

## Leerdoelen

Na het maken van deze opgave moet je

- kunnen werken met enumeratietypen en **switch**-statements in Java;
- kunnen werken met encapsulatie;
- interfaces kunnen implementeren.

## Een plattegrond voor de loipe in Sochi

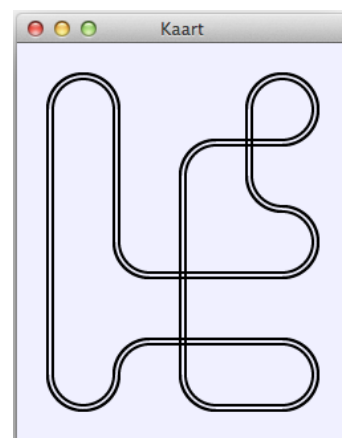
Bij het langlaufen volgt de sporter op skies een parcours van twee smalle sporen in de sneeuw. Zo'n parcours heet een loipe. Zo'n loipe is iedere wedstrijd anders. We weten alleen dat het parcours altijd door een regelmatig rooster van vierkanten gaat. Het parcours komt via het midden van een van de zijkanten zo'n vierkant binnen en zal het via het midden van een van de andere drie zijkanten het vierkant weer verlaten. Het parcours kan zichzelf wel loodrecht kruisen, maar zal zich niet splitsen of samenvloeien.

Om op de loipe zo snel mogelijk af te leggen is een goede planning natuurlijk belangrijk. Een eerste vereiste hiervoor is overzicht van het parcours. Helaas belemmert mist en zware sneeuwval in de bergen het zicht op het parcours. Een skier kan het parcours voor de wedstrijd wel verkennen. Door te onthouden hoe het parcours in de verschillende vierkanten verloopt kan de coach een goed overzicht krijgen. De skier schrijft per vierkant op of zij rechtdoor (s, **S**traight), linksaf (l, **L**eft), of rechtsaf (r, **R**ight) gaat. Een skier start altijd aan de rand van een vierkant met de skies richting het noorden.

Na het verkennen van het parcours door de skier heeft men een representatie van het parcours als `"srrsslslrrsrrrslssslslslrrsss"`. Deze representatie van het parcours is natuurlijk niet erg inzichtelijk. Een plaatje zo als hiernaast geeft een veel beter inzicht. Je gaat de reeks letters op twee verschillende manieren om te zetten in een tekening. De eerste keer teken je het parcours met ASCII-art, door middel van gewone characters wordt de loipe zo goed mogelijk getekend. De tweede manier van tekenen gebruikt gegeven Java-code die het parcours kan tekenen zo als hierboven.

Dit plaatje laat zien dat een loipe een rondgaand pad kan zijn, maar dat hoeft natuurlijk niet zo te zijn.

Om inzicht te hebben over de vorderingen van de sporter kan men ook de huidige positie van de sporter op het parcours grafisch weergeven.



## 1 Fragmenten van de loipe

Maak een enumeratietype `Fragment` met de constanten `NZ`, `OW`, `NO`, `NW`, `ZO`, `ZW`, `KR` om de richting van de fragmenten van de loipe aan te geven. Hierin is `KR` een kruising. Voor de overige fragmenten geeft de naam aan hoe het parcours op dat veld loopt, zo is `NZ` een (rechte) Noord-Zuid verbinding en `NW` een bocht van het Noorden naar het Westen (of omgekeerd natuurlijk). Enumeratietypen en **switch**-statements ken je al uit Programmeren, in Java werken enumeratietypen bijna net zo als in C++. Zie bijvoorbeeld de sheets van de colleges.

## 2 De Loipe

Maak een klasse `Loipe` om een overzicht van de loipe te maken. De constructor `Loipe (String pad)` maakt een overzicht van de loipe door het pad om te zetten in een tweedimensionale rij van fragmenten.

Volgens goed gebruik in de informatica gebruiken we `loipe [0][0]` als de linker *bovenhoek* van de plattegrond. De  $y$ -waarden worden *groter* als we naar beneden gaan. Dit is dus precies omgekeerd als in de wiskunde.

## 2.1 Afmetingen van de loipe

Om de tweedimensionale rij van fragmenten te kunnen maken moeten we de afmetingen van de loipe weten. We kunnen deze uit de rij van stappen afleiden door (bijvoorbeeld) op coördinaten  $(0,0)$  te beginnen en voor iedere stap de coördinaat  $(x,y)$  te berekenen. Merk op dat `l` en `r` niet draaien op de plaats zijn, de skier gaat hier een vierkant binnen en gaat er via een van de aanliggende zijden weer uit. Met een extra enumeratietype voor de huidige *richting* en wat **switch**-statements kan dat helder en compact geformuleerd worden.

Het verschil tussen de minimale en maximale waarde van  $x$  geeft de breedte van de te maken kaart, en het verschil tussen de minimale en maximale waarde van  $y$  geeft de hoogte. We noemen deze waarde respectievelijk  $x_{min}$ ,  $x_{max}$ ,  $y_{min}$ , en  $y_{max}$ .

