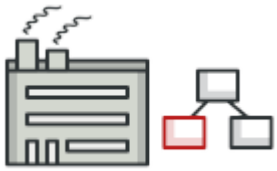

Factory Method in Java

Factory method is a creational design pattern which solves the problem of creating product objects without specifying their concrete classes.

The Factory Method defines a method, which should be used for creating objects instead of using a direct constructor call (`new` operator). Subclasses can override this method to change the class of objects that will be created.

> If you can't figure out the difference between various factory patterns and concepts, then read our **Factory Comparison**.

Learn more about Factory Method →

Navigation

```
/**
 * Common interface for all buttons.
 */
public interface Button {
    void render();
    void onClick();
}
```

buttons/HtmlButton.java: Concrete product

```
package refactoring_guru.factory_method.example.buttons;

/**
 * HTML button implementation.
 */
public class HtmlButton implements Button {

    public void render() {
        System.out.println("<button>Test Button</button>");
        onClick();
    }

    public void onClick() {
        System.out.println("Click! Button says - 'Hello World!'");
    }
}
```

buttons/WindowsButton.java: One more concrete product

```
package refactoring_guru.factory_method.example.buttons;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Windows button implementation.
 */
public class WindowsButton implements Button {
    JPanel panel = new JPanel();
    JFrame frame = new JFrame();
```

```
    public void render() {
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hello World!");
        label.setOpaque(true);
        label.setBackground(new Color(235, 233, 126));
        label.setFont(new Font("Dialog", Font.BOLD, 44));
        label.setHorizontalAlignment(SwingConstants.CENTER);
        panel.setLayout(new FlowLayout(FlowLayout.CENTER));
        frame.getContentPane().add(panel);
        panel.add(label);
        onClick();
        panel.add(button);

        frame.setSize(320, 200);
        frame.setVisible(true);
        onClick();
    }

    public void onClick() {
        button = new JButton("Exit");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                frame.setVisible(false);
                System.exit(0);
            }
        });
    }
}
```

factory

factory/Dialog.java: Base creator

```
package refactoring_guru.factory_method.example.factory;

import refactoring_guru.factory_method.example.buttons.Button;

/**
 * Base factory class. Note that "factory" is merely a role for the class. It
 * should have some core business logic which needs different products to be
 * created.
 */
public abstract class Dialog {

    public void renderWindow() {
        // ... other code ...
```

```
        okButton.render();
    }

    /**
     * Subclasses will override this method in order to create specific button
     * objects.
     */
    public abstract Button createButton();
}
```

factory/HtmlDialog.java: Concrete creator

```
package refactoring_guru.factory_method.example.factory;

import refactoring_guru.factory_method.example.buttons.Button;
import refactoring_guru.factory_method.example.buttons.HtmlButton;

/**
 * HTML Dialog will produce HTML buttons.
 */
public class HtmlDialog extends Dialog {

    @Override
    public Button createButton() {
        return new HtmlButton();
    }
}
```

factory/WindowsDialog.java: One more concrete creator

```
package refactoring_guru.factory_method.example.factory;

import refactoring_guru.factory_method.example.buttons.Button;
import refactoring_guru.factory_method.example.buttons.WindowsButton;

/**
 * Windows Dialog will produce Windows buttons.
 */
public class WindowsDialog extends Dialog {

    @Override
    public Button createButton() {
        return new WindowsButton();
```

Demo.java: Client code

```
package refactoring_guru.factory_method.example;

import refactoring_guru.factory_method.example.factory.Dialog;
import refactoring_guru.factory_method.example.factory.HtmlDialog;
import refactoring_guru.factory_method.example.factory.WindowsDialog;

/**
 * Demo class. Everything comes together here.
 */
public class Demo {
    private static Dialog dialog;

    public static void main(String[] args) {
        configure();
        runBusinessLogic();
    }

    /**
     * The concrete factory is usually chosen depending on configuration or
     * environment options.
     */
    static void configure() {
        if (System.getProperty("os.name").equals("Windows 10")) {
            dialog = new WindowsDialog();
        } else {
            dialog = new HtmlDialog();
        }
    }

    /**
     * All of the client code should work with factories and products through
     * abstract interfaces. This way it does not care which factory it works
     * with and what kind of product it returns.
     */
    static void runBusinessLogic() {
        dialog.renderWindow();
    }
}
```

OutputDemo.txt: Execution result (HtmlDialog)