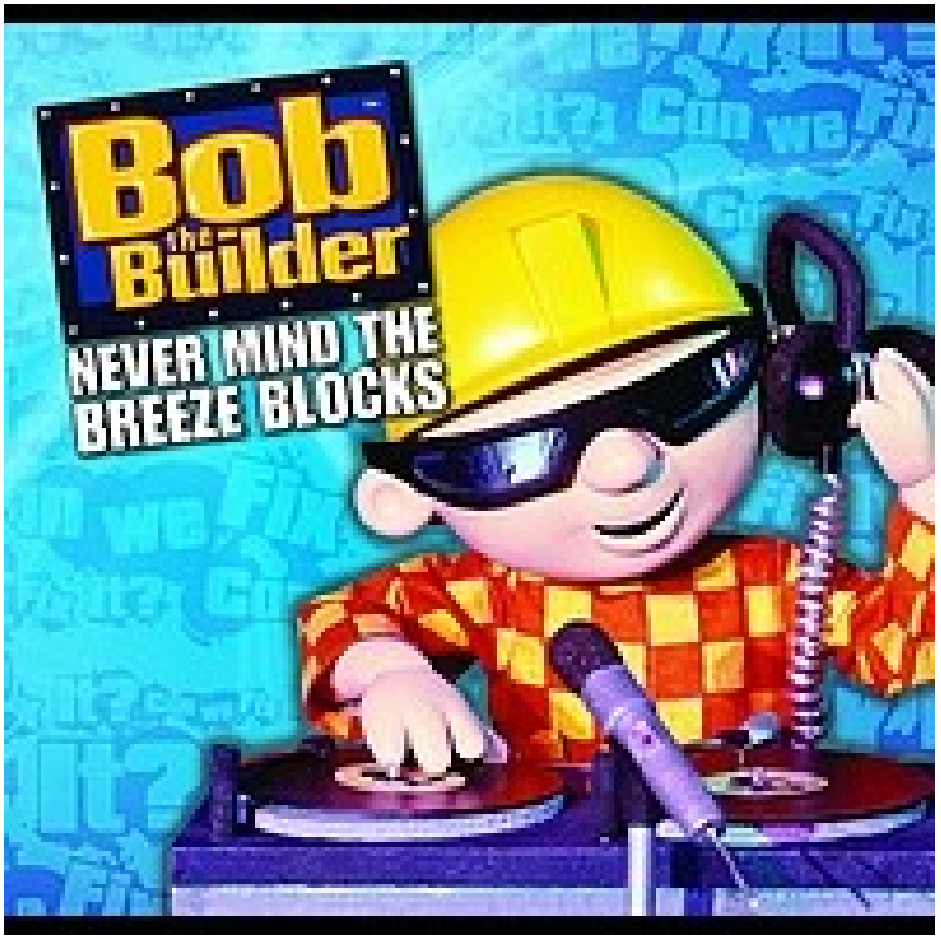
Builder

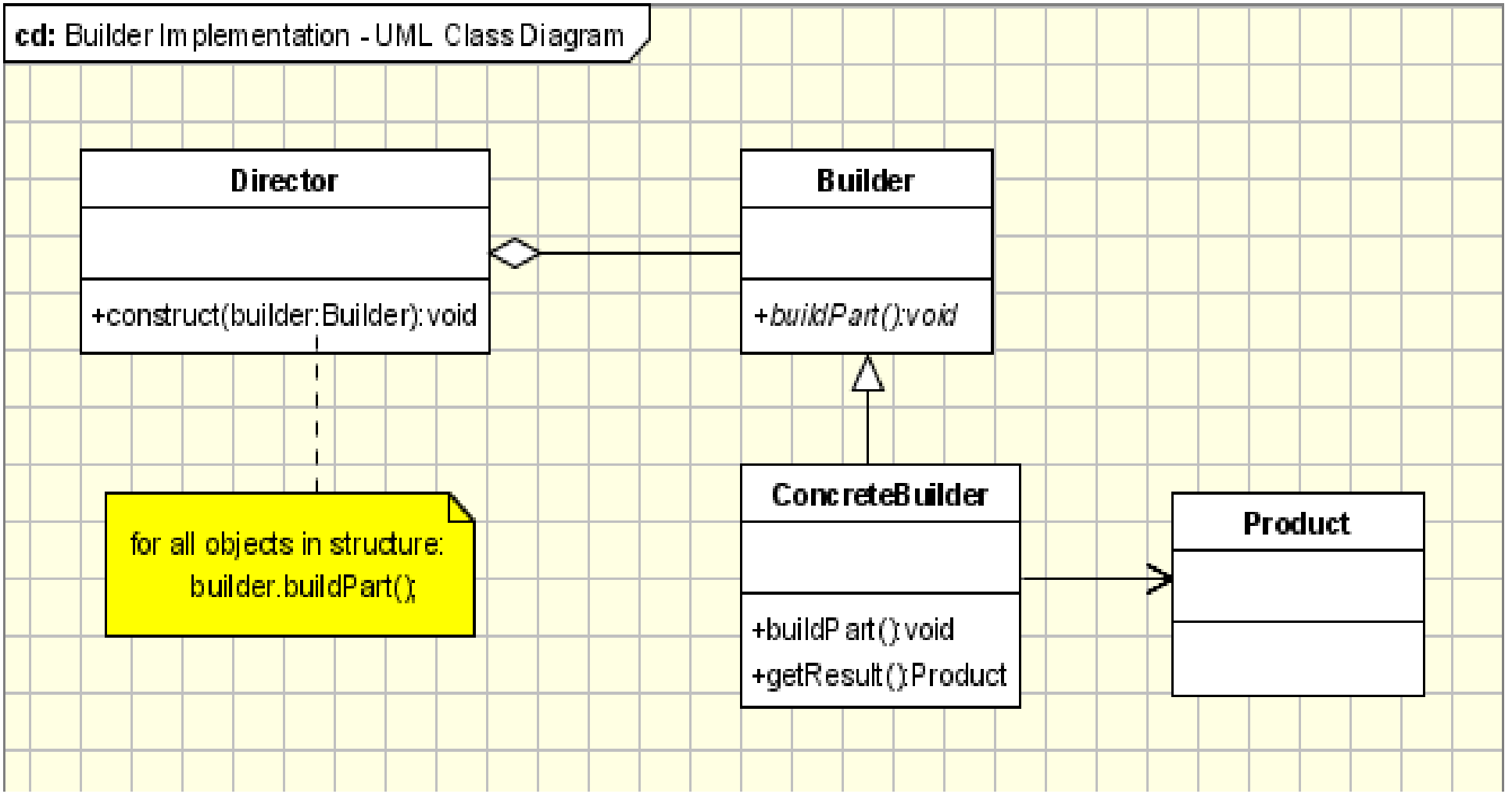


A creational pattern whose intent is to separate the construction of a complex object from its representation. By doing so the same construction process can create different representations.

Builder

The steps of construction are abstracted so that different implementations of these steps can construct different representing objects. Often used to build products in accordance with the Composite Pattern.

