# Design Patterns

## Introductie

Een Design Pattern is een bepaalde methode die je kan gebruiken om specifieke problemen op te lossen of de code beter te maken door bijvoorbeeld de ‘coupling’ laag te houden.

Er bestaan allerlei design patterns die allemaal verschillende functies hebben. Dit verslag zal zich enkel richten op het design pattern dat gebruikt werd, namelijk een Facade.

## Facade

Een facade is een design pattern die ervoor zal zorgen dat bepaalde stukken code afgeschermd worden en enkel door één klasse aangesproken kan worden. Het is een soort interface voor een deel van de code.

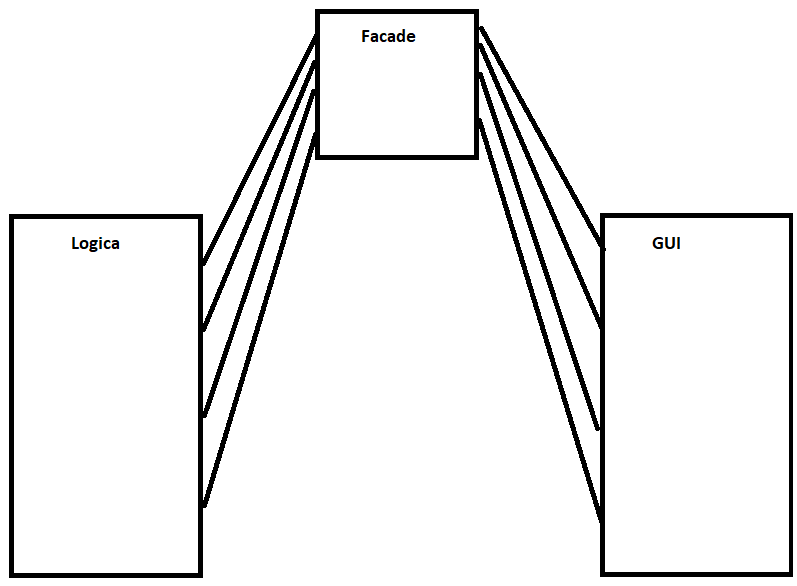
Een facade heeft meerdere voordelen;

* Interface: mensen die de afgeschermde code gebruiken, kunnen dit doen vanuit één klasse zonder de inwendige structuur of werking van heel de code te begrijpen. Dit vergemakkelijkt het gebruik
* Low coupling: Het programma dat gebruik maakt van de afgeschermde code is enkel ge’coupled’ aan de facadeklasse. Dit wil zeggen dat hij niet afhankelijk is aan de code en dus ook niet aangepast moet worden bij een verandering van die code. Enkel de facade moet aangepast worden
* Het zorgt voor een duidelijkere en overzichtelijkere code.

## Toegepast

Wij gebruiken de facade om de GUI-klasse waarin alle interactie met de user gebeurt af te zonderen van de logica. Deze klasse noemt ‘Control’  
Als de informatie van een game moet opgevraagd worden, gebeurt dit via de facade. Als een bepaald veld zoals instellingen aangepast moet worden, gebeurt dit via de facade.

Zo is de GUI niet gebonden aan de interne logica en bij bepaalde aanpassingen van deze code zal de GUI geen problemen ondervinden.



# GUI

## Introductie

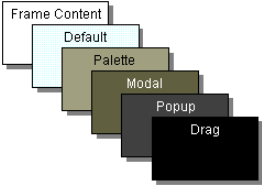
Dit verslag zal de werking van de GUI verklaren. Eerst worden er wat termen toegelicht waarna de code zal worden geanalyseerd.

De GUI zal geschreven worden in Java gebruikmakend van de GUI-library ‘swing’, wat een extensie van de oudere library ‘awt’ is.

Container  
Een container is een object waarin componenten kunnen zitten. Dit is het overkoepelende ding waar je later objecten zoals knopjes of foto’s in zet.   
Een container is eigenlijk zelf ook een component die componenten in zichzelf zet.   
De drie soorten containers zijn

* Panel: Het simpelste van de drie. Gewoon een plaats waarin componenten (dus ook andere panels) passen.
* Frame: een top-level container. Heeft decorations (werkbalk vanboven met titel en knopjes)
* Window: een top-level container zonder decorations.

