Abstract Factory in Java

Abstract Factory is a creational design pattern, which solves the problem of creating entire product families without specifying their concrete classes.

Abstract Factory defines an interface for creating all distinct products but leaves the actual product creation to concrete factory classes. Each factory type corresponds to a certain product variety.

The client code calls the creation methods of a factory object instead of creating products directly with a constructor call (`new` operator). Since a factory corresponds to a single product variant, all its products will be compatible.

Client code works with factories and products only through their abstract interfaces. This lets the client code work with any product variants, created by the factory object. You just create a new concrete factory class and pass it to the client code.

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

Learn more about Abstract Factory →

Navigation

WINTER SALE IS ON!

buttons
- Button
- MacOSButton
- WindowsButton

checkboxes
- Checkbox
- MacOSCheckbox
- WindowsCheckbox

factories
- GUIFactory
- MacOSFactory
- WindowsFactory

app
- Application
- Demo
- OutputDemo

Complexity: ★★☆

Popularity: ★★★

Usage examples: The Abstract Factory pattern is pretty common in Java code. Many frameworks and libraries use it to provide a way to extend and customize their standard components.

Here are some examples from core Java libraries:

- `javax.xml.parsers.DocumentBuilderFactory#newInstance()`
- `javax.xml.transform.TransformerFactory#newInstance()`
- `javax.xml.xpath.XPathFactory#newInstance()`

Identification: The pattern is easy to recognize by methods, which return a factory object. Then, the factory is used for creating specific sub-components.

Families of cross-platform GUI components and their production

Windows.

The abstract factory defines an interface for creating buttons and checkboxes. There are two concrete factories, which return both products in a single variant.

Client code works with factories and products using abstract interfaces. It makes the same client code working with many product variants, depending on the type of factory object.

buttons: First product hierarchy

buttons/Button.java

```java
package refactoring_guru.abstract_factory.example.buttons;

/**
 * Abstract Factory assumes that you have several families of products,
 * structured into separate class hierarchies (Button/Checkbox). All products of
 * the same family have the common interface.
 *
 * This is the common interface for buttons family.
 */
public interface Button {
    void paint();
}
```

buttons/MacOSButton.java

```java
package refactoring_guru.abstract_factory.example.buttons;

/**
 * All products families have the same varieties (MacOS/Windows).
 *
 * This is a MacOS variant of a button.
 */
public class MacOSButton implements Button {

    @Override
    public void paint() {
        System.out.println("You have created MacOSButton.");
```

buttons/WindowsButton.java

```java
package refactoring_guru.abstract_factory.example.buttons;

/**
 * All products families have the same varieties (MacOS/Windows).
 *
 * This is another variant of a button.
 */
public class WindowsButton implements Button {

    @Override
    public void paint() {
        System.out.println("You have created WindowsButton.");
    }
}
```

checkboxes: Second product hierarchy

checkboxes/Checkbox.java

```java
package refactoring_guru.abstract_factory.example.checkboxes;

/**
 * Checkboxes is the second product family. It has the same variants as buttons.
 */
public interface Checkbox {
    void paint();
}
```

checkboxes/MacOSCheckbox.java

```java
package refactoring_guru.abstract_factory.example.checkboxes;

/**
```

```java
 * This is a variant of a checkbox.
 */
public class MacOSCheckbox implements Checkbox {

    @Override
    public void paint() {
        System.out.println("You have created MacOSCheckbox.");
    }
}
```

checkboxes/WindowsCheckbox.java

```java
package refactoring_guru.abstract_factory.example.checkboxes;

/**
 * All products families have the same varieties (MacOS/Windows).
 *
 * This is another variant of a checkbox.
 */
public class WindowsCheckbox implements Checkbox {

    @Override
    public void paint() {
        System.out.println("You have created WindowsCheckbox.");
    }
}
```

factories

factories/GUIFactory.java: Abstract factory

```java
package refactoring_guru.abstract_factory.example.factories;

import refactoring_guru.abstract_factory.example.buttons.Button;
import refactoring_guru.abstract_factory.example.checkboxes.Checkbox;

/**
 * Abstract factory knows about all (abstract) product types.
 */
public interface GUIFactory {
}
```

factories/MacOSFactory.java: Concrete factory (macOS)

```
package refactoring_guru.abstract_factory.example.factories;

import refactoring_guru.abstract_factory.example.buttons.Button;
import refactoring_guru.abstract_factory.example.buttons.MacOSButton;
import refactoring_guru.abstract_factory.example.checkboxes.Checkbox;
import refactoring_guru.abstract_factory.example.checkboxes.MacOSCheckbox;

/**
 * Each concrete factory extends basic factory and responsible for creating
 * products of a single variety.
 */
public class MacOSFactory implements GUIFactory {

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

    @Override
    public Checkbox createCheckbox() {
        return new MacOSCheckbox();
    }
}
```

factories/WindowsFactory.java: Concrete factory (Windows)

```
package refactoring_guru.abstract_factory.example.factories;

import refactoring_guru.abstract_factory.example.buttons.Button;
import refactoring_guru.abstract_factory.example.buttons.WindowsButton;
import refactoring_guru.abstract_factory.example.checkboxes.Checkbox;
import refactoring_guru.abstract_factory.example.checkboxes.WindowsCheckbox;

/**
 * Each concrete factory extends basic factory and responsible for creating
 * products of a single variety.
```

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

    @Override
    public Checkbox createCheckbox() {
        return new WindowsCheckbox();
    }
}
```

app

app/Application.java: Client code

```
package refactoring_guru.abstract_factory.example.app;

import refactoring_guru.abstract_factory.example.buttons.Button;
import refactoring_guru.abstract_factory.example.checkboxes.Checkbox;
import refactoring_guru.abstract_factory.example.factories.GUIFactory;

/**
 * Factory users don't care which concrete factory they use since they work with
 * factories and products through abstract interfaces.
 */
public class Application {
    private Button button;
    private Checkbox checkbox;

    public Application(GUIFactory factory) {
        button = factory.createButton();
        checkbox = factory.createCheckbox();
    }

    public void paint() {
        button.paint();
        checkbox.paint();
    }
}
```

```java
package refactoring_guru.abstract_factory.example;

import refactoring_guru.abstract_factory.example.app.Application;
import refactoring_guru.abstract_factory.example.factories.GUIFactory;
import refactoring_guru.abstract_factory.example.factories.MacOSFactory;
import refactoring_guru.abstract_factory.example.factories.WindowsFactory;

/**
 * Demo class. Everything comes together here.
 */
public class Demo {

    /**
     * Application picks the factory type and creates it in run time (usually at
     * initialization stage), depending on the configuration or environment
     * variables.
     */
    private static Application configureApplication() {
        Application app;
        GUIFactory factory;
        String osName = System.getProperty("os.name").toLowerCase();
        if (osName.contains("mac")) {
            factory = new MacOSFactory();
        } else {
            factory = new WindowsFactory();
        }
        app = new Application(factory);
        return app;
    }

    public static void main(String[] args) {
        Application app = configureApplication();
        app.paint();
    }
}
```

OutputDemo.txt: Execution result

```
You create WindowsButton.
You created WindowsCheckbox.
```

WINTER SALE IS ON!

← Design Patterns in Java

Builder in Java →

Abstract Factory in Other Languages

Home Refactoring Design Patterns Premium Content
Forum Contact us

Ukrainian office:
FOP Olga Skobeleva
Abolmasova 7
Kyiv, Ukraine, 02002
Email:
support@refactoring.guru

Spanish office:
Oleksandr Shvets
Avda Pamplona 64
Pamplona, Spain, 31009
Email:
support@refactoring.guru