public String getStreetName() {
        return streetName;
    }

    public String getStreetType() {
        return streetType;
    }

    public String getCity() {
        return city;
    }

    public String getProvince() {
        return province;
    }

    public String getPostalCode() {
        return postalCode;
    }

}
```

Example (Step 4)

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

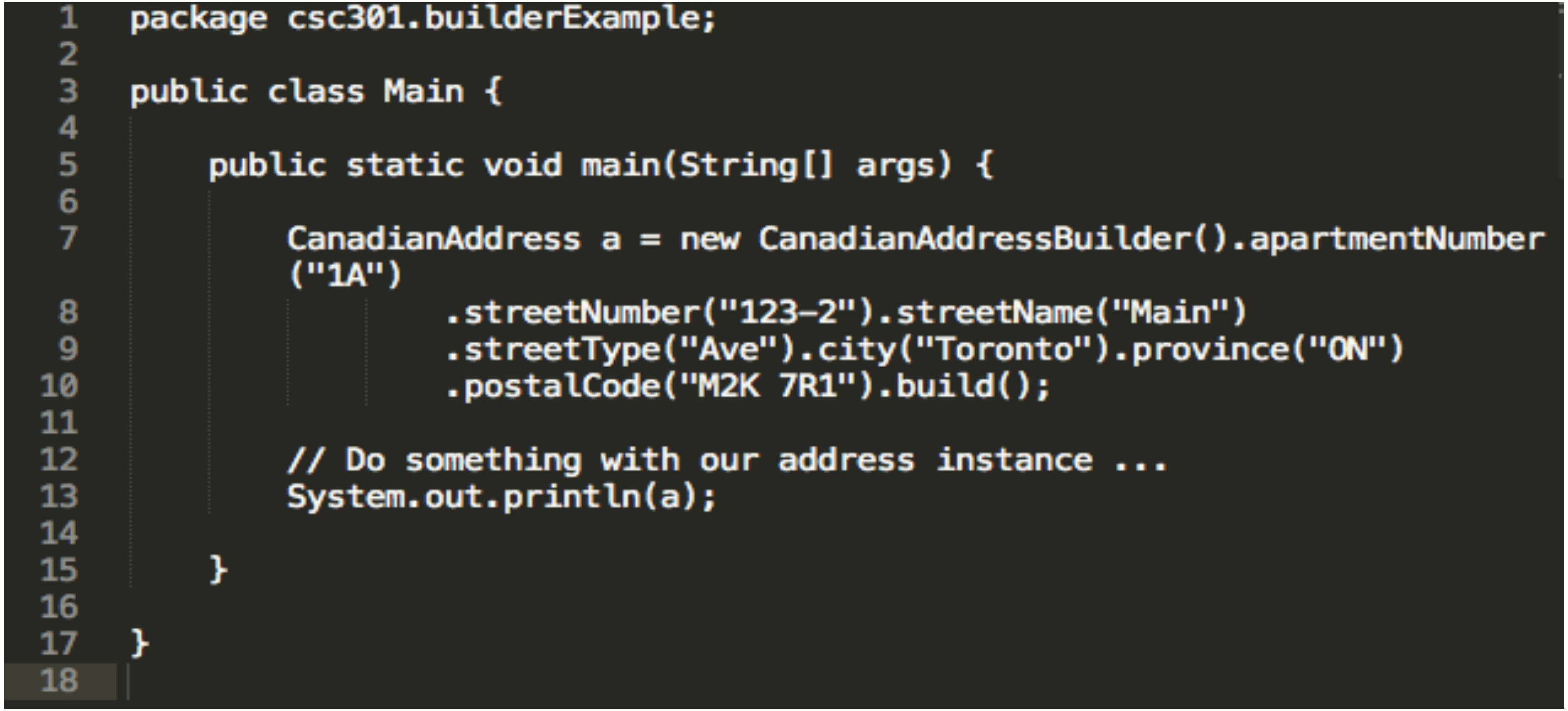
Problem: Want to prevent the ability to directly create new instances using “new CanadianAddress()”

Example (Step 4)



