Oefening 3: Weerstanden (vervolg)

3.1 Abstracte klassen

Implementeer een klasse *ParallelSchakeling*. Het is de bedoeling dat deze klasse dezelfde functionaliteit aanbiedt als de klasse *SerieSchakeling*. Geef deze twee soorten schakeling een gemeenschappelijke superklasse *Schakeling*, waarin je hun gemeenschappelijke functionaliteit onderbrengt. Specifiek voor een parallelschakeling is de methode *vervangingsweerstand* die natuurlijk op de correcte manier moet berekend worden.

Laten we nu eens nadenken over de methode *spanning*. In principe zouden we ook hier voor de twee soorten schakelingen dezelfde implementatie kunnen gebruiken, die gewoon het product berekent van de stroom en de vervangingsweerstand. (Op analoge manier als de methode *stroom*). We zouden, met andere woorden, deze methode ook op het niveau van Schakeling kunnen definiëren. Als je dit probeert, zal je waarschijnlijk merken dat de compiler hier niet tevreden over is (net zoals voor de methode *stroom*). Los dit probleem op door van *Schakeling* een **abstracte klasse** te maken met een **abstracte methode**.

Test je klasse uit door een parallelschakeling van 3 weerstanden te maken, die bestaat uit een weerstand van 2Ω, een weerstand van 3Ω en een regelbare weerstand met bereik van 5Ω tot 10Ω. Bereken eens de spanning die bij een stroom van 1A hoort, en kijk daarna hoe deze evolueert als je de waarde van de regelbare weerstand met een percentage van 25% verhoogt. (Voor deze verhoging ga je gebruik moeten maken van het concept **typecasting**) .

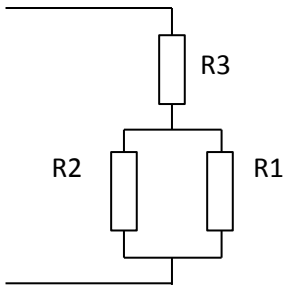
3.2 Algemeen – algemener – algemeenst

Tot dusver hebben we enkel maar serie- en parallelschakelingen beschouwd die zijn opgebouwd uit individuele weerstanden. Veralgemeen dit door toe te staan dat de componenten van een serie- of parallelschakeling op hun beurt opnieuw ook serie- of parallelschakelingen mogen zijn. Je zal hiervoor een nieuwe klasse *Component* in het leven moeten roepen, die een abstracte superklasse is van zowel *Schakeling* als *Weerstand*.

Je hebt nu al een aantal klassen, dus om het overzicht te behouden, is het nuttig om eens een klassediagramma te tekenen.

Overloop nu nog eens al je code en controleer of alle methoden op de juiste plaats in de hiërarchie staan – verplaats methoden die beter ergens anders zouden staan. In het algemeen staat alles best zo hoog mogelijk!

Test tot slot je implementatie eens uit door de stroom doorheen volgende schakeling te berekenen bij een spanning van 5V. Hier is R1= 3Ω,R2= 2Ω en R3= 5Ω.



3.3 Uitbreiding

Weerstanden zijn te herkennen aan een kleurencode, die bestaat uit vier gekleurde banden, waarvan de eerste drie de verhoopte weerstand van deze component aangeven. Dit systeem wordt uitgelegd op: http://en.wikipedia.org/wiki/Electronic_color_code.

Voeg een constructor toe aan de klasse *Weerstand* die als argument drie kleuren neemt en een corresponderende weerstand creëert. Om dit te kunnen doen, moet de betekenis van de kleurencodes ergens in je programma vervat zitten. Er zijn verschillende manieren om dit objectgericht klaar te spelen. Deze opgave onderzoekt er hier één van.

Om onnodig typwerk te besparen, gaan we enkel de kleuren zwart (0), bruin (1) en rood (2) gebruiken. Maak een abstracte klasse *GekleurdeBand* met subklassen *ZwarteBand, BruineBand en RodeBand*. Deze klassen bieden een publieke methode *getalWaarde* aan, die het overeenkomstig getal teruggeeft.

De klasse *Weerstand* krijgt dan een constructor met drie *GekleurdeBand*-en als argument.

Geef je drie kleurklassen ook een toString()-methode.

Maak een main-methode die een weerstand van 12Ω aanmaakt dmv. de kleurencode bruin-rood-zwart.

In een groter software-systeem zal de informatie over kleuren van de weerstanden waarschijnlijk oorspronkelijk binnenkomen als String. Voorzie een statische methode *zetStringOm* in de klasse GekleurdeBand , die een string “zwart”, “bruin” of “rood” omzet in een object van de juiste klasse.

Geef je klasse Weerstand nu een extra constructor die als argument drie Strings neemt, en gebruik deze in je main-methode