252-0027-00: Einführung in die Programmierung

Übungsblatt 11

Abgabe: 12. Dezember 2023, 23:59

Checken Sie mit Eclipse wie bisher die neue Übungsvorlage aus. Importieren Sie beide Eclipse-Projekte (das Projekt für den Bonus und das Projekt für die restlichen Aufgaben).

Aufgabe 1: Maps

1. Implementieren Sie eine Methode `U11Map.arrayToMap(String[] A)`, die einen Array `A` von Strings als Parameter akzeptiert und eine Map `M` von String zu Integer zurückgibt. Jeder String, der in `A` auftritt, soll in `M` auf die Zahl 0 abgebildet werden. Sie können davon ausgehen, dass `A` nicht `null` ist und dass kein String der empty (leere) String ist.

   Zum Beispiel gibt `arrayToMap` für den Array `[one, two, three, one]` die Map `{one=0, two=0, three=0}` zurück.

2. Implementieren Sie eine Methode `U11Map.arrayToMapOne(String[] A)`, die einen Array `A` von Strings als Parameter akzeptiert und eine Map `M` von String zu Integer zurückgibt. Jeder String, der in `A` auftritt, soll in `M`, falls der String nur einmal vorkommt, auf die Zahl 0 abgebildet werden und, falls der String mehrfach vorkommt, auf die Zahl 1 abgebildet werden. Sie können davon ausgehen, dass `A` nicht `null` ist und dass kein String der empty (leere) String ist.

   Zum Beispiel gibt `arrayToMapOne` für `[one, two, three, one]` die Map `{one=1, two=0, three=0}` zurück und für `[one, two, three, one, one, four, two]` die Map `{four=0, one=1, two=1, three=0}`.

Aufgabe 2: Expression Evaluator

In dieser und in folgenden Übungen werden Sie eine Reihe von Programmen schreiben, welche andere Programme interpretieren, kompilieren oder (in kompilierter Form) ausführen. Die Programmiersprachen definieren wir selber.

Als Einstieg schreiben Sie ein Programm, welches mathematische Ausdrücke (*expressions*) auswertet. Die Ausdrücke bestehen aus Zahlen, Variablen, Operatoren wie $+$ oder $-$ und einfachen Funktionen wie $\sin()$ oder $\cos()$. Die genaue Syntax für diese Ausdrücke finden Sie als EBNF-Beschreibung in Abbildung 1.

digit ⇐ `0` | `1` | ... | `9`
char ⇐ `A` | `B` | ... | `Z` | `a` | `b` | ... | `z`
num ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]
var ⇐ *char* { *char* }
func ⇐ *char* { *char* } `(`
op ⇐ `+` | `-` | `*` | `/` | `^`
open ⇐ `(`
close ⇐ `)`

atom ⇐ *num* | *var*
term ⇐ *open expr close* | *func expr close* | *atom*
expr ⇐ *term* [*op term*]

Abbildung 1: **EBNF-Beschreibung** von *expr*

Ein Programm, das Ausdrücke auswertet, muss natürlich entscheiden, ob eine gegebene Zeichenkette überhaupt ein gültiger Ausdruck ist[1]. Das nennt man *parsen* und ein solches Programm heisst *Parser*. Aus einer EBNF-Beschreibung wie dieser kann man einfach einen Parser erstellen[2]:

- Regeln werden zu Methoden.
- Alternativen werden zu `if`-Anweisungen.
- Regeln auf der RHS werden zu Methodenaufrufen.

Man unterscheidet dabei zwischen zwei Arten von Regeln: *Parser-Regeln* und *Tokenizer-Regeln*. Zuerst teilt ein *Tokenizer* die Zeichenkette aufgrund der Tokenizer-Regeln in eine Reihe von Tokens auf. In unserer EBNF-Beschreibung sind die Tokenizer-Regeln rot dargestellt. Die grauen Regeln werden zwar intern vom Tokenizer verwendet, aber erzeugen keine eigenen Tokens. Zum Beispiel erzeugt die Zeichenkette "`sin(1 + x) * 3.14`" die folgende Reihe von Tokens:

func : `sin(` | *num* : `1` | *op* : `+` | *var* : `x` | *close* : `)` | *op* : `*` | *num* : `3.14`