```java
package csc301.builderExample;

public class CanadianAddressBuilder {

    private String apartmentNumber;
    private String streetNumber;
    private String streetName;
    private String streetType;
    private String city;
    private String province;
    private String postalCode;



    public CanadianAddressBuilder apartmentNumber(String apartmentNumber){
        this.apartmentNumber = apartmentNumber;
        return this;
    }

    public CanadianAddressBuilder streetNumber(String streetNumber){
        this.streetNumber = streetNumber;
        return this;
    }

    public CanadianAddressBuilder streetName(String streetName){
        this.streetName = streetName;
        return this;
    }

    public CanadianAddressBuilder streetType(String streetType){
        this.streetType = streetType;
        return this;
    }

    public CanadianAddressBuilder city(String city){
        this.city = city;
        return this;
    }

    public CanadianAddressBuilder province(String province){
        this.province = province;
        return this;
    }

    public CanadianAddressBuilder postalCode(String postalCode){
        this.postalCode = postalCode;
        return this;
    }


    public CanadianAddress build(){
        // At this point, we can validate the arguments
        CanadianAddress a = new CanadianAddress();
        a.setApartmentNumber(apartmentNumber);
        a.setStreetNumber(streetNumber);
        a.setStreetName(streetName);
        a.setStreetType(streetType);
        a.setCity(city);
        a.setProvince(province);
        a.setPostalCode(postalCode);
        return a;
    }

}
```

Example (Step 4 Cont.)

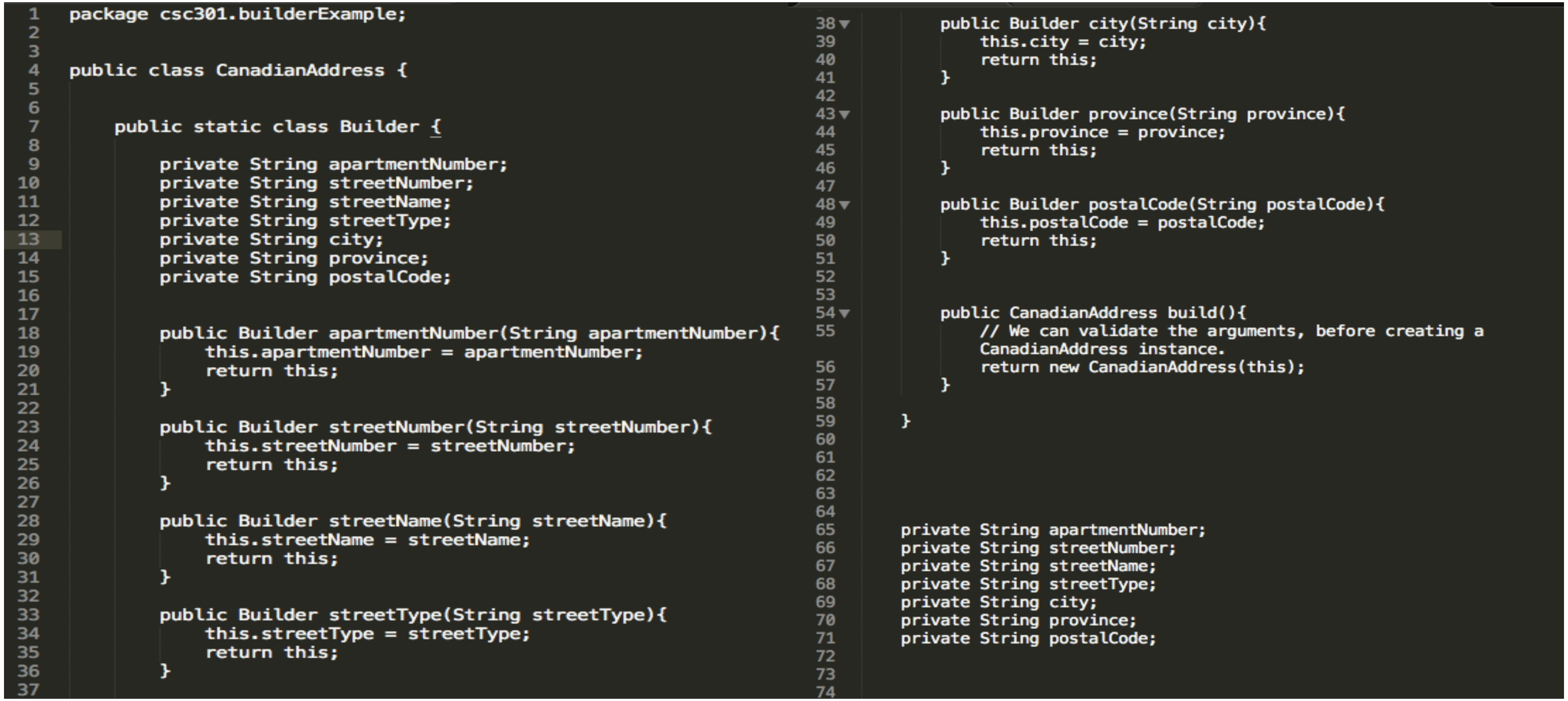


```java
package csc301.builderExample;

