# Mastermind

Een globale omschrijving van de werking van het programma:

Programmeer het spel Mastermind tegen de computer, daarbij kun je gebruik maakt van Object Oriented Programming.

Met mastermind bedoelen we dit spel: https://nl.wikipedia.org/wiki/Mastermind

Een uitgebreide uitleg vind je hier: https://nl.wikihow.com/Mastermind-spelen

Hieronder staat het spelverloop uitgelegd.

**Spelverloop**

De computer kiest random vier letters (kleuren) uit de verzameling {a, b, c, d, e, f}.

De gekozen rij wordt de "code" genoemd.

De volgorde is van belang; een letter mag vaker voorkomen.

De gebruiker moet de verborgen code proberen te achterhalen.

De gebruiker geeft een code van vier letters op.

De computer geeft een reactie op de ingegeven code, door te antwoorden met:

-> het aantal correcte letters die op de juiste plaats staan (mbv het teken +)

-> het aantal correcte letters dat NIET op de juiste plaats staat (met behulp van het teken ?)

De gebruiker geeft nu een nieuwe code op, gebaseerd op deze nieuwe informatie.

Als alle vier letters op de juiste plaats staan, is de code gekraakt en het spel ten einde.

Een lopend spel kan worden beëindigd door het invoeren van een q; alle andere invoer moet ofwel correct zijn (dus in de verzameling voorkomen), ofwel resulteren in opnieuw bevragen van de gebruiker.

Tref onderstaand stap voor stap de functionaliteiten staan van beginnend tot moeilijk.

**Doelstellingen een**

Maak met behulp van een loop een programma aan die iedere keer om input van de gebruiker vraagt.

De invoer van de gebruiker wordt op het scherm getoond.

Wanneer de gebruiker een q invoert, dan stopt het spel.

**Doelstelling twee**

Het spel weet nu hardcoded een code van 4 letters.

De gebruiker geeft een code in.

Als de eerste letter overeenkomt met de code dan toont de computer: "goed gedaan"

Als de eerste letter niet overeenkomt met de code dan toont de computer: "helaas"

Iedere ronde opnieuw.

**Doelstelling drie**

Het spel verzint iedere keer een nieuwe random code.

De functionaliteit uit doelstelling twee werkt nog steeds.

Als de eerste letter goed is geraden, dan toont de computer de code en is het spel afgelopen.

**Doelstelling vier**

De computer controleert nu heel de code.

Er staan nu evenveel +jes als dat er kleuren op de goede plek staan.

Als er 4 +jes op de goede plek staan dan toont de computer de code en is het spel afgelopen.

Ook staat de computer maximaal 12 pogingen toe. Indien je dan nog niet klaar bent toont de computer dat je af bent.

**Doelstelling vijf**

De computer kan nu ook bepalen of een kleur wel voorkomt maar nog niet op de goede plaats staat.

Dit geeft hij aan met een ?

**Doelstelling zes**

Je kunt het nu met twee (of meer) personen spelen. Iedere speler kan zijn naam invullen.

Er wordt bijgehouden welke speler in hoeveel pogingen de code kraakte.

Degene met de laagste punten wordt tot winnaar uitgeroepen.

**Doelstelling zeven**

Er wordt een geschiedenis bijgehouden.

Deze geschiedenis kan naar een file geschreven worden en wanneer het spel opgestart wordt weer geladen.

# Kermis

Je gaat een programma maken dat de omzet van een Kermis bijhoudt.

De Kermis heeft een zestal attracties, die zijn allemaal subclasse van een klasse Attractie.

Zodra het programma start kan er een getal worden ingegeven,

1 voor botsauto's,

2 voor spin,

3 voor spiegelpaleis,

4 voor spookhuis,

5 voor hawaii,

6 voor ladderklimmen.

**Doelstelling 1:**

Zorg dat het bovenstaande stabiel werkt en dat iedere keer dat het getal wordt ingegeven er getoond wordt welke attractie draait.

Aangenomen is dat je dit realiseert door een methode van een object aan te roepen, dus: attractie.draaien(); ipv System.out.println("de attractie draait"); In de methode draaien van de attractie mag System.out.println("de attractie '' draait"); natuurlijk wel gebruikt worden.

Iedere attractie heeft een naam, een prijs en een oppervlakte.

**Doelstelling 2:**

Botsauto's kosten 2.50,

Spin 2.25,

Spiegelpaleis 2.75,

Spookhuis 3.20,

Hawaii 2.90 en

Ladderklimmen 5.00.

Houdt in een centraal object Kassa bij wat de omzet is van de hele kermis.

Houdt in iedere attractie bij hoeveel omzet elke attractie draait.

Ook willen we weten hoe vaak er kaartjes per attractie zijn gekocht, en natuurlijk voor de attracties in het totaal.

Met de invoer van de letter "o" zien we de omzet. Met de invoer van de letter "k" zien we het aantal kaartjes.

**Doelstelling 3:**

Spin en Hawaii zijn zgn RisicoRijkeAttracties.

RisicoRijkeAttracties moeten een keuring ondergaan nadat ze opgesteld zijn. Deze methode heet dus opstellingsKeuring().

RisicoRijkeAttracties hebben een draaiLimiet. Dit is het aantal maal dat ze mogen draaien voordat er een onderhoudsbeurt moet plaatsvinden. Voor de Spin is dit 5 en voor de Hawaii is dit 10. In deze stap mag de onderhoudsbeurt nog handmatig worden aangeroepen.