Danach entscheidet der Parser aufgrund der Parser-Regeln (oben in Schwarz dargestellt), ob eine solche Reihe von Tokens einen gültigen Ausdruck darstellt. Abbildung 2 zeigt, wie die Parser-Methode für *term* aussehen könnte.

a) In der Übungsvorlage finden Sie eine `Tokenizer`-Implementation, eine Vorlage für den `ExprParser` und eine `EvaluatorApp` mit einer `main()`-Methode. Diese parst die vom Benutzer eingegebenen Zeichenketten und gibt an, ob sie gültige Ausdrücke sind. Wenn der Benutzer "exit" eingibt, terminiert das Programm. Ihre Aufgabe ist es, den `ExprParser` zu schreiben.

   Erstellen Sie in der schon vorgegebenen `parse(String)`-Methode eine `Tokenizer`-Instanz. Die Methoden des `Tokenizers` sind denen der `Scanner`-Klasse nachempfunden. Sie können also die `hasNext*()`-Methoden verwenden, um zu prüfen, welche Art von Token als nächstes kommt, und die `next*()`-Methoden, um Tokens zu "konsumieren". Schreiben Sie die nötigen `parse*(...)`-Methoden, eine für jede Parser-Regel. Die erste Ihrer `parse*(...)`-Methoden

[1] ähnlich wie Sie, wenn Sie mit einer Tabelle überprüfen, ob ein Symbol einer EBNF-Beschreibung entspricht
[2] Im Allgemeinen, d.h. für gewisse andere EBNF-Beschreibungen, ist das leider nicht möglich.

```
/* checks if the next tokens form a valid term */
void parseTerm(...) {
    if(next token is a "open") {
        consume "open" token
        // check if the next tokens are a valid expr:
        parseExpr(...);
        check whether next token is a "close" & consume
    }
    else if(next token is a "func") {
        consume "func" token
        // check if the next tokens are a valid expr:
        parseExpr(...);
        check whether next token is a "close" & consume
    }
    else {
        // check if the tokens are a valid atom:
        parseAtom(...);
    }
}
```

Abbildung 2: Parser-Methode für *term*

```
/* evaluate the next tokens as a term */
double evalTerm(...) {
    if(next token is a "open") {
        consume "open" token
        double val = evalExpr(...);
        check whether next token is a "close" & consume
        return val;
    }
    else if(next token is a "func") {
        consume "func" token
        double arg = evalExpr(...);
        check whether next token is a "close" & consume
        double result = apply function to arg
        return result;
    }
    else {
        return evalAtom(...);
    }
}
```

Abbildung 3: Evaluator-Methode für *term*

rufen Sie von `parse(String)` aus auf. Diese Methoden sollen eine `EvaluationException` mit einer sinnvollen Fehlermeldung werfen, falls die Zeichenkette kein gültiger Ausdruck ist. Falls z.B. nach "`(`" und einer *expr* das Token "`10`" statt "`)`" folgt, könnte die Fehlermeldung lauten:

```
Syntax error: unexpected token '10', expected ')'
```

b) Um aus dem `ExprParser` einen `ExprEvaluator` zu machen, kann man die Methoden so ändern, dass sie im selben Zug das Resultat berechnen. Jede Methode überprüft dann nicht nur, ob die nächsten Tokens der Regel entsprechen, sondern gibt auch gleich den Wert des entsprechenden Ausdruck-Teils zurück. Dies sehen Sie in Abbildung 3.

Benennen Sie die Klasse und die Methoden um[3], so dass sie die neue Funktionalität widerspiegeln. Nun können Sie entscheiden: Erstens, welche Funktionen sind erlaubt? Für Aufgabe 3 sollten Sie mindestens sin(), cos() und tan() unterstützen, aber auch andere Funktionen wie abs() oder log() könnten später Spass machen[4]. Zweitens können Sie entscheiden, wie Sie mit Variablen umgehen. Sie sollten mindestens eine "x"-Variable unterstützen, und wir empfehlen, dass Sie den Wert dafür dem `ExprEvaluator`-Konstruktor übergeben. Sie sollten eine Exception werfen, falls unbekannte Funktionen oder Variablen in einem Ausdruck vorkommen.