public class Main {

    public static void main(String[] args) {

        CanadianAddress a = new CanadianAddressBuilder().apartmentNumber
        ("1A")
                .streetNumber("123-2").streetName("Main")
                .streetType("Ave").city("Toronto").province("ON")
                .postalCode("M2K 7R1").build();

        // Do something with our address instance ...
        System.out.println(a);

    }

}
```

Example (Step 5)

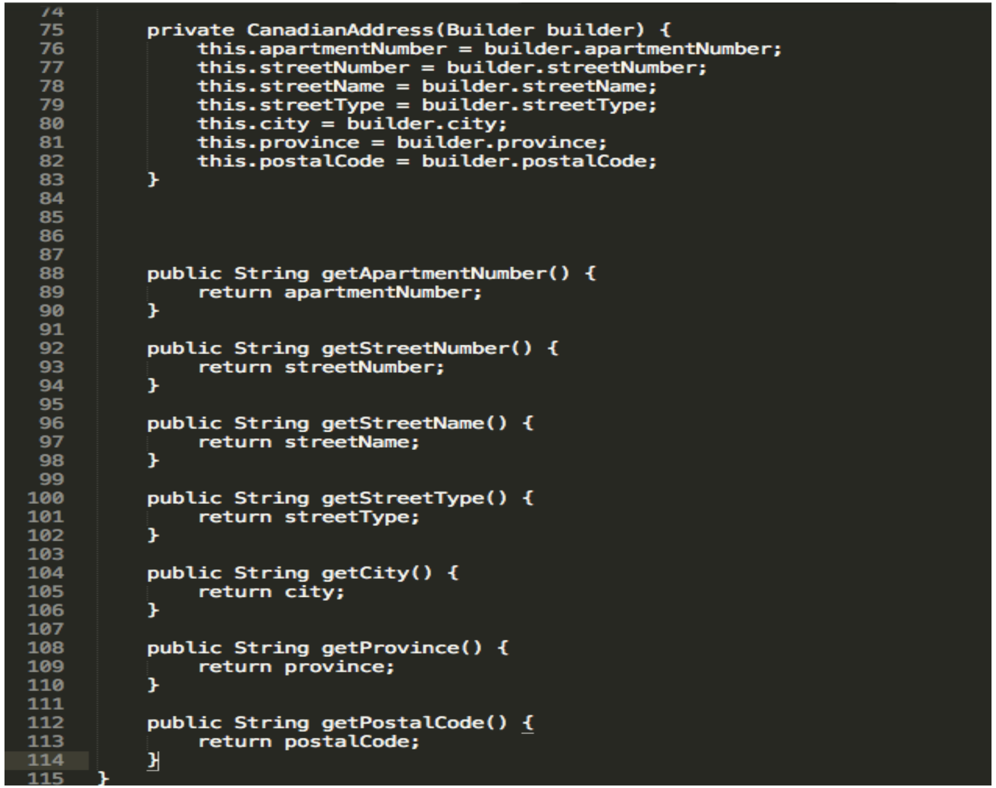
- Builder class inside CanadianAddress class
- Changed constructor of CanadianAddress to be private and takes single argument of type CanadianAddress.Builder
- Cannot create new instances of CanadianAddress since constructor is private
- Can validate arguments inside build method

Example (Step 5)



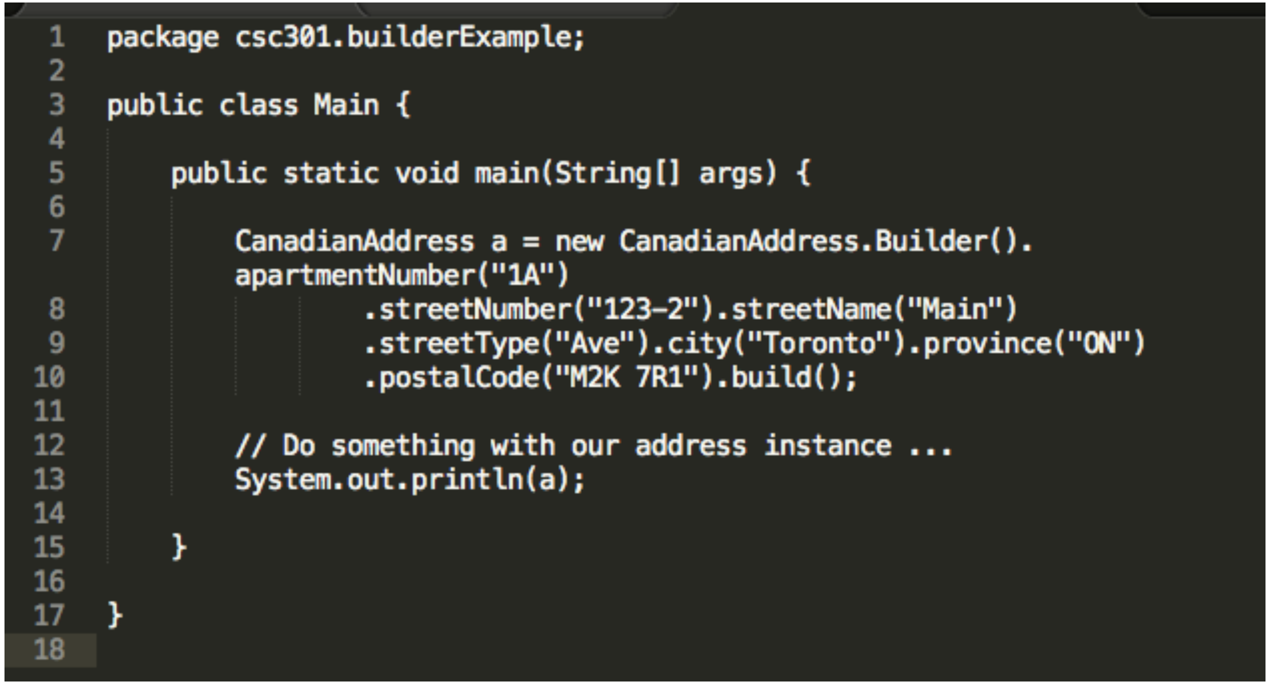
```
package csc301.builderExample;