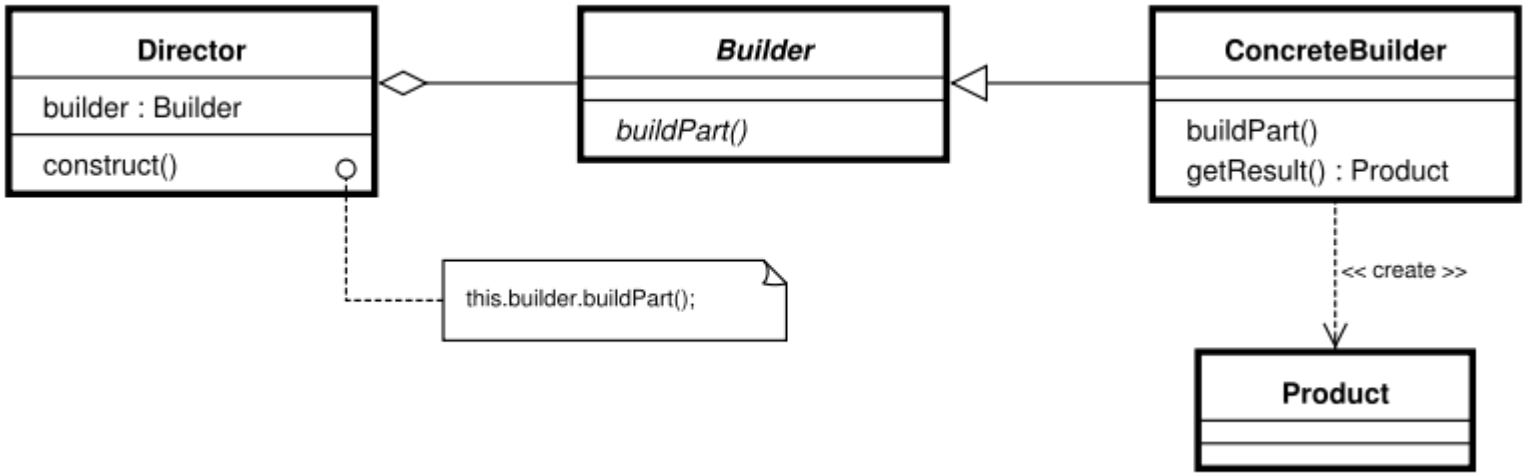
UML


cd: Builder Implementation - UML Class Diagram
Director
+construct(builder:Builder):void
Builder
+buildPart():void
for all objects in structure:
builder.buildPart();
ConcreteBuilder
+buildPart():void
+getResult():Product
Product

UML Participants

- The participants classes in this pattern are:
- The **Builder** class specifies an abstract interface for creating parts of a **Product** object.
- The **ConcreteBuilder** constructs and puts together parts of the product by implementing the **Builder** interface. It defines and keeps track of the representation it creates and provides an interface for saving the product.
- The **Director** class constructs the complex object using the **Builder** interface.
- The **Product** represents the complex object that is being built.

UML 2


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

Components of UML 2

- Builder: Abstract interface for creating objects (product)
- Concrete Builder: Provides implementations for Builder. It is an object able to construct other objects. It constructs and assembles parts to build the objects
- Director: It is responsible for managing the correct **sequence** of object creation. It receives a Concrete Builder as a parameter and executes the necessary operations on it
- Product: The final object created by the Director using Builder.

Comparison with other Abstract Factory

- Builder constructs a complex object step by step.
- Abstract Factory emphasizes a **family** of product objects.
- Builder returns the product as a final step. Abstract Factory returns the object 'immediately'.
- Builder often builds a Composite.
- Builder is more flexible, but more complex.
- Builder can use other patterns to implement which components are built.

Builder Examples

- Vehicle Manufacturer: a set of parts can be used to build a car, motorcycle, or bicycle. Builder in this case is VehicleBuilder. It specifies the interface for building any of the vehicles using the same set of parts and a different set of rules for every type of vehicle. ConcreteBuilders are attached to each of the objects that are being constructed. The Product is the vehicle that is being constructed and the Director is the manufacturer and its shop

Builder Examples: Pizza

```
class Pizza { // product
        private String dough;
        private String sauce;
        private String topping;
        private String str;

        private Pizza() {}

        public static Builder createBuilder() {
                return new Builder();
        }

        public String getDough() {
                return dough;
        }

        public String getSauce() {
                return sauce;
        }

        public String getTopping() {
                return topping;
        }
```

Pizza

```
@Override
    public String toString() {
            if (str == null)
                    str = "Dough:" + dough + " Topping:" + topping + " Sauce:" + sauce;
            return str;
    }
```

Pizza Builder

```
public static class Builder {
        private final Pizza obj = new Pizza();
        private boolean done;

        private Builder() {}

        public Pizza build() {
                done = true;
                return obj;
        }

        public Builder setDough(String dough) {
                check();
                obj.dough = dough;
                return this;
        }
```