Am Schluss sollte die `EvaluatorApp` das Resultat der eingegebenen Ausdrücke ausgegeben, statt nur zu sagen, ob sie gültig sind. Wenn Sie wollen, können Sie dem Benutzer auch die Möglichkeit geben, Werte für Variablen zu definieren.

xkcd: (by Randall Munroe (CC BY-NC 2.5)

[3]Verwenden Sie dafür die *Rename*-Funktion von Eclipse, welche Sie in der Übungsstunde gesehen haben.
[4]Schauen Sie sich die `Math`-Klasse für weitere Kandidaten an (oder implementieren Sie selber welche).

Aufgabe 3: Funktionsplotter

Mit dem `ExprEvaluator` können wir ein praktisches Programm schreiben: einen Funktionsplotter. Dieser interpretiert einen eingegebenen Ausdruck als Funktion $f(x)$, wertet $y = f(x)$ für verschiedene x aus und zeichnet die resultierenden (x, y)-Punkte.

In der Übungsvorlage finden Sie eine neue `PlotterWindow`-Klasse, welche eine Erweiterung der bekannten `Window`-Klasse ist. Der Unterschied besteht darin, dass das Fenster ein Eingabefeld enthält, wo der Benutzer eine Funktion eingeben kann. Diese kann mit der `getFunction()`-Methode abgerufen werden. Vervollständigen Sie das Programm `PlotterApp`.

a) Im ersten Schritt sollen Sie eine Koordinaten-Transformation von den Ur-Koordinaten (x, y) (in welchen $f(x)$ definiert ist) zu den GUI-Koordinaten (X, Y) implementieren.

Dem `PlotterApp`-Konstruktor werden Werte für x_{min}, x_{max}, y_{min} und y_{max} übergeben. Diese Werte geben an, welcher Teil des Ur-Koordinatensystems im Fenster sichtbar ist. Hier sind einige Beispiele für die Transformation (w und h stehen für die Breite und Höhe des Fensters):

Ur-Koordinaten (x, y)	$\rightarrow$	(X, Y) GUI-Koordinaten
$(0, 0)$		$(\frac{w}{2}, \frac{h}{2})$ falls $x_{\text{min}} = -x_{\text{max}}$ und $y_{\text{min}} = -y_{\text{max}}$
$(x_{\text{min}}, y_{\text{min}})$		$(0, h)$
$(x_{\text{max}}, y_{\text{max}})$		$(w, 0)$

Implementieren Sie zwei Methoden `toGuiX()` und `toGuiY()`, welche die Transformation berechnen. Erweitern Sie dann die `PlotterApp.run()`-Methode so, dass sie die x- und y-Achse des Ur-Koordinatensystems zeichnet. Plotten Sie ein paar Punkte im Ur-Koordinatensystem, um zu sehen, ob Ihre Berechnung korrekt ist (oder besser, schreiben Sie Tests).

Tipp: Die aktuelle Grösse des Fensters (bzw. des Teils, auf dem Sie zeichnen können) bekommen Sie wie gewohnt mit `window.getWidth()` und `window.getHeight()`.

b) Erweitern Sie `PlotterApp` so, dass die Funktion geplottet wird. Iterieren Sie dazu über alle Werte der X-Achse im GUI-Koordinatensystem ($0 \leq X < w$), finden Sie für jedes X das dazugehörige x im Ur-Koordinatensystem und berechnen Sie $y = f(x)$. Transformieren Sie diese y-Werte zurück ins GUI-Koordinatensystem und verbinden Sie die (X, Y)-Punkte mit Linien. Zusätzlich zur `toGuiY()`-Methode brauchen Sie dafür auch eine `fromGuiX()`-Methode.