public class CanadianAddress {


    public static class Builder {

        private String apartmentNumber;
        private String streetNumber;
        private String streetName;
        private String streetType;
        private String city;
        private String province;
        private String postalCode;


        public Builder apartmentNumber(String apartmentNumber){
            this.apartmentNumber = apartmentNumber;
            return this;
        }

        public Builder streetNumber(String streetNumber){
            this.streetNumber = streetNumber;
            return this;
        }

        public Builder streetName(String streetName){
            this.streetName = streetName;
            return this;
        }

        public Builder streetType(String streetType){
            this.streetType = streetType;
            return this;
        }

        public Builder city(String city){
            this.city = city;
            return this;
        }

        public Builder province(String province){
            this.province = province;
            return this;
        }

        public Builder postalCode(String postalCode){
            this.postalCode = postalCode;
            return this;
        }


        public CanadianAddress build(){
            // We can validate the arguments, before creating a
            CanadianAddress instance.
            return new CanadianAddress(this);
        }

    }






    private String apartmentNumber;
    private String streetNumber;
    private String streetName;
    private String streetType;
    private String city;
    private String province;
    private String postalCode;
```

Example (Step 5 Cont)



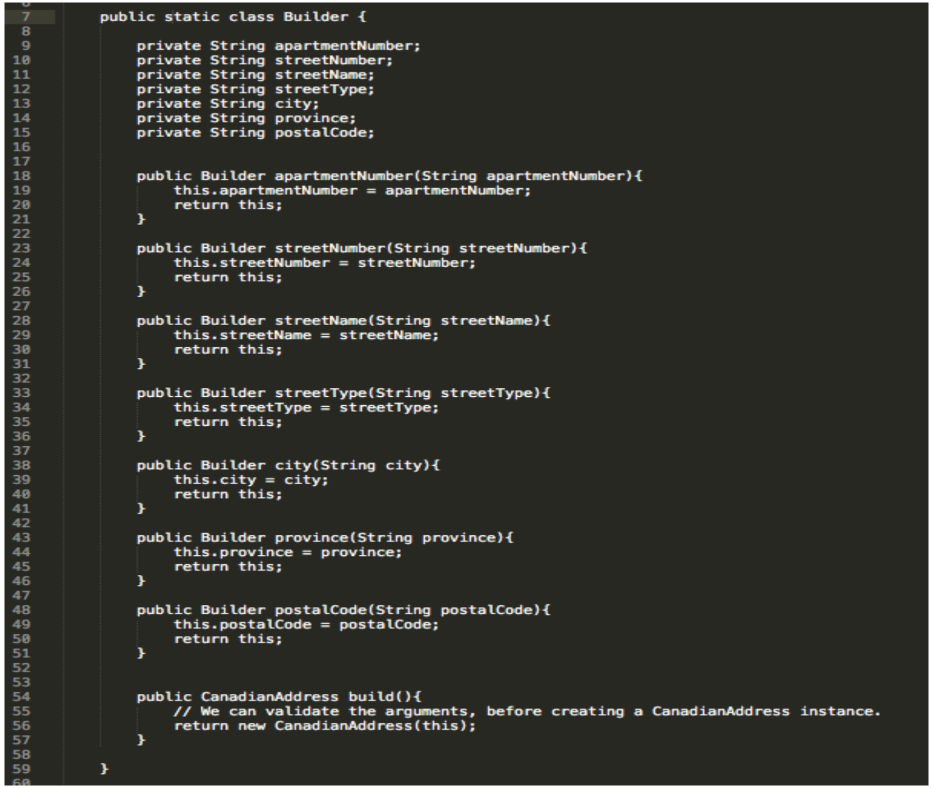
```
    private CanadianAddress(Builder builder) {
        this.apartmentNumber = builder.apartmentNumber;
        this.streetNumber = builder.streetNumber;
        this.streetName = builder.streetName;
        this.streetType = builder.streetType;
        this.city = builder.city;
        this.province = builder.province;
        this.postalCode = builder.postalCode;
    }



