Notification Showing: Strategy use case

https://github.com/unitools-apps/UniTools-android

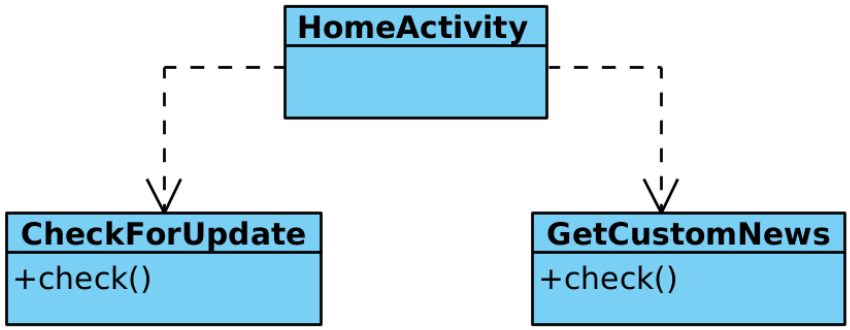
☹ Problem

The `GetCustomNews` and `CheckForUpdate` classes differ only in the execution of a portion of their `check() method. It is precisely` in the last lines of the `check()` method where a behavior is executed, always knowing that a condition is met, where it is checked if the notification is really needed to display it[i].

The small differences mean that, if a new type of check is implemented, this check will also have a similar implementation, differing only in its last lines. Therefore, in the face of small differences and large copy-pastes of code, there must be a way to avoid duplication of code around the codebase.

The client class, `HomeActivity`, makes use of the check() methods of the `GetCustomNews` and `CheckForUpdate` classes in its `onCreate() method`.

`UML Diagram`



GetCustomNews.java

```java
public class GetCustomNews {

    private static final String API_URL = "https://unitools.ir/config/pushNotification.json";

    public static void Check() {

        new Thread(() -> {
            try {
                URL url = new URL(API_URL);
                BufferedReader in =
                        new BufferedReader(new InputStreamReader(url.openStream()));
                StringBuilder page = new StringBuilder();
                String inLine;

                while ((inLine = in.readLine()) != null) {
                    page.append(inLine);
                }

                in.close();

                PushNotifyModel pnm = new Gson().fromJson(page.toString(), PushNotifyModel.class);
                if (pnm.enable)
                    AppNotification.ShowCustomNewsNotification(pnm.title, pnm.text, pnm.link);

            } catch (Exception e) {
                e.printStackTrace();
            }
        }).start();
    }
}
```

CheckForUpdate.java

```
public class CheckForUpdate {

    private static final String API_URL = "https://unitools.ir/config/version.txt";

    public static void Check() {

        new Thread(() -> {
            try {
                URL url = new URL(API_URL);
                BufferedReader in =
                        new BufferedReader(new InputStreamReader(url.openStream()));
                StringBuilder page = new StringBuilder();
                String inLine;

                while ((inLine = in.readLine()) != null) {
                    page.append(inLine);
                }



                  in.close();
                  if (!CH.getString(R.string.app_version).equals(page.toString()))
                      AppNotification.ShowUpdateAvailable();

              } catch (Exception e) {
                  e.printStackTrace();
              }
          }).start();
      }
  }
```

HomeActivity.java

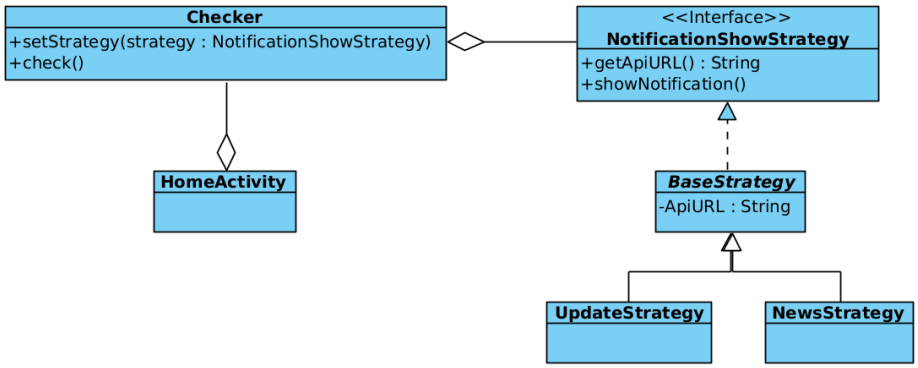
```
public class HomeActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        CH.initStatics(this);
        SetupNav();
        CheckForUpdate.Check();
        GetCustomNews.Check();
        ReminderAndAutoSilent.Setup(this);

        MyDataBeen.onAppStarts(this);
    }
```