Um $y = f(x)$ zu berechnen, brauchen Sie natürlich Ihren `ExprEvaluator`. Übergeben Sie ihm in jeder Iteration den aktuellen x-Wert. Falls bei der Evaluation ein Fehler auftritt, sollen Sie die Fehlermeldung auf dem Fenster ausgeben.

c) **Optional:** Erweitern Sie Ihren Plotter um zusätzliche Features, zum Beispiel:

- Achsen-Striche und -Beschriftung: Zeichnen Sie zusätzlich zu den Achsen Haupt- und Nebenstriche und fügen Sie die dazugehörigen Werte hinzu.
- Automatische Skalierung der y-Achse: Berechnen Sie y_{min} und y_{max} basierend auf den erhaltenen y-Werten in jeder Iteration der `while(window.isOpen())`-Schleife neu.
- Machen Sie Ihre `PlotterApp` interaktiv. Erlauben Sie z.B. Rein- und Rauszoomen oder Verschieben, oder zeigen Sie für den x-Wert, auf den die Maus zeigt, den y-Wert an.

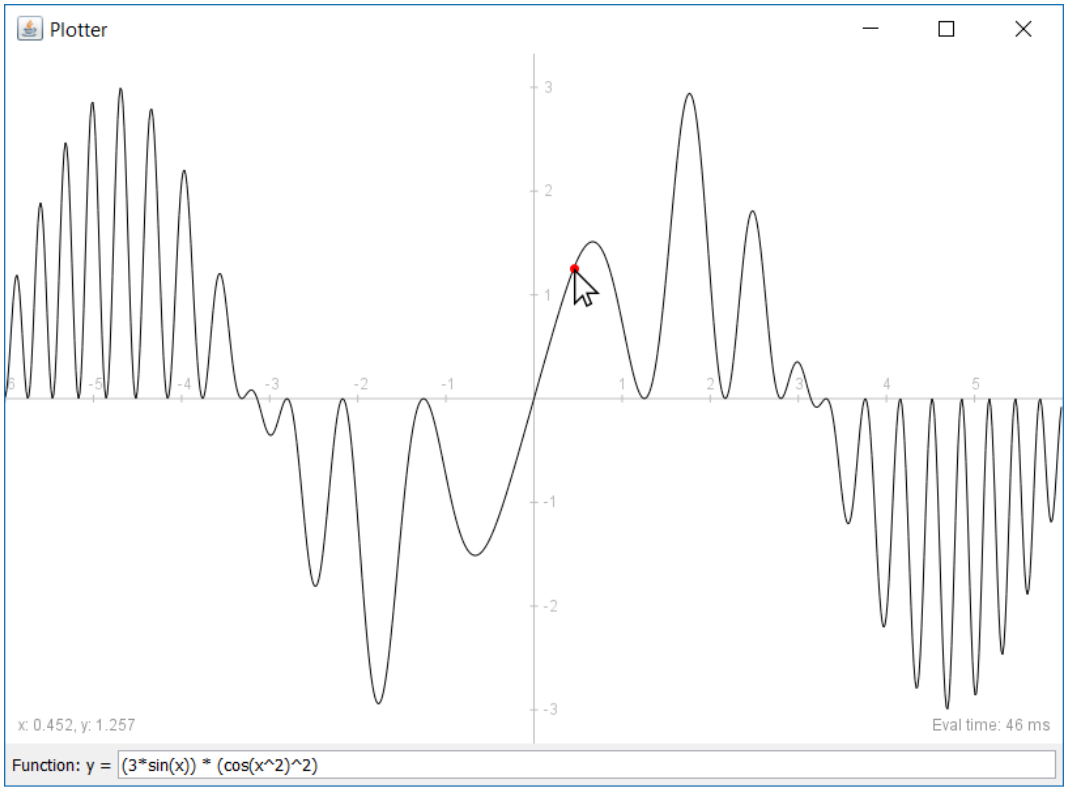
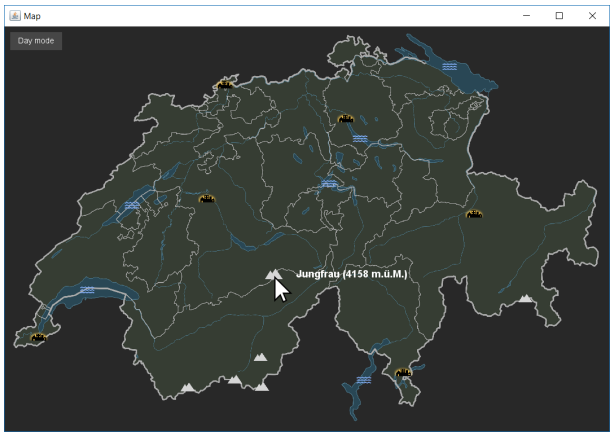


