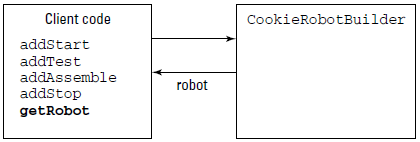
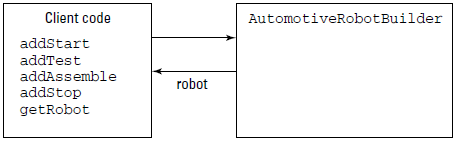
**BUILDER PATTERN**

Xem Template Method Pattern trước khhi xem tiếp

Chúng ta đi tiếp với trường hợp được nêu ra ở Template Method Pattern, lúc này yêu cầu đòi hỏi không chỉ là xây dựng một đối tượng Robot với cấu trúc skeleton có sẵn, yêu cầu lúc này là ta phải xây dựng Robot với skeleton do người sử dụng thiết lập, có nghĩa là cấu trúc của Robot sẽ đa dạng hóa và do người dùng tự quyết định. Rõ ràng với yêu cầu này thì Template Method Pattern không thể đáp ứng được, vì bản chất skeleton của nó là không đổi và không cho kế thừa. Lúc này Builder Pattern được sinh ra để giải quyết vấn đề trên.

Nguyên tắc hoạt động của Builder Pattern như sau. Ví dụ với ứng dụng trên, ta xây dựng một đối tượng sản xuất Robot, công việc của đối tượng này là cung cấp các tùy chọn để kết cấu thành Robot, sau đó khi đã hoàn thành Robot mong muốn, ta có thể gọi hàm trả về loại robot mà ta mong muốn. Ta coi thử mô hình hoạt động.





Như chúng ta thấy trong hình Client code cung cấp các hàm addStart, addTest, addAssemble, addStop để tùy chọn xây dựng Robot và hàm getRobot để trả về loại Robot cần sau khi đã xây dựng xong.

Builder Pattern có vẻ tượng tự như Factory Pattern tuy nhiên nó đa dạng hơn và tính tùy biến cao. Tiếp theo chúng ta sẽ coi thử cách code Builder Pattern như thế nào.

Đầu tiên do có nhiều kiểu xây dựng khác nhau nên ta phải tạo một interface RobotBuilder quy định cấu trúc của Builder như thế nào

public interface RobotBuilder

{

public void addStart();

public void addGetParts();

public void addAssemble();

public void addTest();

public void addStop();

public RobotBuildable getRobot();

}

Sau đó ta xây dựng các class thực thi nó.

public class CookieRobotBuilder implements RobotBuilder

{

.

.

.

}

Trong ví dụ này ta thấy nếu ta xây dựng RobotBuilder khác thì ta dùng class khác implement RobotBuilder.

Tiếp theo, ta biết là CookieRobotBuilder sẽ trả về 1 đối tượng robot, do đó ta phải có đối tượng trả về trong Builder

public class CookieRobotBuilder implements RobotBuilder

{

**CookieRobotBuildable robot;**

public CookieRobotBuilder()

{

**robot = new CookieRobotBuildable();**

}

.

.

.

}

Tiếp theo, ta sẽ xây dựng bộ ghi nhận hành động, chú ý tại sao ở đây là ghi nhận chứ không phải hành động, vì ở đây là đối tượng xây dựng, nó không can thiệp vào các thành phần nội bộ bên trong đối tượng Robot (có sự tách rời độc lập) do đó ta chỉ xây bộ ghi nhận các hành động.

public class CookieRobotBuilder implements RobotBuilder

{

CookieRobotBuildable robot;

ArrayList<Integer> actions;

public CookieRobotBuilder()

{

robot = new CookieRobotBuildable();

actions = new ArrayList<Integer>();

}

**public void addStart()**

**{**

**actions.add(new Integer(1));**

**}**

**public void addGetParts()**

**{**

**actions.add(new Integer(2));**

**}**

**public void addAssemble()**

**{**

**actions.add(new Integer(3));**

**}**

**public void addTest()**

**{**

**actions.add(new Integer(4));**

**}**

**public void addStop()**

**{**

**actions.add(new Integer(5));**

**}**

.

.

.

}

Cuối cùng ta xây dựng hàm trả về đối tượng đã xây dựng xong, tất nhiên trong đó ta cũng phải gửi kèm danh sách hành động của người dùng.

public class CookieRobotBuilder implements RobotBuilder

{

CookieRobotBuildable robot;

ArrayList<Integer> actions;

public CookieRobotBuilder()

{

robot = new CookieRobotBuildable();

actions = new ArrayList<Integer>();

}

public void addStart()

{

actions.add(new Integer(1));

}

.

.

.

public void addStop()

{

actions.add(new Integer(5));

}

**public RobotBuildable getRobot()**

**{**

**robot.loadActions(actions);**

**return robot;**

**}**

}

Tiếp theo chúng ta xây dựng class RobotBuilable, đối tượng này có tác dụng nhận chỉ thị từ Builder và xây dựng nên đối tượng thích hợp. Tuy nhiên, đầu tiên ta cũng nhớ rằng có nhiều loại Robot do đó ta cũng phải dùng interface sẵn cho nó

public interface RobotBuildable

{

public void go();

}

Sau đó ta tạo class kế thừa, hàm đọc các hành động (được gửi từ Builder) và thực thi các hành động đó

public class CookieRobotBuildable implements RobotBuildable

{

ArrayList<Integer> actions;

public CookieRobotBuildable()

{

}

**public final void go()**

**{**

**Iterator itr = actions.iterator();**

**while(itr.hasNext()) {**

**switch ((Integer)itr.next()){**

**case 1:**

**start();**

**break;**

**case 2:**

**getParts();**

**break;**

**case 3:**

**assemble();**

**break;**

**case 4:**

**test();**

**break;**

**case 5:**

**stop();**

**break;**

**}**

**}**

**}**

.

.

.

public void loadActions(ArrayList a)

{

actions = a;

}

}

Cuối cùng là ta xây dựng các phương thức hành động cụ thể của đối tượng

public class CookieRobotBuildable implements RobotBuildable

{

ArrayList<Integer> actions;

public CookieRobotBuildable()

{

}

public final void go()

{

Iterator itr = actions.iterator();

while(itr.hasNext()) {

switch ((Integer)itr.next()){

case 1:

start();

break;

case 2:

getParts();

break;

case 3:

assemble();

break;

case 4:

test();

break;

case 5:

stop();

break;

}

}

}

**public void start()**

**{**

**System.out.println(“Starting....”);**

**}**

**public void getParts()**

**{**

**System.out.println(“Getting flour and sugar....”);**

**}**

**public void assemble()**

**{**

**System.out.println(“Baking a cookie....”);**

**}**

**public void test()**

**{**

**System.out.println(“Crunching a cookie....”);**

**}**

**public void stop()**

**{**

**System.out.println(“Stopping....”);**

**}**

public void loadActions(ArrayList a)

{

actions = a;

}

}

Như vậy là ta đã xong 2 đối tượng quan trọng trong mô hình Builder Pattern, tất nhiên nếu