😊 Solution

The Strategy pattern allows you to extract the variant behavior, between `CheckForUpdate` and GetCustomNews, `into a separate class hierarchy from` the Strategy NotificationShowStrategy interface and combine the original classes into a Checker class, thereby reducing duplicate code.

In addition, for the client class, `HomeActivity`, the algorithmic variants of checking, between an update and a news item, can be swapped during initialization in the `onCreate() implementation`.

`UML Diagram`



`NotificationShowStrategy.java`

```
package strategy;

public interface NotificationShowStrategy {
    public String getApiURL();
    public void showNotification();
}
```

BaseStrategy.java

```java
package strategy;

public abstract class BaseStrategy implements NotificationShowStrategy {
    protected String ApiURL;

    @Override
    public String getApiURL(){
        return ApiURL;
    }
}
```

UpdateStrategy.java

```java
package strategy;

public class UpdateStrategy extends BaseStrategy {


    public UpdateStrategy(){
        this.ApiURL = "https://unitools.ir/config/version.txt";
    }

    @Override
    public void showNotification() {
        AppNotification.ShowUpdateAvailable();
    }

}
```

NewsStrategy.java

```java
package strategy;

public class NewsStrategy extends BaseStrategy {

    public NewsStrategy(){
        this.ApiURL = "https://unitools.ir/config/pushNotification.json";
    }

    @Override
    public void showNotification() {
        PushNotifyModel pnm = new Gson().fromJson(page.toString(), PushNotifyModel.class);
        AppNotification.ShowCustomNewsNotification(pnm.title, pnm.text, pnm.link);

    }

}
```

Checker.java

```java
public class Checker {
    NotificationShowStrategy strategy;

    public void setStrategy(NotificationShowStrategy strategy){
        this.strategy = strategy;
    }

    public void check(){
        new Thread(() -> {
            try {
                URL url = new URL(spec: strategy.getApiURL());
                BufferedReader in =
                        new BufferedReader(new InputStreamReader(in: url.openStream()));

                StringBuilder page = new StringBuilder();
                String inLine;

                while ((inLine = in.readLine()) != null) {
                    page.append(str: inLine);
                }
                in.close();
                strategy.showNotification();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }).start();
    }
}
```

HomeActivity.java

```java
package activities;

import strategy.Checker;
import strategy.NewsStrategy;
import strategy.UpdateStrategy;

//...

public class HomeActivity {
    private Checker checker;

    // ...
    protected void onCreate() {
        // ...
        checker.setStrategy(new UpdateStrategy());
        checker.check();
        checker.setStrategy(new NewsStrategy());
        checker.check();
        // ...
    }

    // ...
}
```

⚖ Pros and Cons

✓The details of the implementation of the algorithm are separated to show notifications depending on whether it is a news story, a version update and another one that appears in the future. The separation falls on the implementation and the `Checker` that uses it for `HomeActivity`.

✓Open Closed Principle. You can introduce new notification strategies without modifying `Checker` or `HomeActivity`, customer classes.

✕The complexity of understanding and introducing new classes and interfaces, to provide a solution to a small portion of the codebase.

---

[i] Observer is a design pattern applicable in the call of these methods from a publisher to subscribers of this type, considering the following conditions presented:
The AppNotification::ShowUpdateAvailable `method is called` when there is only a version change of the app.
The AppNotification::ShowCustomNewsNotification method is called only if a `PushNotificationModel` is enabled.