Abbildung 4: `PlotterApp` mit Zusatz-Features

Aufgabe 4: Interfaces

In dieser Aufgabe üben Sie den Umgang mit Java-Interfaces. Das Besondere an Interfaces ist, dass eine Klasse nicht nur eines, sondern beliebig viele davon implementieren kann.

Die Bibliothek, welche die `Window`-Klasse enthält, enthält auch einige Interfaces. Diese erlauben es, GUI-Programme modularer zu schreiben: Während Sie bisher alle Zeichenbefehle und Interaktionen in der `while(window.isOpen())`-Schleife durchführen mussten, können Sie mit diesen Interfaces verschiedene *Komponenten* erstellen, welche sich selbstständig zeichnen und auf Benutzereingaben reagieren. Ihre Aufgabe ist es, die interaktive Karte vom letzten Übungsblatt mit solchen Komponenten zu implementieren und zusätzlich eine Taste, welche in den "Nachtmodus" wechselt, hinzuzufügen:



In der Vorlage finden Sie weiter die `SwissMap`- und die verschiedenen POI-Klassen. Wie in der letzten Übung wird in der Vorlage nur der Kartenhintergrund gezeichnet. Allerdings geschieht dies nun mithilfe des `Drawable`-Interfaces, welches von `SwissMap` implementiert wird. Dies bedeutet, dass die `SwissMap`-Deklaration den Teil "`implements Drawable`" enthält und dass `SwissMap` die `Drawable.draw()`-Methode implementiert. Beachten Sie, dass sich das `Drawable`-Interface, sowie weitere Interfaces und Klassen sich im `gui.component`-Package befinden und wie die `Window`-Klasse importiert werden müssen. Sie können dafür z.B. `import gui.component.*;` verwenden.

Um das `SwissMap`-Objekt im Fenster anzuzeigen, wird es in der `show()`-Methode als Komponente dem Fenster hinzugefügt (`window.addComponent(this);`). Für alle so hinzugefügten `Drawable`-Komponenten wird in `Window.refresh...()` die `draw()`-Methode aufgerufen. Ihre Aufgabe ist es, das ganze Programm mit solchen Komponenten zu implementieren, so dass in der `while(window.isOpen())`-Schleife nur `window.refresh...()` aufgerufen werden muss.

a) Ändern Sie die `PointOfInterest`-Klasse so ab, dass sie ebenfalls `Drawable` implementiert. Sie können die `draw()`-Methode direkt in `PointOfInterest` implementieren, oder in jeder der Subklassen `City`, `Mountain` und `Lake` separat[5]. Im zweiten Fall müssen Sie aber eine (leere) `draw()`-Implementierung in `PointOfInterest` erstellen, um den Compiler zufriedenzustellen[6].

   Die `draw()`-Methode soll den entsprechenden POI auf das übergebene `window` zeichnen. Wenn Sie wollen, können Sie dafür `drawImageCentered()` und die PNG-Bilder in Ihrem Projektordner

[5] Sie können auch Teile davon in `PointOfInterest` und den Rest in den Subklassen implementieren, was vor allem später, wenn Sie auch die Beschreibung anzeigen, Sinn macht.