## 2.2 Vullen van de loipe

We weten nu de afmetingen van de kaart met fragmenten van de loipe en het beginpunt. We kunnen nu de juiste fragmenten in de loipe plaatsen door het pad opnieuw af te lopen. Het is heel goed mogelijk dat de loipe ten oosten of ten zuiden van het startpunt komt. Door het pad op de kaart niet te beginnen op  $(0,0)$ , maar op  $(-x_{min}, -y_{min})$  zorgen we dat het pad toch niet van de kaart afloopt.

Als de loipe nog leeg is kunnen we direct het fragment plaatsen dat correspondeert met de beweging van de skier. Als de loipe al gevuld is moet dit een kruisend fragment zijn. We veranderen dit fragment in `KR`. Dit vullen van de loipe vergt behoorlijk wat gevalsonderscheid, gebruik de switch-statements goed en introduceer zo nodig wat hulpfuncties. Merk op dat fragmenten van de loipe die niet op het pad liggen de waarde **null** houden.

## 2.3 Positie van de Sporter

De methode `Punt start()` geeft het startpunt van de sporter op de kaart aan, dat wil zeggen  $(-x_{min}, -x_{max})$ . Met de methode `Punt stap()` krijgen we het volgende punt op de loipe.

Je mag kiezen of je aan het eind van het parcours weer vooraan begint, of aangeeft dat er geen volgend punt meer is.

De klasse `Punt` bevat minstens de  $x$  en  $y$  waarde van de een punt op de kaart en bijbehorende getters. Op Blackboard staat een basis voor deze klasse. Je mag dit zelf zo veel groter maken als nodig of handig.

## 2.4 Interface van de Loipe

Zorg dat de loipe ten behoeve van het tekenen de volgende interface implementeert.

```
public interface InfoLoipe {
    public int getWidth();           // grootte in oost-west richting
    public int getHeigth();          // grootte in noord-zuid richting
    public Fragment getFragment(int x, int y); // fragment van de loipe op positie (x,y)

    public Punt start();              // Het startpunt op de kaart
    public Punt stap();               // het volgende punt op de route
}
```

Op deze manier weten de teken klassen wat ze van de loipe kunnen gebruiken, zonder daar alle details over te hoeven weten. Ook de route over de kaart is zo eenvoudig te volgen.

# 3 Het TekenLoipe Interface

Omdat we in deze opgave op twee verschillende manieren willen gaan tekenen maken we een Java interface dat deze teken-klassen moeten gaan aanbieden:

```
public interface TekenLoipe {
    public void teken();           // teken de gegeven loipe
    public void setPosition(Punt p); // teken de sporter op de gegeven positie
}
```

We kunnen dan op een uniforme manier beiden manieren van teken voor het parcours gebruiken.

#### 4 Het parcours tekenen in letters (AsciiArt)

Maak een klasse `AsciiArt` die het parcours als een aantal regels met letters en symbolen kan afdrukken. Gebruik bijvoorbeeld een `-` voor een oost-west verbinding, een `|` voor een noord-zuid verbinding, en een `+` voor een kruising. Een simpel parcours beschreven door `"srssrsslsllsss"` kan er uit zien als:

Deze klasse dien natuurlijk het interface `TekenLoipe` te implementeren. Via het `InfoLoipe` interface kan `AsciiArt` de informatie uit de `loipe` halen.

## 5 Het volgen van de route

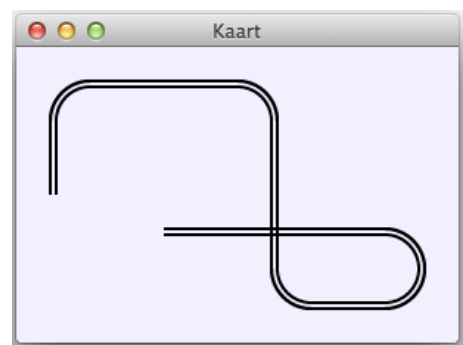
Voor het volgen de route gebruiken we `start` en `stop` uit het `InfoLoipe` interface. Met `setPosition` kan de huidige positie getekend worden in `ascii-art` getekend worden, hier als een `*`.

$$\begin{array}{ccccccc} / & - & . & & / & - & . \\ | & * & / & - & + & - & / \\ | & | & | & & \backslash & - & . \\ | & \backslash & - & + & - & - & / \\ | & / & - & + & - & - & . \\ \backslash & - & / & & \backslash & - & - & / \end{array}$$

Maak een interactief programma dat bij iedere input een stap verder gaat.

## 6 Het parcours als tekening

Op de Bb vind je de klasse `LoipePlaatje` waarmee je het parcours als plaatje kunt tonen. Een `LoipePlaatje`-object zal bij het aanroepen van de `teken`-methode een windowtje openen met daarin een plaatje van de loipe. Om te kunnen tekenen gebruikt de klasse `LoipePlaatje` plaatjes die op blackboard gegeven zijn als png-file. In een NetBeans project worden deze plaatjes verwacht in de `src`-folder van het project. We zullen later in deze cursus leren hoe we zo'n `LoipePlaatje` klasse moeten maken. Als alles goed is moet een programma dat correct werkt met ascii-art ook met deze plaatjes correct werken.



## Inleveren

**Vóór zondag 1 maart, 11:00 uur, via Blackboard.** Lever *alle* .java files uit de source folder van je project in. Neem een pad en de bijbehorende AsciiArt uitvoer als commentaar op in je hoofdklasse.