Memory Management

Screen Orientation Change

- When the screen orientation changes the android platform destroys the current activity and create a new one.
- In doing so the resources are reloaded in order to recreate the user interface controls.
- When dealing with huge resources (e.g. big bitmap images) we can save memory and avoid the reload by keeping them in static fields.

Screen Orientation Change

```
...
private static Drawable background;

@Override
protected void onCreate(Bundle state)
{
    super.onCreate(state);
    TextView txt = new TextView(this);
    txt.setText("Leaks are bad");
    if(background==null)
    {
        background = getDrawable(R.drawable.image);
    }
    label.setBackgroundDrawable(background);
    ...
    setContentView(label);
}
...
```

Holding the resource in a static variable will avoid its reloading when the application changes its screen orientation.

Static Inner Classes

- We better use static inner classes instead of non static ones. This way less memory might be consumed.

Weak References

- Wrap references for less important and relatively big objects using `WeakReference` objects.

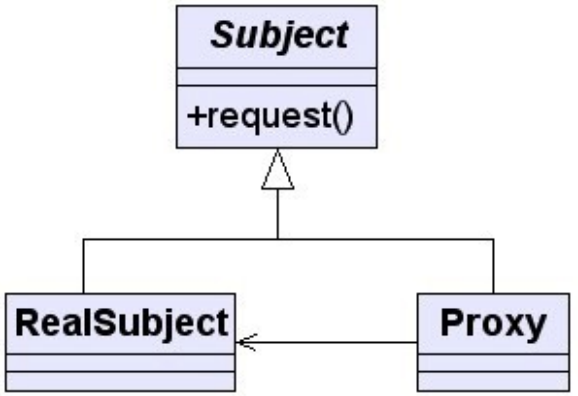
Garbage Collector

- The garbage collector is not perfect. Calling `System.gc()` from time to time is a good practice for avoiding memory problems.

The Proxy Design Pattern

- Implementing this design pattern (when relevant) our application will consume less memory.