[6] Hier könnte man wiederum `abstract`-Klassen verwenden.

verwenden. Um den POI an der richtigen Position darzustellen, brauchen Sie die `toGuiX()`- und `toGuiY()`-Methoden von `SwissMap`. Erstellen Sie deshalb in der `PointOfInterest`-Klasse ein Feld, wo eine Referenz zur `SwissMap`-Instanz gespeichert werden kann, und ändern Sie alle nötigen Konstruktoren und Konstruktoren-Aufrufe so ab, dass die Instanz übergeben wird. In der `SwissMap.show()`-Methode könnten die Instanzierungen der POI-Klassen z.B. so aussehen:

```
new City(this, "Zürich", 683354, 247353, 396030, 91.88)
```

Damit die POIs gezeichnet werden, müssen sie (gleich wie die `SwissMap`) als Komponenten zur `Window`-Instanz hinzugefügt werden. Starten Sie das Programm und stellen Sie sicher, dass alles richtig angezeigt wird, bevor Sie weiterfahren.

b) Erweitern Sie `PointOfInterest` jetzt so, dass der Benutzer (wie in der letzten Übung) mit der Maus auf ein POI zeigen kann, um dessen Beschreibung anzuzeigen. Dazu muss die Klasse zusätzlich das `Hoverable`-Interface implementieren, welches die beiden Methoden `onMouseEnter()` und `onMouseExit()` und zusätzlich `getBoundingBox()` deklariert. Mit letzterer Methode kann eine Komponente ihren interaktiven Bereich mithilfe eines `Rectangle`-Objekts angeben[7]. Wenn der Benutzer dann seine Maus in diesen Bereich hinein oder aus dem Bereich hinaus bewegt, wird `onMouseEnter()` bzw. `onMouseExit()` aufgerufen.

   Implementieren Sie `onMouseEnter()` und `-Exit()` also so, dass sich der POI merkt, ob im Moment gerade auf ihn gezeigt wird, und verwenden Sie diese Information dann beim Zeichnen in `draw()`, um die Beschreibung entweder anzuzeigen oder nicht.

c) Erstellen Sie als letztes eine neue Klasse `NightModeButton`, welche nicht nur `Drawable` und `Hoverable`, sondern auch `Clickable` implementiert und als Taste auf der Karte angezeigt wird. Wenn der Benutzer auf diese Taste klickt, soll die Karte in den "Nachtmodus" wechseln, welcher alle Komponenten in einer dunkleren Version anzeigt (und bei erneutem Klick zurück).

   `Clickable` deklariert die Methode `onLeftClick()`. Implementieren Sie sie so, dass sie ein Feld `nightMode` in der `SwissMap`-Instanz verändert (welche Sie wieder via Konstruktor dem `NightModeButton` übergeben sollten). Ändern Sie danach alle `draw()`-Methoden so ab, dass die Komponenten hell oder dunkel gezeichnet werden, abhängig vom `nightMode`-Feld der `SwissMap`. Erstellen Sie schliesslich in `SwissMap.show()` eine `NightModeButton`-Instanz und fügen Sie sie als Komponente dem Fenster hinzu.

   Beachten Sie, dass `Clickable` auch `onRightClick()` deklariert. Der Compiler zwingt Sie, diese Methode ebenfalls zu implementieren, aber Sie können sie leer lassen.

Aufgabe 5: Loop Invariante

Gegeben ist die Methode `findLargestSmaller(int[] al, int value)`, die in einem *sortierten nicht-leeren* Array von `int`-Werten den grössten Wert findet, der strikt kleiner als `value` ist. `value` muss strikt grösser als `al[0]` sein.

Was ist die Invariante für den Loop in der Methode `findLargestSmaller`? Wenn die Elemente `a[0]` .... `a[K]` des Arrays `a` sortiert sind, dann können Sie das mit `Sorted(a, 0, K)` abkürzen.

[7] Die `Rectangle`-Klasse befindet sich ebenfalls in `gui.component`.

```
static int findLargestSmaller(int[] al, int value) {
    // Precondition: Sorted(al, 0, al.length-1) && al.length >= 1 && value > al[0]
    int candidate  = al[0];
    int next = 1;

    // Loop Invariante:
    while (next < al.length && value > al[next] ) {
        candidate = al[next];
        next++;
    }

    // Postcondition: Sorted(al, 0, al.length-1) && al.length >= 1 &&
        ((next < al.length && value <= al[next] && candidate == al[next-1]) ||
        (next == al.length && candidate == al[al.length-1] && value > candidate))

    return candidate;
}
```

Schreiben Sie die Loop-Invariante in die Datei "LoopInvariante.txt".

