**TEMPLATE METHOD PATTERN**

Tình huống được đặt ra như sau, chúng ta phải xây dựng thuận toán cho robot gồm các hoạt động start, assemble, test, stop … Chúng ta sẽ bắt đầu xây dựng như sau

public class Robot

{

public Robot()

{

}

public void go()

{

start();

getParts();

assemble();

test();

stop();

}

public void start()

{

System.out.println(“Starting....”);

}

public void getParts()

{

System.out.println(“Getting a carburetor....”);

}

public void assemble()

{

System.out.println(“Installing the carburetor....”);

}

public void test()

{

System.out.println(“Revving the engine....”);

}

public void stop()

{

System.out.println(“Stopping....”);

}

}

Chúng ta thấy kỹ ngoài các hàm miêu tả hoạt động, nó còn chứa một hàm go miêu tả hoạt động chính của robot. Khi sử dụng class này, ta có thể dùng như sau

public class TestRobot

{

public static void main(String args[])

{

Robot robot = new Robot();

robot.go();

}

}

Vấn đề tiếp theo được đưa ra là ta phải thiết kế tiếp một đối tượng cookie robot, đối tượng này có các hành động tương tự như đối tượng Robot chỉ khác một vài thứ như assemble. Từ vấn đề trên người ta xây dựng nên Template Method pattern. Nguyên tắc sử dụng template method pattern như sau, ta khởi tạo một class abstract có chứa các hàm sẽ được override bởi các class con và các hàm chứa các thuật toán đó (trong ví dụ là hàm go) khi sử dụng ta có thể tạo 1 class con kế thừa class abstract này và override các phương thức. Mô hình này được ví như skeleton – khung sườn và các hàm được override xung quanh bộ khung này.

Tiếp tục với ví dụ trên theo mô hình template method pattern ta phải xây dựng class Robot lại thành class abstract.

public abstract class RobotTemplate

{

public final void go()

{

start();

getParts();

assemble();

test();

stop();

}

public void start()

{

System.out.println(“Starting....”);

}

public void getParts()

{

System.out.println(“Getting parts....”);

}

public void assemble()

{

System.out.println(“Assembling....”);

}

public void test()

{

System.out.println(“Testing....”);

}

public void stop()

{

System.out.println(“Stopping....”);

}

}

Sau khi có được class abstract mà ta gọi là khung sườn, chúng ta tiếp tục xây dựng class con kết thừa nó mà ở đây là class CookieRobot

public class CookieRobot extends RobotTemplate

{

private String name;

public CookieRobot(String n)

{

name = n;

}

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 String getName()

{

return name;

}

}

Như vậy là ta đã sử dụng Template Method Pattern, tiếp theo ta có thể thử dùng nó như sau

public class TestTemplate

{

public static void main(String args[])

{

CookieRobot cookieRobot = new CookieRobot(“Cookie Robot”);

System.out.println();

System.out.println(cookieRobot.getName() + “:”);

cookieRobot.go();

}

}

Ngoài ra ta có thể thêm vào các Hook trong khung skeleton hoặc các thuật toán vào nếu mún làm cho skeleton thêm đa dạng.

public abstract class RobotHookTemplate

{

public final void go()

{

start();

getParts();

assemble();

if (testOK()){

test();

}

stop();

}

public void start()

{

System.out.println(“Starting....”);

}

public void getParts()

{

System.out.println(“Getting parts....”);

}

public void assemble()

{

System.out.println(“Assembling....”);

}

public void test()

{

System.out.println(“Testing....”);

}

public void stop()

{

System.out.println(“Stopping....”);

}

public boolean testOK()

{

return true;

}

}

Sau đó ta có thể dùng các Hook này trong class kết thừa

public class CookieHookRobot extends RobotHookTemplate

{

private String name;

public CookieHookRobot(String n)

{

name = n;

}

public void getParts()

{

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

}

public void assemble()

{

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

}

public String getName()

{

return name;

}

public boolean testOK()

{

return false;

}

}