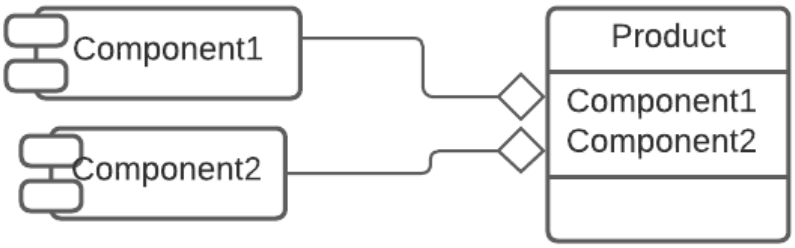
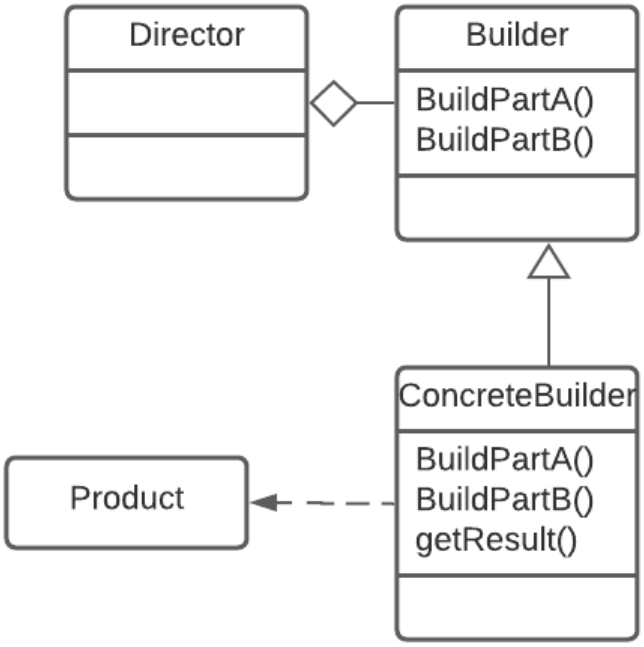
Builder Design Pattern

The Builder Design Pattern allows us to build complex objects by separating the construction of the objects from its representation.
-- Separates construction of complex objects
Unlike the Factory and Abstract Factory which build objects in one shot. Builder allows us to build complex objects one step at a time.
-- Same construction process for different representations
It allows us to use the same step or process to create different representation of the complex object.



This is particularly useful design when creating an object that is made up of other objects. Each component can be instantiated separately snd then put together before it's returned to the caller.
Not only does separating an object into individual components, helps avoid class explosion, it makes it easier to maintain.