Aufgabe 6: Contact Tracing (Bonus!)

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe "Leistungskontrolle" im `www.vvz.ethz.ch`). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich die allgemeinen Regeln.

In dieser Aufgabe implementieren Sie eine Contact-Tracing-Applikation, welche es ermöglichen soll, Kontakte während eines Virus-Ausbruches nachzuverfolgen. Ihre Implementierung soll zunächst Begegnungen zwischen verschiedenen `Person`-Instanzen anonym protokollieren, so dass bei einem positivem Test die Benachrichtigung aller Personen möglich ist, die direkt oder indirekt mit einer positiv getesteten Person in Kontakt standen.

Anonyme Begegnungen. Um Anonymität zu gewährleisten, dürfen zwei Personen A und B bei einer Begegnung lediglich anonyme Integer-IDs austauschen, ohne dabei die Identität der jeweils anderen Person aufzudecken. Beide Personen speichern hierbei sowohl die eigene ID als auch die ID der anderen Person. Bei der positiven Testung von A kann dann mithilfe der anonymen IDs, die A genutzt hat, festgestellt werden, ob B einer dieser IDs begegnet ist. Um zu vermeiden, dass wiederkehrende IDs die Identifikation einer Person über mehrere Begegnungen hinweg ermöglichen, benutzt jede Person für jede Begegnung frische IDs, welche über eine zentrale Klasse `ContactTracer` vergeben werden. Frisch bedeutet hierbei, dass eine ID zuvor noch nie bei einer Begegnung verwendet wurde.

Direkte und indirekte Kontakte. Nachdem eine Reihe an Begegnungen protokolliert wurden, wird eine oder mehrere Personen positiv getestet. Mit dem erfassten Netzwerk aus Begegnungen soll Ihre Applikation dann zwei verschiedene Arten an Kontaktpersonen bestimmten:

- Als *direkte Kontakte* gelten alle Personen, die eine Begegnung mit einer positiv getesteten Person hatten.
- Als *indirekte Kontakte* hingegen gelten alle Personen, die zwar selbst keine Begegnung mit einer positiv getesteten Person hatten, jedoch Kontakt mit mindestens einer anderen Person, welche als direkter Kontakt gilt, hatten. Indirekte Kontakte mit mehr als einer Zwischenperson müssen Sie dabei nicht berücksichtigen.

Sie dürfen dabei annehmen, dass zunächst alle Begegnungen erfasst werden und erst dann Personen positiv getestet werden. Nach der ersten positiven Testung finden keine weiteren Begegnungen mehr statt.

Benachrichtigungen. Da nicht alle Personen gleichermassen gefährdet sind, soll Ihre Applikation die Benachrichtigung der Kontaktpersonen vom Alter, der Art des Kontaktes, sowie dem Testergebnis der jeweiligen Kontaktperson abhängig machen. Dabei soll eine der drei Warnstufen *Keine Benachrichtigung*, *Low-Risk-Benachrichtigung* oder *High-Risk-Benachrichtigung* ausgesprochen werden. Zu Beginn haben alle Personen die Standard-Warnstufe *Keine Benachrichtigung* und gelten als negativ getestet. Davon ausgehend sollen nach jedem registrierten positiven Test die zugehörigen Kontaktpersonen wie folgt benachrichtigen werden:

Testergebnis der Kontaktperson	Alter der Kontaktperson	Direkter Kontakt	Indirekter Kontakt
Positiv	-	Keine Benachr.	Keine Benachr.
Negativ	$\leq$ 60 Jahre alt	High-Risk	Keine Benachr.
Negativ	$>$ 60 Jahre alt	High-Risk	Low-Risk

Eine negativ getestete Person, die höchstens 60 Jahre alt ist und die nur in indirektem Kontakt zu einer positiven Person stand, soll beispielsweise keine Benachrichtigung erhalten (Reihe 2). Eine negativ getestete Person über 60 Jahre hingegen soll als indirekter Kontakt eine Low-Risk-Benachrichtigung erhalten (Reihe 3).

