**Builder**

**Definition**

Construct a complex object from simple objects step by step.

**Where to use & benefits**

* Make a complex object by specifying only its type and content. The built object is shielded from the details of its construction.
* Want to decouple the process of building a complex object from the parts that make up the object.
* Isolate code for construction and representation.
* Give you finer control over the construction process.
* Related patterns include
  + [Abstract Factory](http://www.javacamp.org/designPattern/abstractfactory.html), which focuses on the layer over the factory pattern (may be simple or complex), whereas a builder pattern focuses on building a complex object from other simple objects.
  + [Composite](http://www.javacamp.org/designPattern/composite.html), which is often used to build a complex object.

**Example**

To build a house, we will take several steps:

1. build foundation,
2. build frame,
3. build exterior,
4. build interior.

Let's use an abstract class HouseBuilder to define these 4 steps. Any subclass of HouseBuilder will follow these 4 steps to build house (that is to say to implement these 4 methods in the subclass). Then we use a WorkShop class to force the order of these 4 steps (that is to say that we have to build interior after having finished first three steps). The TestBuilder class is used to test the coordination of these classes and to check the building process.

import java.util.\*;

class WorkShop {

//force the order of building process

public void construct(HouseBuilder hb) {

hb.buildFoundation();

hb.buildFrame();

hb.buildExterior();

hb.buildInterior();

}

}

//set steps for building a house

abstract class HouseBuilder {

protected House house = new House();

protected String showProgress() {

return house.toString();

}

abstract public void buildFoundation();

abstract public void buildFrame();

abstract public void buildExterior();

abstract public void buildInterior();

}

class OneStoryHouse extends HouseBuilder {

public OneStoryHouse(String features) {

house.setType(this.getClass() + " " + features);

}

public void buildFoundation() {

//doEngineering()

//doExcavating()

//doPlumbingHeatingElectricity()

//doSewerWaterHookUp()

//doFoundationInspection()

house.setProgress("foundation is done");

}

public void buildFrame() {

//doHeatingPlumbingRoof()

//doElectricityRoute()

//doDoorsWindows()

//doFrameInspection()

house.setProgress("frame is done");

}

public void buildExterior() {

//doOverheadDoors()

//doBrickWorks()

//doSidingsoffitsGutters()

//doDrivewayGarageFloor()

//doDeckRail()

//doLandScaping()

house.setProgress("Exterior is done");

}

public void buildInterior() {

//doAlarmPrewiring()

//doBuiltinVacuum()

//doInsulation()

//doDryWall()

//doPainting()

//doLinoleum()

//doCabinet()

//doTileWork()

//doLightFixtureBlinds()

//doCleaning()

//doInteriorInspection()

house.setProgress("Interior is under going");

}

}

class TwoStoryHouse extends HouseBuilder {

public TwoStoryHouse(String features) {

house.setType(this.getClass() + " " + features);

}

public void buildFoundation() {

//doEngineering()

//doExcavating()

//doPlumbingHeatingElectricity()

//doSewerWaterHookUp()

//doFoundationInspection()

house.setProgress("foundation is done");

}

public void buildFrame() {

//doHeatingPlumbingRoof()

//doElectricityRoute()

//doDoorsWindows()

//doFrameInspection()

house.setProgress("frame is under construction");

}

public void buildExterior() {

//doOverheadDoors()

//doBrickWorks()

//doSidingsoffitsGutters()

//doDrivewayGarageFloor()

//doDeckRail()

//doLandScaping()

house.setProgress("Exterior is waiting to start");

}

public void buildInterior() {

//doAlarmPrewiring()

//doBuiltinVacuum()

//doInsulation()

//doDryWall()

//doPainting()

//doLinoleum()

//doCabinet()

//doTileWork()

//doLightFixtureBlinds()

//doCleaning()

//doInteriorInspection()

house.setProgress("Interior is not started yet");

}

}

class House {

private String type = null;

private List features = new ArrayList();

public House() {

}

public House(String type) {

this.type = type;

}

public void setType(String type) {

this.type = type;

}

public String getType() {

return type;

}

public void setProgress(String s) {

features.add(s);

}

public String toString() {

StringBuffer ff = new StringBuffer();

String t = type.substring(6);

ff.append(t + "\n ");

for (int i = 0; i < features.size(); i ++) {

ff.append(features.get(i) + "\n ");

}

return ff.toString();

}

}

class TestBuilder {

public static void main(String[] args) {

HouseBuilder one = new OneStoryHouse("2 bedrooms, 2.5 baths, 2-car garage, 1500 sqft");

HouseBuilder two = new TwoStoryHouse("4 bedrooms, 4 baths, 3-car garage, 5000 sqft");

WorkShop shop = new WorkShop();

shop.construct(one);

shop.construct(two);

System.out.println("Check house building progress: \n");

System.out.println(one.showProgress());

System.out.println(two.showProgress());

}

}

//need jdk1.5 above to compile

| C:\ Command Prompt  C:\> javac TestBuilder.java  C:\> java TestBuilder  Check house building progress:  OneStoryHouse 2 bedrooms, 2.5 baths, 2-car garage, 1500 sqft  foundation is done  frame is done  Exterior is done  Interior is under going  TwoStoryHouse 4 bedrooms, 4 baths, 3-car garage, 5000 sqft  foundation is done  frame is under construction  Exterior is waiting to start  Interior is not started yet  C:\> |
| --- |

To fine tune the above example, every do method can be designed as a class. Similar functional class can be designed once and used by other classes. e.g. Window, Door, Kitchen, etc.

Another example, such as writing a Pizza program. Every gradient can be designed as a class. One pizza at least consists of several gradients. Different pizza has different gradients. A builder pattern may be adopted.