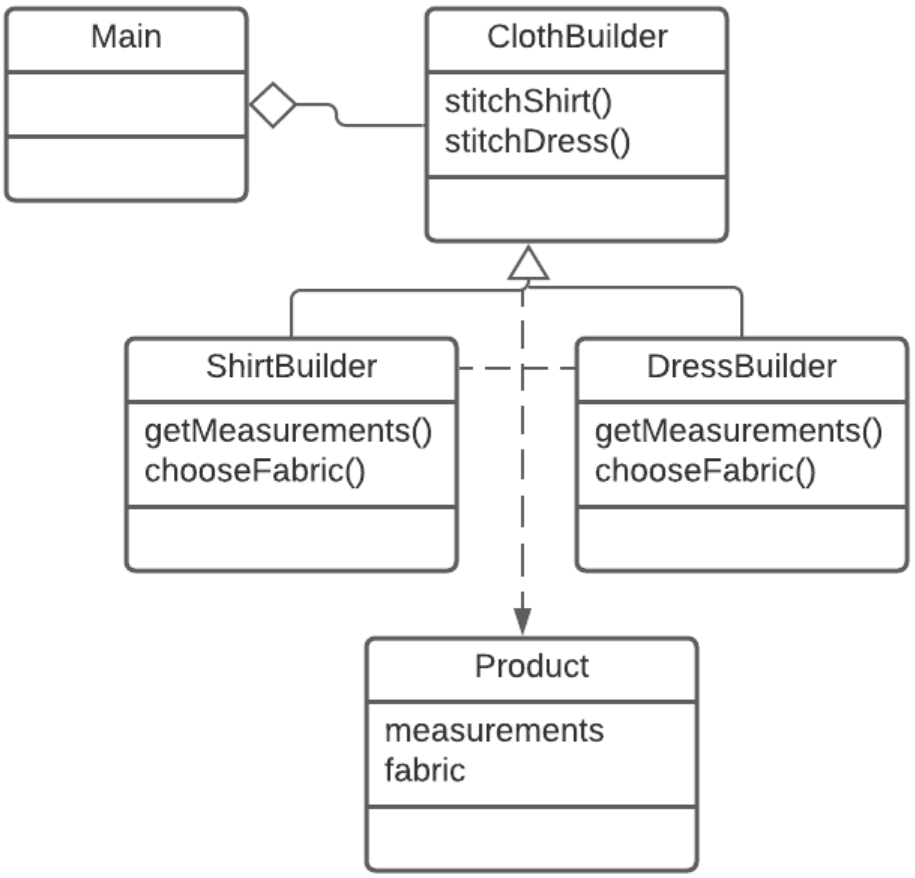


This is the Generic Class diagram for Builder pattern.

The “Product” represents the complex object that is being built one step at a time.
The “Builder” outlines the steps needed to create a product. It knows in general what the individual steps need to be followed to create that final project.
The “ConcreteBuilder” is where the implementation of the different components are specified.
The “Director” invokes the Builder to construct the desired product. The ConcreteBuilder returns the Product to the Director.

Example:

The Tailor stitches the cloths such as Pants, Shirts, Dresses, etc.




```
class Cloth {
protected:
        string _cloth;
public:
        string getCloth(){
                return _cloth;
        }
};
```

```
class Shirt : public Cloth{
public:
	Shirt(){
		cout<<endl<<”Stitch the shirt as per measurements with the given fabric”;
		_cloth = “Shirt Piece”;
	}
};

class Dress : public Cloth {
public:
	Dress(){
		cout<<endl<<”Stitch the dress as per measurements with the given fabric”;
		_cloth = “Dress Material”;
	}
};

public Product {
private:
	Cloth *cloth;
	string bag;
public:
	Product(string type){
		bag = type;
	}
	void setCloth(Cloth *c)
	{
		cloth = c;
	}

	string tryCloth(){
		bag = bag + " from " + cloth->getCloth();
		return bag;
	}
};

class ClothBuilder {
protected:
	Product* _product;
public:
	virtual void stitchCloth() {};
	Product* getCloth(){
		return _product;
	}
};

class ShirtBuilder : public ClothBuilder {
public:
	ShirtBuilder() {
```

```
                _product = new Product("Shirt");
        }
        void stitchCloth(){
                Shirt *shirt = new Shirt;
                _product->setCloth(shirt);
        }
};

class DressBuilder : public ClothBuilder {
public:
        DressBuilder() {
                _product = new Product("Dress");
        }
        void stitchCloth(){
                Dress *dress = new Dress;
                _product->setCloth(dress);
        }
};

int main()
{
        ClothBuilder *stitch = new ClothBuilder;
        Product *prod;
        int choice = 0;
        cout<<"Select a clothing :: "<<endl;
        cout<<"1. Shirt "<<endl;
        cout<<"2. Dress "<<endl;
        cout<<"Selection :  ";
        cin>>choice;
        cout<<endl;
        switch(choice){
                case 1:
                        stitch = new ShirtBuilder;
                        break;
                case 2:
                        stitch = new DressBuilder;
                        break;
                default:
                        cout<<"Soon we are adding new clothings as per your
selection!!"<<endl;
                        stitch = nullptr;
                        break;
        }

        if(stitch){
                cout<<"Stitching selected cloth"<<endl;
                stitch->stitchCloth();
                prod = stitch->getProduct();
```

```
            cout<<prod->tryCloth()<<endl;
        }

        return 0;
}
```