Een container is opgedeeld in meerdere lagen met elks hun eigen functionaliteit

* Root pane: de bovenste pane die alles omvat.
* Content pane: Hierin zitten alle componenten, alle objecten.
* Glass pane: een onzichtbare laag die events kan onderscheppen en de normale laag kan overschrijven. Voorbeeld is een knop die enkel op bepaalde condities verschijnt. Als de glass pane nooit nodig is, kan men hier gewoon alle event listeners op zetten.
* Layered pane: Deze laag bevat diepte zodat bepaalde componenten over de andere kunnen schrijven. Bijvoorbeeld een pop-up menu dat over de default componenten schrijft of een venster dat je wegsleept dat even over alle rest verschijnt

## Component

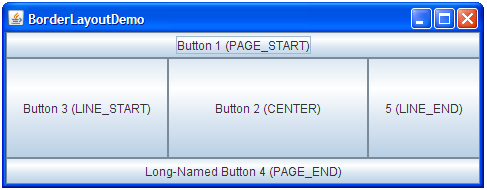
Een component is een object zoals een knop, foto of balkje die in een container moet zitten. Deze moeten meestal een event handling hebben door middel van een action listener. Eerst zullen we enkele components toelichten die we nodig zullen hebben in het maken van onze GUI.

* JLabel: Een vakje waarin tekst of een afbeelding gezet kan worden.
* JButton: Een knopje
* JColorChooser: Een palet waarin je een kleur kan selecteren.
* ImageIcon: De initialisatie van een foto-object
* JSlider: Een schuifbalk dat waardes in een gegeven interval kan aanduiden.

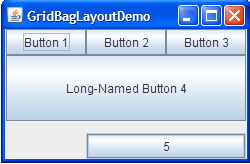
## Layout manager

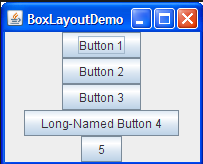
Java Swing heeft zogenaamde ‘Layout managers’ die ervoor zorgen dat de componenten goed geordend worden. Als men de ‘absolute layout’ zou moeten gebruiken, namelijk voor elke component vaste pixels of marges definiëren, zou dit niet enkel vrij verwarrend en abstract zijn, maar al snel voor problemen zorgen bij een andere schermresolutie of bij het aanpassen van de grootte van het programma.

Hiervoor bestaan Layout Managers die dit werk niet enkel versimpelen maar ook beschermen tegen fouten.   
De layout managers lijken erg simpel op begin, maar deze kunnen dus ook genest worden. Een top-level frame kan via een bepaalde layout managers panes ordenen, welke dan weer hun componenten via een andere layout manager ordenen.

Er zijn een achttal layout managers, maar voor eenvoud zullen hier enkel de gebruikte worden toegelicht.

* BorderLayout: bevat de vier windrichten en het midden. Belangrijk om weten is dat de bovenste en onderste balken de linkse en rechtse overschrijven. Verder is het gebruik erg simpel en voor de hand liggend.



* GridBagLayout: De ingewikkeldste van ze allemaal. Bij deze layout kan een container opgedeeld worden in rijen en kolommen waarin componenten kunnen geplaatst worden, met of zonder marges, over één of meerdere cellen.
* Box: Een simpele layout manager die uit één kolom bestaat waarin componenten onder elkaar geplaatst kunnen worden. Heeft ook een ‘vertical strut’ om een spatie tussen componenten te maken

Bij de GridBagLayout wordt de ordening van de componenten gedaan via een zogenaamde ‘GridBagConstraints’ welke allerlei attributen heeft zoals de x en y positie, padding, grootte, enz enz. Deze attributen moeten telkens aangepast worden bij het toevoegen van een nieuw component. Als er dus een component komt die wél padding heeft, moet die padding dus terug ook op 0 worden gezet bij een volgende component die geen padding heeft.

## Threads

Swing werkt met Threads om ervoor te zorgen dat het menu nooit bevriest of geen actie kan opnemen. De drie hoofdcategorieën zijn:

* Initial thread: De init code uitvoeren
* Event dispatch thread: Hier wordt alle event handling uitgevoerd.
* Worker thread: Dit zijn de threads waarin werk-intensieve processen worden uitgevoerd.

Deze threads zijn al geïntegreerd in in de Swing framework.

## Event listener

Een component moet weten wanneer er een actie is gebeurt. Is er op een knop gedrukt of is er een toets ingetypt, moet de knop dit weten en een bepaalde actie uitvoeren.   
Dit wordt gedaan door event listeners. Deze objecten kunnen op allerlei verschillende events getriggerd worden. In deze objecten staat er dan ook wat er moet gebeuren eens het event is gebeurt.