    public String getApartmentNumber() {
        return apartmentNumber;
    }

    public String getStreetNumber() {
        return streetNumber;
    }

    public String getStreetName() {
        return streetName;
    }

    public String getStreetType() {
        return streetType;
    }

    public String getCity() {
        return city;
    }

    public String getProvince() {
        return province;
    }

    public String getPostalCode() {
        return postalCode;
    }
}
```

Example (Step 5 Cont)



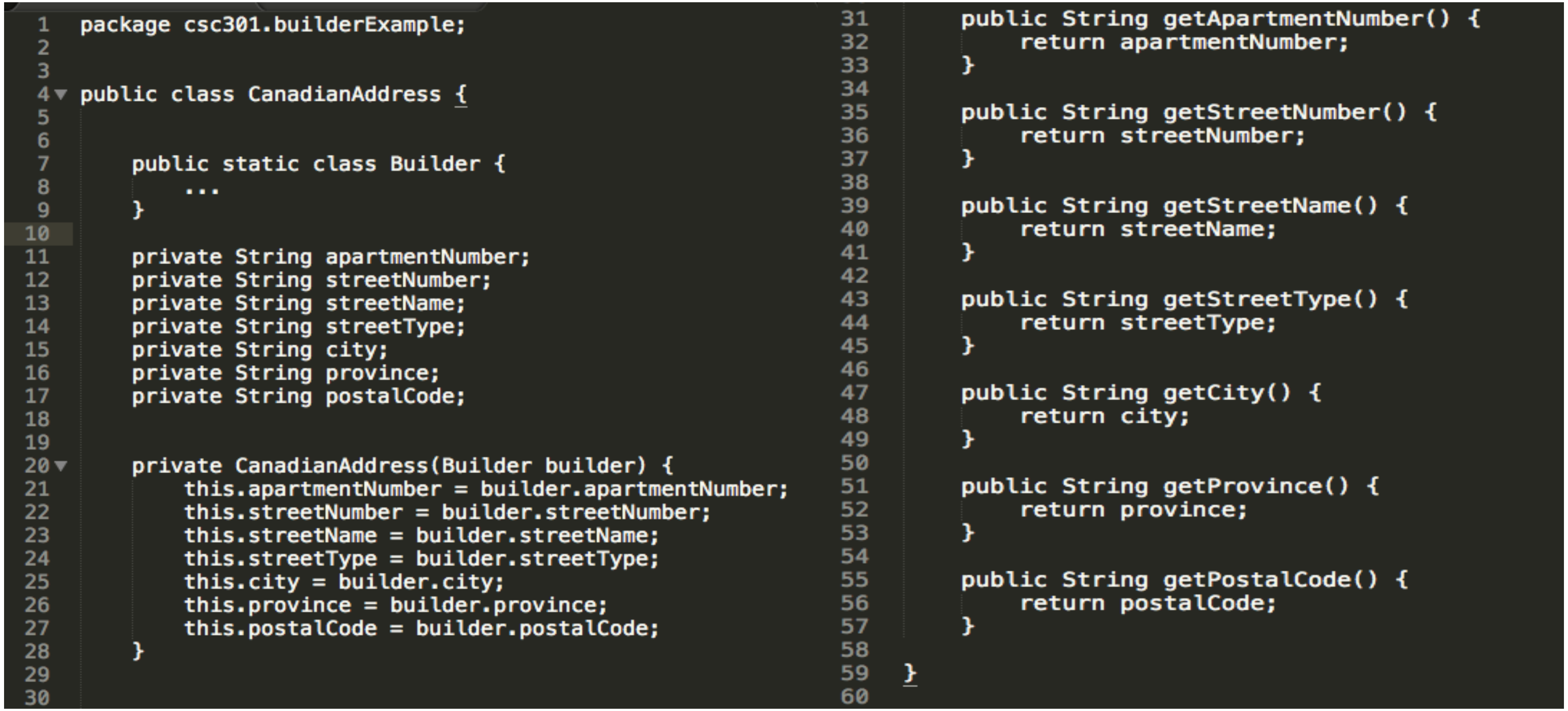
```java
package csc301.builderExample;

public class Main {

    public static void main(String[] args) {

        CanadianAddress a = new CanadianAddress.Builder().
        apartmentNumber("1A")
                .streetNumber("123-2").streetName("Main")
                .streetType("Ave").city("Toronto").province("ON")
                .postalCode("M2K 7R1").build();

        // Do something with our address instance ...
        System.out.println(a);

    }

}
```

Example (Complete, Builder Class)



```java
    public static class Builder {

        private String apartmentNumber;
        private String streetNumber;
        private String streetName;
        private String streetType;
        private String city;
        private String province;
        private String postalCode;


        public Builder apartmentNumber(String apartmentNumber){
            this.apartmentNumber = apartmentNumber;
            return this;
        }

        public Builder streetNumber(String streetNumber){
            this.streetNumber = streetNumber;
            return this;
        }

        public Builder streetName(String streetName){
            this.streetName = streetName;
            return this;
        }

        public Builder streetType(String streetType){
            this.streetType = streetType;
            return this;
