

## COSC 436: Object-Oriented Design and Programming

### In-class Exercise: Factory Method Design Pattern

#### Problem:

The objective of this exercise is to implement the Factory Method design pattern.

#### Tasks:

In this exercise, **Buttons** play a product role and **Dialogs** act as creators. Different types of dialogs require their own types of elements. We will create a subclass for each **dialog** type and **override** their factory methods. Each **dialog** type will instantiate proper **button** classes. **Base dialog** works with products using their common interface, so its code remains functional after all changes.

1. We have an interface, called **Button**, which defines two methods, **render()** and **onClick()**.

```
1 public interface Button {
2     void render();
3     void onClick();
4 }
```

2. Create an **HtmlButton** class, which implements Button. Provide the implementation for both **render()** and **onClick()** methods.

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

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

```
public class HtmlButton implements Button {
    @Override
    public void render() {
        System.out.println(x:"<button>Test Button</button>");
        onClick();
    }

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

3. Create a **WindowsButton** class, which also implements Button. Provide the implementation for both **render()** and **onClick()** methods.

```
JPanel panel = new JPanel();
JFrame frame = new JFrame();
JButton button;
```

Name: E10\_David\_Mugwaneza

```
public void render() {  
}  
  
public void onClick() {  
}  
}
```

```
import java.awt.event.ActionEvent;  
import javax.swing.*;  
  
public class WindowsButton implements Button {  
    JPanel panel = new JPanel();  
    JFrame frame = new JFrame();  
    JButton button;  
  
    @Override  
    public void render() {  
        frame.setTitle(title:"Windows Button");  
        button = new JButton(text:"Test Button");  
        button.addActionListener((ActionEvent e) -> {  
            onClick();  
        });  
  
        panel.add(button);  
        frame.add(panel);  
        frame.setSize(width:200, height:100);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(b:true);  
    }  
  
    @Override  
    public void onClick() {  
        JOptionPane.showMessageDialog(frame, message:"Click! Button says - 'Hello World!'");  
    }  
}
```

4. Create a base creator, called **Dialog**. Have two methods in it.

```
public void renderWindow() {  
}  
public abstract Button createButton();  
  
public abstract class Dialog {  
    public void renderWindow() {  
        Button okButton = createButton();  
        okButton.render();  
    }  
  
    public abstract Button createButton();  
}
```

Name: E10\_David\_Mugwaneza

5. Create two concrete creators: **HtmlDialog** and **WindowsDialog**, by providing implementation for the abstract method `createButton()`. One returns new `WindowsButton()`, the other returns new `HtmlButton()`.

```
public class HtmlDialog extends Dialog {
    @Override
    public Button createButton() {
        return new HtmlButton();
    }
}

public class WindowsDialog extends Dialog
    @Override
    public Button createButton() {
        return new WindowsButton();
    }
}
```

6. Use the following client code to test it.

```
public class Client {
    private static Dialog dialog;

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

    /**
     * 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();
        }
    }

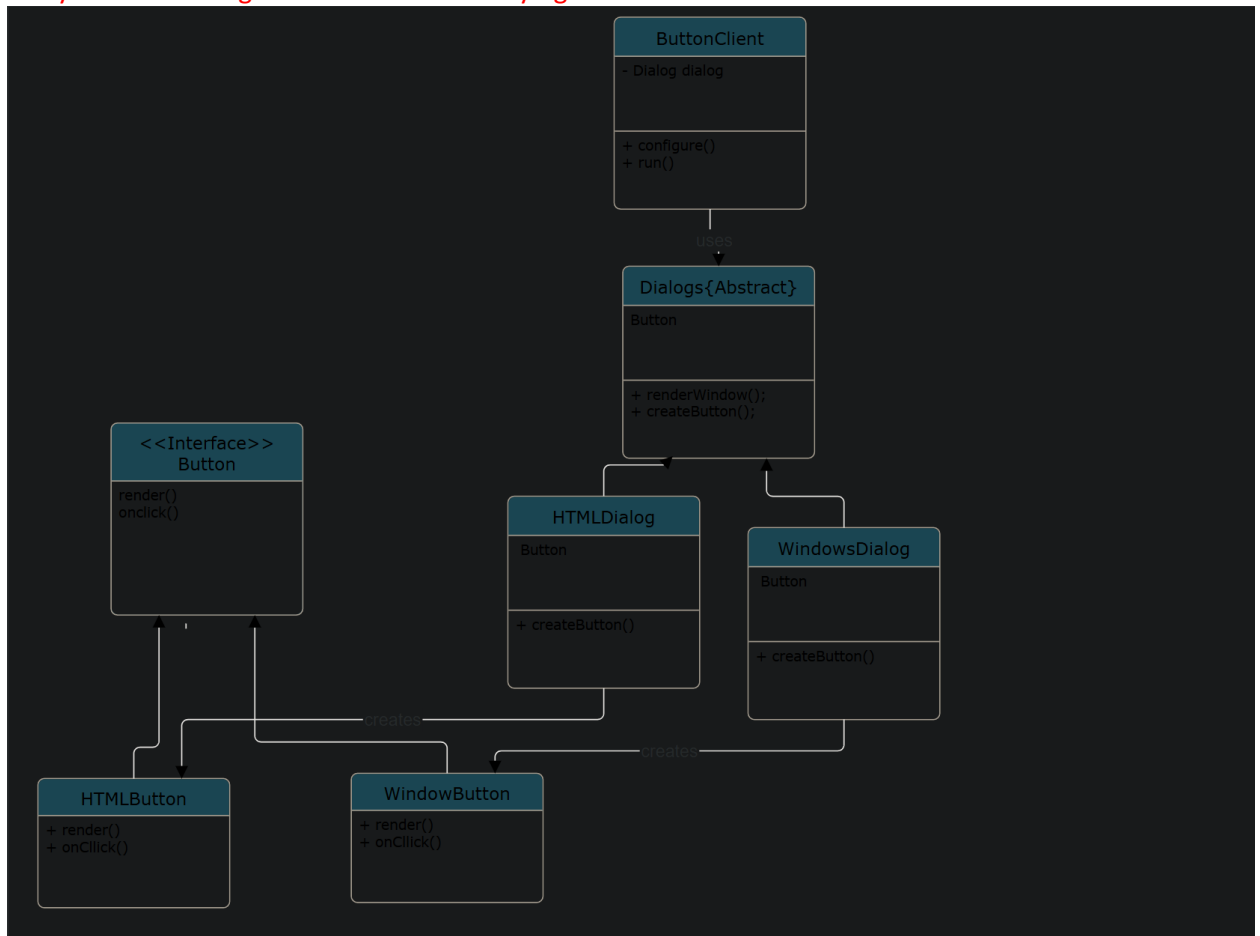
    /**
     * All of the client code should work with factories and products through
     * abstract interfaces. This way it does not care which factory it works
     * with and what kind of product it returns.
     */
    static void run() {
        dialog.renderWindow();
    }
}
```

Name: E10\_David\_Mugwaneza



7. Can you draw a class diagram for these classes?

Sorry couldn't change the color without buying the license.



Upload your code to the Blackboard when you are done.