Wenn mehrere Personen positiv getestet werden, soll Ihre Applikation immer die höchste geltende Warnstufe für die anderen, negativ getesteten Personen berechnen. Dabei ist die Ordnung der Warnstufen wie folgt definiert: *Keine Benachrichtigung* < *Low-Risk Benachrichtigung* < *High-Risk Benachrichtigung*. Positiv getestete Personen hingegen sollen immer die Warnstufe *Keine Benachrichtigung* erhalten. Im Allgemeinen dürfen Sie zudem annehmen, dass eine Person, die einmal positiv getestet wurde, für den Rest der Laufzeit Ihrer Applikation als positiv getestet gilt.

Implementierung. Erweiteren Sie den vorgegebenen Code für die Klasse `ContactTracer` und das Interface `Person` wie folgt, um die Contact-Tracing-Applikation umzusetzen:

Implementieren Sie das Interface `Person` mit den folgenden public Methoden:

- `Person.getUsedIds()`. Diese Methode gibt die Liste aller IDs zurück (`List<Integer>`), die für diese `Person` als frische ID verwendet wurden, um eine Begegnung zu protokollieren. Nach Hinzufügen einer ID in diese Liste muss dieselbe ID in die jeweilige `Person.getSeenIds()`-Liste des Gegenübers eingetragen sein.

- `Person.getSeenIds()`. Diese Methode gibt die Liste aller IDs zurück (`List<Integer>`), die diese Person als die frische ID des jeweiligen Gegenübers bei einer Begegnung protokolliert hat. Nach Hinzufügen einer ID in diese Liste muss dieselbe ID in die jeweilige `Person.getUsedIds()`-Liste des Gegenübers eingetragen sein.

- `Person.getNotification()`. Diese Methode gibt den aktuellen Benachrichtigungsstatus der `Person` zurück. Der Rückgabewert soll vom Enum-Typ `NotificationType` sein, welcher vorgegeben ist und die drei möglichen Warnstufen modelliert. `NotificationType` ist im Interface `Person` definiert und enthält die drei Werte `NoNotification` (keine Benachrichtigung), `LowRiskNotification` (Low-Risk-Benachrichtigung) und `HighRiskNotification` (High-Risk-Benachrichtigung).

- `Person.setTestsPositively()`. Diese Methode wird aufgerufen, um eine `Person` als positiv getestet zu markieren. Nach dem Aufrufen dieser Methode sollen automatisch alle Kontakte von A benachrichtigt worden sein und die entsprechenden Warnstufe per `Person.getNotification()` zurückgeben.

Implementieren Sie zusätzlich die Klasse `ContactTracer`, welche die folgenden public Methoden besitzt:

- `ContactTracer.registerEncounter(Person p1, Person p2)`. Mit dieser Methode wird eine (beidseitige) Begegnung zwischen `Person`-Objekten `p1` und `p2` protokolliert, indem die beiden Personen anonyme IDs austauschen. Die ausgetauschten IDs müssen dabei unterschiedlich sein. Eine Begegnung zwischen `p1` und `p2` ist beidseitig und muss somit auch als Begegnung zwischen `p2` und `p1` gewertet werden.

- `ContactTracer.createPerson(int age)`. Diese Methode gibt ein `Person`-Objekt zurück. Das Alter der Person ist durch den `age` Parameter bestimmt.

Alle `Person`-Objekte werden von der Methode `ContactTracer.createPerson(int age)` erstellt. Der `ContactTracer` wird über den parameterfreien Konstruktor `ContactTracer()` instanziiert. Sie dürfen annehmen, dass nie mehr als 1024 Begegnungen zwischen Personen protokolliert werden.

Implementieren Sie auf Basis dieser Vorlage eine Lösung für das Contact-Tracing-Problem. Tests finden Sie in der Datei "ContactTracerTest.java". Die Datei "ContactTracerGradingTest.java" enthält die Tests, welche wir bei der Prüfung für die Korrektur verwendet haben. Wir empfehlen, diese Tests erst zu verwenden, wenn Sie denken, dass Ihre Lösung korrekt ist, damit Sie sehen können, wie Sie bei einer Prüfung abgeschnitten hätten.