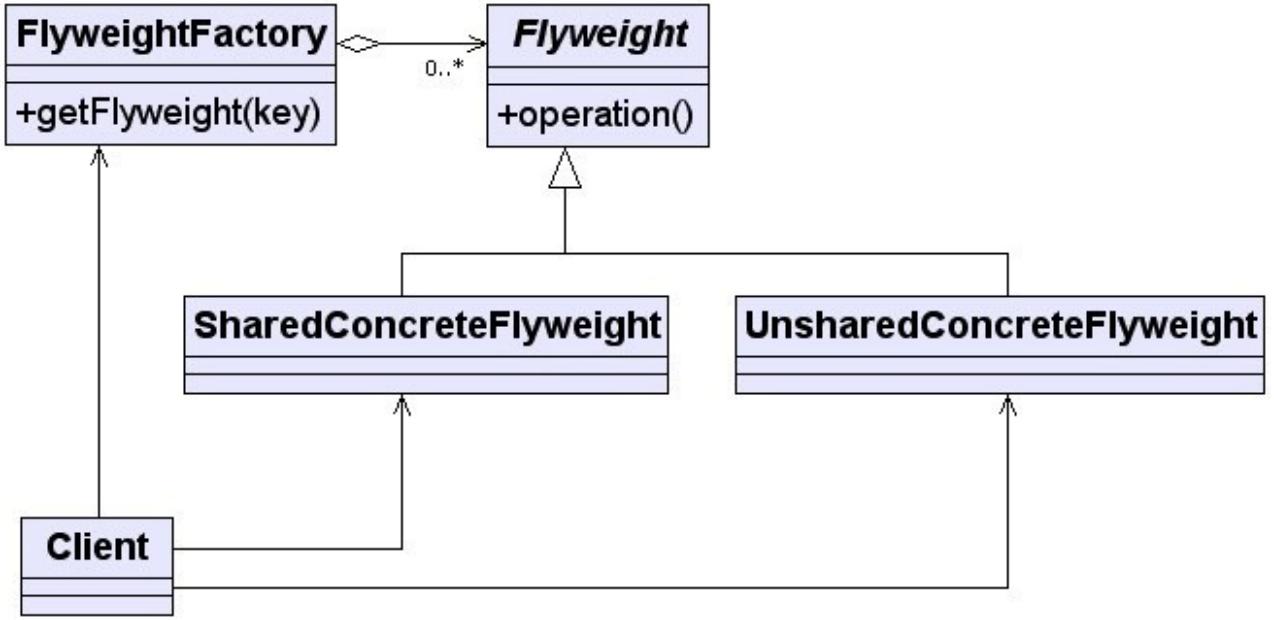
The Proxy Design Pattern



The Flyweight Design Pattern

- Implementing this design pattern (when relevant) our application will consume less memory.

The Flyweight Design Pattern



Static References

- Be careful of holding either directly or indirectly the context (activity) reference in a static variable. Such cases will interfere the garbage collector work.

Extending The Application Class

- We can define a new class that extends Application and configure it as the application class.

```
<application android:icon="@drawable/icon"
android:label="@string/app_name"
android:name="com.lifemichael.MyApplication">
        ...
</application>
```

Extending The Application Class

- Using the application context for maintaining the application resources.
- Doing so we will avoid recreating the resources in according with the activity life cycle.

Memory Management

Screen Orientation Change

- When the screen orientation changes the android platform destroys the current activity and create a new one.
- In doing so the resources are reloaded in order to recreate the user interface controls.
- When dealing with huge resources (e.g. big bitmap images) we can save memory and avoid the reload by keeping them in static fields.

Screen Orientation Change

```
...
private static Drawable background;

@Override
protected void onCreate(Bundle state)
{
    super.onCreate(state);
    TextView txt = new TextView(this);
    txt.setText("Leaks are bad");
    if(background==null)
    {
        background = getDrawable(R.drawable.image);
    }
    label.setBackgroundDrawable(background);
    ...
    setContentView(label);
}
...
```

Holding the resource in a static variable will avoid its reloading when the application changes its screen orientation.

Static Inner Classes

- We better use static inner classes instead of non static ones. This way less memory might be consumed.

Weak References

- Wrap references for less important and relatively big objects using `WeakReference` objects.

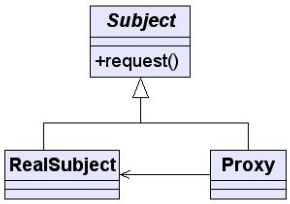
Garbage Collector

- The garbage collector is not perfect. Calling `System.gc()` from time to time is a good practice for avoiding memory problems.

The Proxy Design Pattern

- Implementing this design pattern (when relevant) our application will consume less memory.

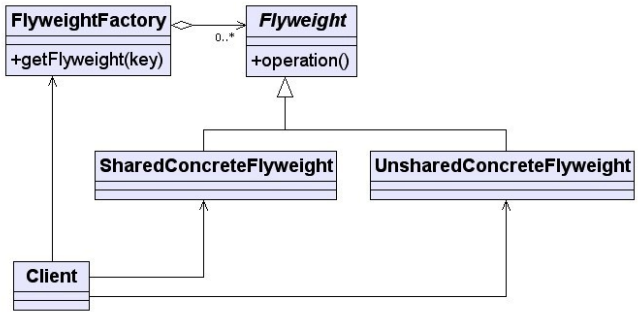
The Proxy Design Pattern



The Flyweight Design Pattern

- Implementing this design pattern (when relevant) our application will consume less memory.

The Flyweight Design Pattern



Static References

- Be careful of holding either directly or indirectly the context (activity) reference in a static variable. Such cases will interfere the garbage collector work.

Extending The Application Class

- We can define a new class that extends Application and configure it as the application class.

```
<application android:icon="@drawable/icon"
android:label="@string/app_name"
android:name="com.lifemichael.MyApplication">
        ...
</application>
```

Extending The Application Class

- Using the application context for maintaining the application resources.
- Doing so we will avoid recreating the resources in according with the activity life cycle.