        }

        public Builder city(String city){
            this.city = city;
            return this;
        }

        public Builder province(String province){
            this.province = province;
            return this;
        }

        public Builder postalCode(String postalCode){
            this.postalCode = postalCode;
            return this;
        }


        public CanadianAddress build(){
            // We can validate the arguments, before creating a CanadianAddress instance.
            return new CanadianAddress(this);
        }

    }
```

Example (Complete, Canadian Address Class)



```
package csc301.builderExample;


public class CanadianAddress {


    public static class Builder {
        ...
    }

    private String apartmentNumber;
    private String streetNumber;
    private String streetName;
    private String streetType;
    private String city;
    private String province;
    private String postalCode;


    private CanadianAddress(Builder builder) {
        this.apartmentNumber = builder.apartmentNumber;
        this.streetNumber = builder.streetNumber;
        this.streetName = builder.streetName;
        this.streetType = builder.streetType;
        this.city = builder.city;
        this.province = builder.province;
        this.postalCode = builder.postalCode;
    }


    public String getApartmentNumber() {
        return apartmentNumber;
    }

    public String getStreetNumber() {
        return streetNumber;
    }

    public String getStreetName() {
        return streetName;
    }

    public String getStreetType() {
        return streetType;
    }

    public String getCity() {
        return city;
    }

    public String getProvince() {
        return province;
    }

    public String getPostalCode() {
        return postalCode;
    }

}
```

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