Bij het maken van bijvoorbeeld een knop zullen de volgende stappen overlopen worden;

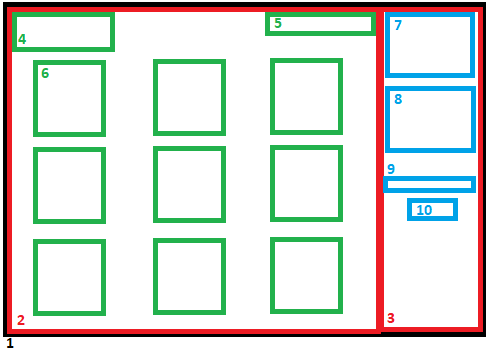
* De knop zelf wordt gemaakt en een naam, vorm, kleur, … gegeven.
* Een event listener wordt gemaakt voor een specifiek event, hier een muisklik.  
  Deze moet ook de resulterende actie krijgen.
* De event listener wordt aan de knop gekoppeld

Als dit gebeurt is, zal bij elke muisklik op de knop de resulterende actie worden uitgevoerd, zoals het starten van een spel.

## Werkwijze & Code

De GUI wordt gemaakt in de klasse GameHub. Deze zal alle objecten van de Swing-library maken en positioneren.   
GameHub zal via de control-facade de logica van de games zelf krijgen.

Layout  
De opdeling van de frame, panels en components zal via een illustratie duidelijk worden gemaakt.



1: Frame  
De overkoepelende frame waarin alle panels zitten. Zal standaard op volledig scherm geopend worden en een ‘default close operation’ krijgen die ervoor zorgt dat het programma normaal sluit bij het afsluiten.

2&3: Panels  
De frame wordt dan opgedeeld in twee panelen, ‘centerPanel’ (2) en rightPanel (3).   
RightPanel zal enkel zichtbaar worden bij het selecteren van een game. CenterPanel is altijd zichtbaar.   
Het positioneren van deze twee panelen werd met de simpele ‘BorderLayout’ gedaan.

4-6: Center Components  
In de code wordt de centerpanel nog eens verdeeld in twee delen, het ‘header’ en het ‘games’ deel.  
(Dit is gewoon de naam van de methodes, centerpanel wordt niet echt opgedeeld in twee delen).

Header bevat een JLabel ‘Title’ (4) en een JButton ‘ColorPicker’ (5). De titel toont het logo en met de ColorPicker kan men de achtergrondkleur aanpassen. Deze worden via het GridBagLayout op de respectievelijke plaatsen top-rechts en top gezet.

Games bevat 9 JLabels ‘pic[1-9]’ die elks een spel voorstellen. Ze bevatten een afbeelding van dit spel. Deze worden ook via het GridBagLayout op de juiste plaats gezet.

7-10: Right components  
Het rechterpaneel krijgt als layout manager een ‘box layout’ en bevat vier componenten onder elkaar gezet;  
Twee JText componenten ‘jtext’(7) en ‘jHighscores’(8), die voor het tonen van de tekst en de highscores van het geselecteerde spel zorgen.   
Hieronder zit een JSlider ‘slider’ (9) die de moeilijkheidsgraad door zal geven.   
tenslotte is er een JButton ‘start’ (10) die het spel zelf zal starten.

## Event listeners

Nu moeten aan bepaalde componenten event listeners gebonden worden.

5: aan de color picker zal een mouseEvent (klik) listener gebonden worden die een JColorChooser opent waarmee een nieuw achtergrondkleur geselecteerd kan worden.

6: Aan de games wordt er een mouseEvent (dus een klik) listener gebonden die voor dat spel de rightPanel zal maken.

9: Aan de instelling wordt een stateChangeEvent (dus als de waarde verandert) listener gebonden die de settings van dat spel kan aanpassen.

10: De startknop krijgt een actionEvent listener die iets complexer is als de rest. Deze zal in volgende paragraaf verduidelijkt worden

## Starten van een spel

Bij het starten van een spel zal er een processbuilder gemaakt worden. Deze zal het proces ‘game.jar’ uitvoeren met bepaalde parameters die in settings worden opgeslagen. De thread zal dan uitvoeren.

Er wordt dan gekeken naar de inputStream (dus de outputstream van het spel) om waardes zoals naam en score die in het spel worden bepaald door te geven aan gamehub. Deze zullen in het spel doorgegeven worden door System.out.println(“…”);  
Als het process eindigt, zullen deze waardes gebruikt worden om een nieuwe highscore te bepalen en, indien beter dan de top 10, opgeslagen te worden in de highscore-documenten.   
Omdat de highscores misschien aangepast zijn, moet het rechterpaneel terug geupdate worden.

## 