Types of Pop-up Messages

Alert Pop-ups: These are simple information or warning messages that appear as pop-ups. They often have an OK button to dismiss them.
Confirmation Pop-ups: These pop-ups typically have options like "OK" and "Cancel" and are used to confirm or cancel an action.
Prompt Pop-ups: These pop-ups prompt the user to enter some text or data. They have an input field along with "OK" and "Cancel" buttons.

Handling Alert Pop-ups:

Use **driver.switchTo().alert()** to switch the WebDriver's focus to the alert.
Use **alert.accept()** to accept the alert (click the OK button).
Use **alert.dismiss()** to dismiss the alert (click the Cancel button).
Use **alert.getText()** to retrieve the text from the alert.

Handling Confirmation Pop-ups:

Switch to the alert using **driver.switchTo().alert().**
Use **alert.accept()** to confirm the action (click OK) or **alert.dismiss()** to cancel it (click Cancel).

Handling Prompt Pop-ups:

Switch to the alert using **driver.switchTo().alert().**
Use **alert.sendKeys**("text") to enter text into the prompt input field.
Use **alert.accept()** to submit the input or **alert.dismiss()** to cancel it.

Switching Between Windows:

When dealing with pop-ups that open new browser windows or tabs, use **driver.getWindowHandles()** to get a set of window handles.

Use **driver.switchTo().window(windowHandle)** to switch between different windows.

Handling Pop-ups in Frames:

Use **driver.switchTo().frame(frameLocator)** to switch to a frame if the pop-up is inside a frame.

Implicit and Explicit Waits:

Use waits (ImplicitWait or WebDriverWait) to handle the timing of pop-ups and their elements.

Alert Presence Check:

You can check for the presence of an alert using ExpectedConditions.alertIsPresent() in conjunction with WebDriverWait.

Handling Unhandled Pop-ups:
Sometimes, websites use custom pop-ups that don't trigger the browser's built-in alerts. In such cases, you may need to handle them as regular web elements.
Error Handling:
Always use try-catch blocks to handle exceptions that may occur when dealing with pop-ups.

Example of handling an alert pop-up in Selenium:

```
import org.openqa.selenium.Alert;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class HandleAlertPopUp {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path_to_chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        driver.get("https://example.com");

        // Click a button that triggers an alert
        driver.findElement(By.id("alertButton")).click();

        // Switch to the alert
        Alert alert = driver.switchTo().alert();

        // Get the alert text
        String alertText = alert.getText();
        System.out.println("Alert Text: " + alertText);

        // Accept the alert (click OK)
        alert.accept();

        driver.quit();
    }
}
```

1. Open Banking Website

You need to automate opening a banking website using Selenium WebDriver. This involves launching a browser (Chrome in this case), navigating to the Axis banking website, and performing subsequent actions on the webpage.

2. Handle Browser Notification Settings

You need to set preferences using a `HashMap` to control browser notification settings.
- If the argument passed is `1`, notifications should be allowed.
- If the argument passed is `2`, notifications should be blocked.

In Selenium WebDriver, this is typically done using ChromeOptions to set experimental preferences (`prefs`). You configure these preferences before initializing the WebDriver instance.

3. Handle Popup After Opening the Website

After navigating to the banking website, there will be a popup (possibly an alert or a modal) that needs to be handled programmatically.

Axis Bank website:

Step 1: Navigate to `https://www.axisbank.com/`.
Step 2: Set preferences to block browser notifications using `ChromeOptions`.
Step 3: Identify and handle any popups that appear after navigating to the website, such as a cookie consent popup or a promotional modal.

Implementation Steps:

```
import java.util.HashMap;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
```

```
public class BankingWebsiteTest {
    public static void main(String[] args) {
        // Step 1: Set up ChromeDriver and WebDriver
        System.setProperty("webdriver.chrome.driver", "D:\\ATT\\chromedriver.exe");
        ChromeOptions options = new ChromeOptions();

        // Step 2: Set preferences to handle browser notifications
        int notificationSetting = 2; // Change to 1 for allowing notifications, 2 for blocking
        HashMap<String, Object> prefs = new HashMap<>();
        prefs.put("profile.default_content_setting_values.notifications", notificationSetting);
        options.setExperimentalOption("prefs", prefs);

        WebDriver driver = new ChromeDriver(options);
        driver.manage().window().maximize();

        // Step 3: Navigate to the banking website
        driver.get("https://www.axisbank.com/");

        // Step 4: Handle any popup after opening the website
        try {
            // Example: Handle a popup by clicking on an element
            WebElement popupElement = driver.findElement(By.id("popup_id")); // Replace with
actual popup element identifier
            popupElement.click();

            // Alternatively, handle an alert if it appears
            // Alert alert = driver.switchTo().alert();
            // alert.accept(); // Handle alert by accepting it (click OK)
        } catch (Exception e) {
            // Handle exceptions (e.g., popup not found)
            e.printStackTrace();
        } finally {
            // Step 5: Close the browser session
            driver.quit();
```

```
        }
    }
}
```

Explanation:
HashMap is a data structure in Java that stores key-value pairs. In the context of Selenium WebDriver, HashMap is often used to store preferences or configuration settings that can be passed to ChromeOptions or other WebDriver configurations.

In the experiment, HashMap<String, Object> prefs is used to store preferences related to browser notifications ("profile.default_content_setting_values.notifications"). Depending on the value set in the HashMap (1, 2), specific behaviors like allowing or blocking notifications can be controlled when configuring ChromeOptions.