LadderKlimmen is een GokAttractie.

GokAttracties moeten kansSpelBelastingBetalen(). De kansSpelBelasting is het reserveren van 30% over je omzet.

Implementeer de RisicoRijkeAttracties met behulp van Abstracte Klassen.

Implementeer de GokAttractie via een interface.

**Doelstelling 4:**

Maak een Klasse BelastingInspecteur.

Maak OF een input voor een bezoek van de BelastingInspecteur, bv 'b'.

OF maak een random die 1 keer per 15 verkopen ervoor zorgt dat de BelastingInspecteur langskomt.

De belastingInspecteur haalt meteen zijn belasting uit de omzet.

De kassa moet natuurlijk bijhouden dat er een bezoek van de BelastingInspecteur is geweest.

Een belasting inspecteur checkt van alle attracties of die van het Type GokAttractie zijn.

Alleen voor Gokattracties wordt belasting geheven, de andere niet.

Tenslotte maken we de Spin ook een GokAttractie om te kijken of ons Ontwerp goed werkt.

**Doelstelling 5:**

We gaan onze Kermis applicatie een stuk robuuster maken.

In plaats van de de applicatie-bediener zelf moet bijhouden of de Spin of Hawaii al over zijn 5 of 10 ritjes zit, gaat de applicatie dit bijhouden.

Op het moment dat er een 6e of 11e kaartje wordt verkocht, throwt de Attractie een Exception.

De applicatieBediener kan een monteur aanroepen mbv van de knop 'm'. Die doet zijn keuring en reset de attractie.

# Eigen programma

Dit keer stel ik geheel geen functionele eisen vast. Ik zal alleen verzoeken om .... technische eisen.

De technische eisen zijn:

1. Je moet minstens 7 klassen hebben. 1 daarvan moet abstract zijn. Elke klasse moet zijn eigen verantwoordelijkheid hebben.
2. Er moeten in je programma minstens 2 interfaces zijn.
3. 80 % van de methoden moet een ander returntype hebben dan void.
4. 40 % van de parameters moet van een KlasseType zijn (Strings worden niet meegeteld).
5. Minstens 7 keer zal de interface of het returntype of het parametertype moeten zijn.
6. Er zal zelf een Exception-klasse gemaakt moeten worden.
7. Een Exception zal minstens 3 keer gethrowd moeten worden in je programma.
8. In je programma moet een constructor een keer worden overload.
9. In je programma moet een constructor een keer worden geredirect.
10. Je zult minstens 4 methoden moeten overriden
11. Minstens 1 overridden methode moet een Exception throwsen.
12. 1 constructor moet een exception throwen.
13. Gebruik alleen native java, en het moet een console applicatie zijn

# BlackJack

**Uitleg BlackJack**

De Black Jack wordt in de commandline gemaakt.

Het zal een werkend spel moeten zijn bij oplevering.

Doordat het een commandline spel is mag je natuurlijk creatief zijn in de interactie met de gebruiker. H4 kan een harten 4 zijn. Zolang de onderliggende techniek maar werkt.

Op de wikipedia pagina zie je dat er heel veel regels, uitbreidingen en varianten zijn voor een compleet BlackJack spel.

De kunst is als programmeur om deze op te knippen en met kleine doelstellingen te beginnen.

Dat werk heb ik al een beetje voor je gedaan. Hou daarom als leidraad de volgende doelstellingen aan:

**Doelstelling 1**

Elke Black Jack begint met het tonen van de geschudde kaarten.

De Black Jack spelen we met 1 spel van 52 kaarten. Alle kaarten zitten dus 1 keer in het overzicht, en als het spel opnieuw gestart wordt is de volgorde in een random nieuwe volgorde.

**Doelstelling 2**

Een speler kan meedoen met het spel. De speler kan door een 'k' in te voeren om een kaart vragen. Een speler kan altijd met heel het spel stoppen door 'q' in te voeren. De speler mag ook 'p' invoeren om te passen.

Als een speler een kaart vraagt krijgt hij de bovenste kaart van de stapel. Het totaal aantal punten wordt getoont.

De speler mag met 0 kaarten starten. De aas mag als een fixed 11 punten kaart beschouwd worden.

Als de speler past, worden alle kaarten getoont die hij heeft met het totaal aantal punten van de kaarten.

De punten per kaart zijn: De waarde van het getal bij de 2, 3, 4, 5, 6, 7, 8, 9, 10. De Boer, Vrouw, Heer zijn 10 punten. De aas is zoals gezegd 11 punten.

**Doelstelling 3**

Vanaf hier wordt het spannender. Het is vanaf deze doelstelling verplicht om met kaarten als objecten te werken. Kaart kaart = new Kaart(). De vorige doelstellingen konden prima op andere wijzen volbracht worden, vanaf hier is Object Georiënteerd programmeren een vereiste.

De speler krijgt twee kaarten toebedeeld of je wil je wanneer hij start, zoals bij de werkelijke regels. Ook kan de Aas 1 of 11 punten waard zijn. Dat is afhankelijk van het gegeven of hij anders boven de 21 punten uit komt.

Als een speler niet op tijd past, is de speler af.

Er mag geen invoer zijn die het programma doet vastlopen.

Mocht je tijd over hebben dan kun je je bezig gaan houden met meerdere spelers, een bank of zelfs gokken en inzetten.