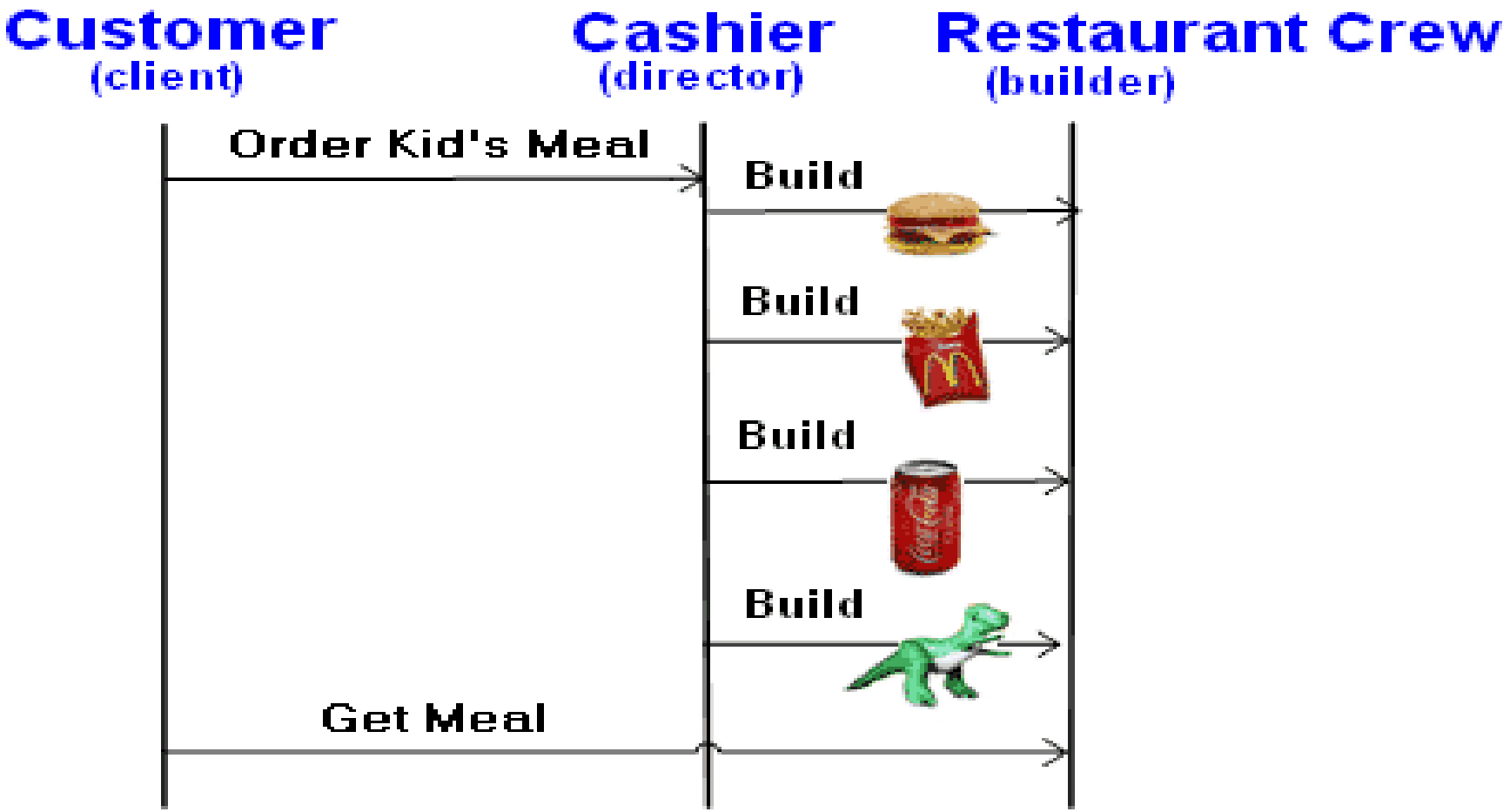
Pizza Builder (cont'd)

```
public Builder setSauce(String sauce) {
                check();
                obj.sauce = sauce;
                return this;
        }

        public Builder setTopping(String topping) {
                check();
                obj.topping = topping;
                return this;
        }

        private void check() {
                if (done)
                    throw new
                     IllegalArgumentException("Do use other builder to create new instance");
        }
    }
```

Pizza Builder Main

```
public class PizzaBuilderExample {
	public static void main(String[] args) {
		Pizza hawaiianPizza = Pizza.createBuilder()
					.setDough("cross")
					.setTopping("ham+pineapple")
					.setSauce("mild")
					.build();

		System.out.println("Hawaiian Pizza: " + hawaiianPizza);
	}
}
```

One Last Example...



Final Thoughts on Builder

- Sometimes creational patterns are complementory: Builder can use one of the other patterns to implement which components get built. Abstract Factory, Builder, and Prototype can use Singleton in their implementations.
- Builder focuses on constructing a complex object step by step. Abstract Factory emphasizes a family of product objects (either simple or complex). Builder returns the product as a final step, but as far as the Abstract Factory is concerned, the product gets returned immediately.
- Builder often builds a Composite.
- Often, designs start out using Factory Method (less complicated, more customizable, subclasses proliferate) and evolve toward Abstract Factory, Prototype, or Builder (more flexible, more complex) as the designer discovers where more flexibility is needed.