Inhoudsopgave

CoderDojo

Configuratie

I'm learning: Java

1 Met deze Sushi kaarten ga je een simpel spel maken met één van de meest populaire programmeertalen ter wereld: Java. Oorspronkelijk waren zowel Minecraft als Gmail in Java geschreven. Java is de programmeertaal die veel programmeurs als eerst hebben geleerd.

2 Het spel dat je gaat maken is iets dat je al kent: je gooit een "munt" op en moet raden op welke kant de munt landt.

3 Vaak moet je van alles en nog wat downloaden om met Java te kunnen programmeren, maar we gebruiken nu een online tool voor deze kaarten. Ga naar https://repl.it/signup.

4 Deze tool heet repl.it. Je moet een account maken. Let op: de website is in het Engels dus roep de hulp van een mentor in als je het lastig vindt.

5 Klik op **myrepls** en maak een nieuw project aan voor `Java`.

6 In het scherm zie je links het gedeelte waar je de code schrijft en rechts waar je het uitgevoerd (geprint) ziet worden.

1 Elke Java programmeur moet met een speciale code beginnen. Laten we ze maar even "toverspreuk" noemen. Later leggen we ze aan je uit. Type dit:

```
class Main {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

2 Klik op de **Run** knop om je code uit te voeren en kijk wat er rechts op je scherm gebeurt!

3 Dit is je eerste Java programma! Veel van die code is de toverspreuk, maar het stukje binnen de accolades ({}) is wat het laat werken. Dit heet de **main method** (= hoofdmethode). De code daarbinnen vertelt aan Java om op het scherm te printen wat er binnen de dubbele aanhalingstekens ("") staat. Dit is hoe het spel met de speler zal gaan praten.

```
System.out.println("Hello world!");
```

4 Een paar dingen over deze coderegel: de manier waarop Repl.it de code in verschillende kleuren zet (de tekstboodschap en het Java commando). Haal de tekst weg (zet het ook weer in de code), zie je hoe de kleuren veranderen?
En: de regel eindigt op ;. Dan weet Java dat dit het eind van de regel is. Type je de puntkomma niet, dan raakt Java in de war.

5 Natuurlijk is "Hallo wereld!" de klassieke manier om je eerste computerprogramma te schrijven. In het spel heb je het niet nodig. Verander de tekst naar "Raadt kop (k) of munt (m)!" en voer de code weer uit. Nu kun je de speler laten raden!

```
System.out.println("Raadt kop (k) of munt (m)!");
```

1 Dus je vraagt de speler om te raden. Maar hoe kun je tegen Java zeggen dat het moet luisteren naar het antwoord? Je zult daarvoor meer code moeten schrijven. Eerst moet je iets gebruiken dat `Scanner` heet en wat je moet `importeren`. Code importeren is de manier om Java te vertellen dat het méér dan alleen de simpele dingen moet gebruiken. Dus we gaan het erin zetten door deze code aan het begin te zetten:

```
import java.util.Scanner;
```

2 Nu moet je een `Scanner` maken binnen je **main method** en het een label geven zodat je het later kunt gebruiken. Ik gebruik `user_input` als mijn label aangezien ik de scanner ga gebruiken om de input van de user (= het antwoord van de speler) te krijgen! Voeg de volgende regel toe aan het begin van je **main method**:

```
Scanner user_input = new Scanner(System.in);
```

Dit vertelt Java om een label voor `Scanner` genaamd `user_input` te maken en het koppelt aan een `new Scanner` die kijkt naar `System.in`, en dat is de plek waar hetgeen de speler typt in terecht komt! Dit proces van een `Scanner` creëren en het een label geven heet **declaring** (= verklaren). Je zult een heleboel dingen verklaren terwijl je dit spel maakt!

3 Nu moet je zorgen dat de Scanner het antwoord van de speler opvangt en opslaat. Na de printregel, waar je de speler vraagt om te raden, voeg je de volgende code toe:

```
String user_guess = user_input.next();
```

Dit **verklaart** een nieuwe `String` genaamd `user_guess`, *wat een stukje tekst is. Het vertelt de* `user_input` om de tekst die de speler typt op te vangen en op te slaan in die `String`.

4 Nu gaan we kijken of we het antwoord van de speler krijgen. Voeg een nieuwe regel toe, meteen na de vorige, die print wat de speler getypt heeft.

```
System.out.println("Je hebt geraden:"+user_guess);
```

Zie je hoe je twee `Strings` samen kunt voegen met behulp van een +? Voer nu je code uit en raadt. Wat gebeurt er?

1 Nu moet je echt een muntje opgooien! Dit lijkt veel op de vorige kaart: je moet dingen importeren en verklaren. Je "muntje" is een willekeurig getal generator - een stukje code dat al bestaat in Java: het kiest getallen met een gelijke kans om een getal tussen 0 en een hoger getal te kiezen. Natuurlijk kies je nu 0 en 1 omdat een munt maar twee kanten heeft. Begin met de `import` regel, net na je vorige `import` regel:

```
import java.util.Random;
```

2 Nu moet je een `Random` (= willekeurig) maken die je kunt gebruiken, net als bij `Scanner`. Voeg deze regel toe aan het begin van de **main method**:

```
Random coin = new Random();
```

3 Nu moet je de munt echt opgooien en het resultaat ergens opslaan. Net als bij de **Scanner** schrijf je beide dingen in één regel, zoals dit:

```
int toss = coin.nextInt(2);
```

Wat valt hieraan op:

* Het getal dat je aan Java gaf is 2. Java gebruikt het lagere getal (0) maar niet het hogere getal (2) als het een getal kiest, dus de getallen tussen 0 en 2 zijn 0 en 1.

* Terwijl je het resultaat van de `Scanner` in een `String` stopt, sla je het resultaat van deze `Random` op in een `int`. `Int` is een afkorting voor **integer**. Een integer is een heel getal, dus deze Random zal je nooit decimalen geven. je kunt er wel één maken die decimalen geeft!

4 Print nu het resultaat om te kijken of je code werkt. Voer het programma een paar keer uit om er zeker van te zijn dat het getal willekeurig is, en haal dan de regel weg die het resultaat print. Je voegt straks weer iets toe, want je wilt dat er komt te staan "kop" of "munt" en niet "0" of "1"!

1 Nu heb je een willekeurige munt, en die landt óf op kop (1) óf op munt (0) elke keer als het programma uitgevoerd wordt. Om het programma aan de speler te vertellen wat de uitkomst is, moet je de getallen omzetten in woorden. Je kunt dit doen met **conditional** (= voorwaardelijke) of **if** (= als) bepalingen.

Zoiets als "*als* de muntwaarde 1 is, print dan 'kop', print *anders* 'munt'".

2 In Java heeft een **if** bepaling een set vierkante haken waarin je de **condition** (= voorwaarde) zet. Java leest de **condition** en bepaalt of het **true** (= waar) of **false** (= niet waar) is. Als het **true** is, voert het de code binnen de accolades {} uit. Als het **false** is, slaat het deze code over en voert uit wat erachter staat.

3 Om te beginnen zorg je ervoor dat de code print dat de munt op kop is gevallen als je een 1 krijgt, of munt als je een 0 krijgt. Daarvoor voeg je deze code toe na `int toss` :

```
if (toss == 1) {
  System.out.println("De munt is op kop geland");
}
if (toss == 0) {
  System.out.println("De munt is op munt geland");
}
```

Zie je dat je twee == tekens moet gebruiken om na te kijken of de waarden aan weerszijden van de == tekens hetzelfde zijn?

4 Nu moet je het antwoord van de speler in een getal omzetten zodat je het kunt vergelijken met toss. Je gebruikt een **if statement** om na te kijken **if** het antwoord van de speler *k* was en stelt een nieuwe **variabele** in, een `int` genaamd `guess_number` naar **1** als het zo is.

Doe hetzelfde voor *m* en *0*. Voeg na je laatste `System.out.println` regel deze code toe:

```
int guess_number;
if (user_guess.equals("k")) {
  guess_number = 1;
}
if (user_guess.equals("m")) {
  guess_number = 0;
}
```

Zie je dat je, om **Strings** te vergelijken, `name.equals(thing_to_compare_to)` moet gebruiken? Het ding binnen de vierkante haakjes kan het label zijn voor een andere **String** of het kan gewoon een tekst**String** zijn, zoals hierboven.

Zie je ook dat `guess_number` gemaakt is zonder het een waarde te geven? Terwijl je tegelijkertijd waarden maakt en geeft aan **variabelen**, kun je dat ook afzonderlijk doen en dat is zinvol als een waarde afhangt van een andere waarde.

Tenslotte, zie je dat je `int` alleen hoeft te gebruiken als je de **variabele** maakt?

5 Oké, nu gaan we de speler vertellen of hij/zij goed geraden heeft. Kijk het antwoord na met de toss (= worp) en vertel ze of ze gewonnen of verloren hebben! Let op: **not equal** (= niet gelijk) is != in Java.

```
}
if (toss ==guess_number) {
  System.out.println("Je hebt juist geraden!");
}
if (toss != guess_number) {
  System.out.println("Je hebt verkeerd geraden! Game over!");
}
```

Speel nu je spel! Op de volgende kaart zie je hoe je kunt blijven spelen zolang je juist raadt.

1 Eén van de dingen die een computer heel krachtig maakt is dat je een code continu kunt laten uitvoeren. Voor eeuwig als je dat zou willen! Dit wordt gedaan met loops (= lussen), wat stukjes code zijn die uitgevoerd blijven worden tot een bepaalde voorwaarde niet waar is (als het nooit waar is, zal de code nooit uitgevoerd worden!).

2 Je gaat een **while** (= terwijl) loop gebruiken, welke dezelfde taak steeds blijft uitvoeren, **while** zo'n voorwaarde waar is. In dit geval: *terwijl* (zo lang) het laatste antwoord van de speler juist was. Je zult een **true** of **false** waarde op moeten slaan dat je vertelt of dat het geval is. Dit heten **boolean** (= booleaans; er zijn slechts twee waarden mogelijk) waarden, en worden opgeslagen in een **variabele** met dezelfde naam. Dus voeg een variabele code toe bovenin je **main method** (waar je alle variabelen zet):

```
boolean user_right = true;
```

De eerste keer moet **true** zijn anders zal de **loop** die je maakt nooit uitgevoerd worden!

3 Zet nu, vlak onder de variabelen in je **main method**, je **while** lus:

```
while (user_right){
}
```

4 Selecteer alle code onder de loop en sleep het *in* de code. Voer het eens uit. Het zou nu continu door moeten gaan, of de speler wint of verliest (aangezien je `user_right` niet op false hebt gezet!) en je zult op de stopknop moeten klikken om te stoppen.

5 Om uit de loop te komen en het programma te stoppen is makkelijk. Je hoeft alleen een regel aan het **if statement** toe te voegen dat ermee om kan gaan als de speler verkeer raadt:

```
while (toss != guess_number) {
  System.out.println("Je hebt verkeerd geraden! Game over!");
  user_right = false;
```

6 Dat was het! Je hebt nu een volledig spel! Voer het uit, test het en probeer te winnen! Er is van alles wat je kunt doen als speler dat dit spel niet aankan. Hoe je daarmee om moet gaan leer je in andere Sushi kaarten.