The Builder Pattern is a creational design pattern that allows you to construct complex objects step by step.

It separates the construction of a complex object from its representation,

allowing the same construction process to create different representations. *public class* BankAccount {  
 *private* String name;  
 *private* String accountNumber;  
 *private* String email;  
 *private boolean* newsletter;  
  
 *//The constructor that takes a builder from which it will create object  
 //the access to this is only provided to builder  
 private* BankAccount(BankAccountBuilder builder) {  
 *this*.name = builder.name;  
 *this*.accountNumber = builder.accountNumber;  
 *this*.email = builder.email;  
 *this*.newsletter = builder.newsletter;  
 }  
  
 *public static class* BankAccountBuilder {  
 *private* String name;  
 *private* String accountNumber;  
 *private* String email;  
 *private boolean* newsletter;  
  
 *//All Mandatory parameters goes with this constructor  
 public* BankAccountBuilder(String name, String accountNumber) {  
 *this*.name = name;  
 *this*.accountNumber = accountNumber;  
 }  
  
 *//setters for optional parameters which returns this same builder  
 //to support fluent design  
 public* BankAccountBuilder withEmail(String email) {  
 *this*.email = email;  
 *return this*;  
 }  
  
 *public* BankAccountBuilder wantNewsletter(*boolean* newsletter) {  
 *this*.newsletter = newsletter;  
 *return this*;  
 }  
  
 *//the actual build method that prepares and returns a BankAccount object  
 public* BankAccount build() {  
 *return new* BankAccount(*this*);  
 }  
 }  
  
 *//getters  
 public* String getName() {  
 *return* name;  
 }  
  
 *public* String getAccountNumber() {  
 *return* accountNumber;  
 }  
  
 *public* String getEmail() {  
 *return* email;  
 }  
  
 *public boolean* isNewsletter() {  
 *return* newsletter;  
 }  
}

*public class* BuilderPatternDriver {  
 *public static void* main(String[] args) {  
 BankAccount newAccount = *new* BankAccount  
 .BankAccountBuilder("Jon", "22738022275")  
 .withEmail("jon@example.com")  
 .wantNewsletter(*true*)  
 .build();  
  
 System.*out*.println("Name: " + newAccount.getName());  
 System.*out*.println("AccountNumber:" + newAccount.getAccountNumber());  
 System.*out*.println("Email: " + newAccount.getEmail());  
 System.*out*.println("Want News letter?: " + newAccount.isNewsletter());  
 }  
}

Name: Jon

AccountNumber:22738022275

Email: null

Want News letter?: false

@Data  
*class* Pizza {  
 *private* String size;  
 *private* String crustType;  
 *private boolean* cheese;  
 *private List*<String> toppings;  
  
 *private* Pizza(Builder builder) {  
 *this*.size = builder.size;  
 *this*.crustType = builder.crustType;  
 *this*.cheese = builder.cheese;  
 *this*.toppings = builder.toppings;  
 }  
  
 *// Getters  
 // You can add setters if needed  
  
 // Static inner Builder class  
 static class* Builder {  
 *private* String size;  
 *private* String crustType;  
 *private boolean* cheese;  
 *private List*<String> toppings = *new* ArrayList<>();  
  
 *public* Builder size(String size) {  
 *this*.size = size;  
 *return this*;  
 }  
  
 *public* Builder crustType(String crustType) {  
 *this*.crustType = crustType;  
 *return this*;  
 }  
  
 *public* Builder cheese(*boolean* cheese) {  
 *this*.cheese = cheese;  
 *return this*;  
 }  
  
 *public* Builder addTopping(String topping) {  
 *this*.toppings.add(topping);  
 *return this*;  
 }  
  
 *public* Pizza build() {  
 *return new* Pizza(*this*);  
 }  
 }  
}  
  
*public class* Builder2 {  
 *public static void* main(String[] args) {

// Pizza pizza1 = new Pizza(); // direct creation is not allowed  
 Pizza pizza = *new* Pizza.Builder()  
 .size("Large")  
 .crustType("Thin crust")  
 .cheese(*true*)  
 .addTopping("Pepperoni")  
 .addTopping("Mushrooms")  
 .build();  
  
 System.*out*.println("Pizza details:");  
 System.*out*.println("Size: " + pizza.getSize());  
 System.*out*.println("Crust Type: " + pizza.getCrustType());  
 System.*out*.println("Cheese: " + pizza.isCheese());  
 System.*out*.println("Toppings: " + pizza.getToppings());  
 }  
}

Pizza details:

Size: Large

Crust Type: Thin crust

Cheese: true

Toppings: [Pepperoni, Mushrooms]