


 **Artemis** 6.4.0

[Course Overview](#)

  ge64baw

[Courses](#) > [Praktikum: Grundlagen der Programmierung WS22/23](#) > [Exercises](#) > [W07H02 - Sicherheit Geht Vor](#)

 **W07H02 - Sicherheit Geht Vor**

Hausaufgabe

Medium

Submission due:


8 months ago



Complaint due:

8 months ago

Points: 6 of 6

Assessment: automatic ?

 Resume practice in exercise

 100%  (8 months ago)

GRADED

Recent results:



Show all results

Tasks:

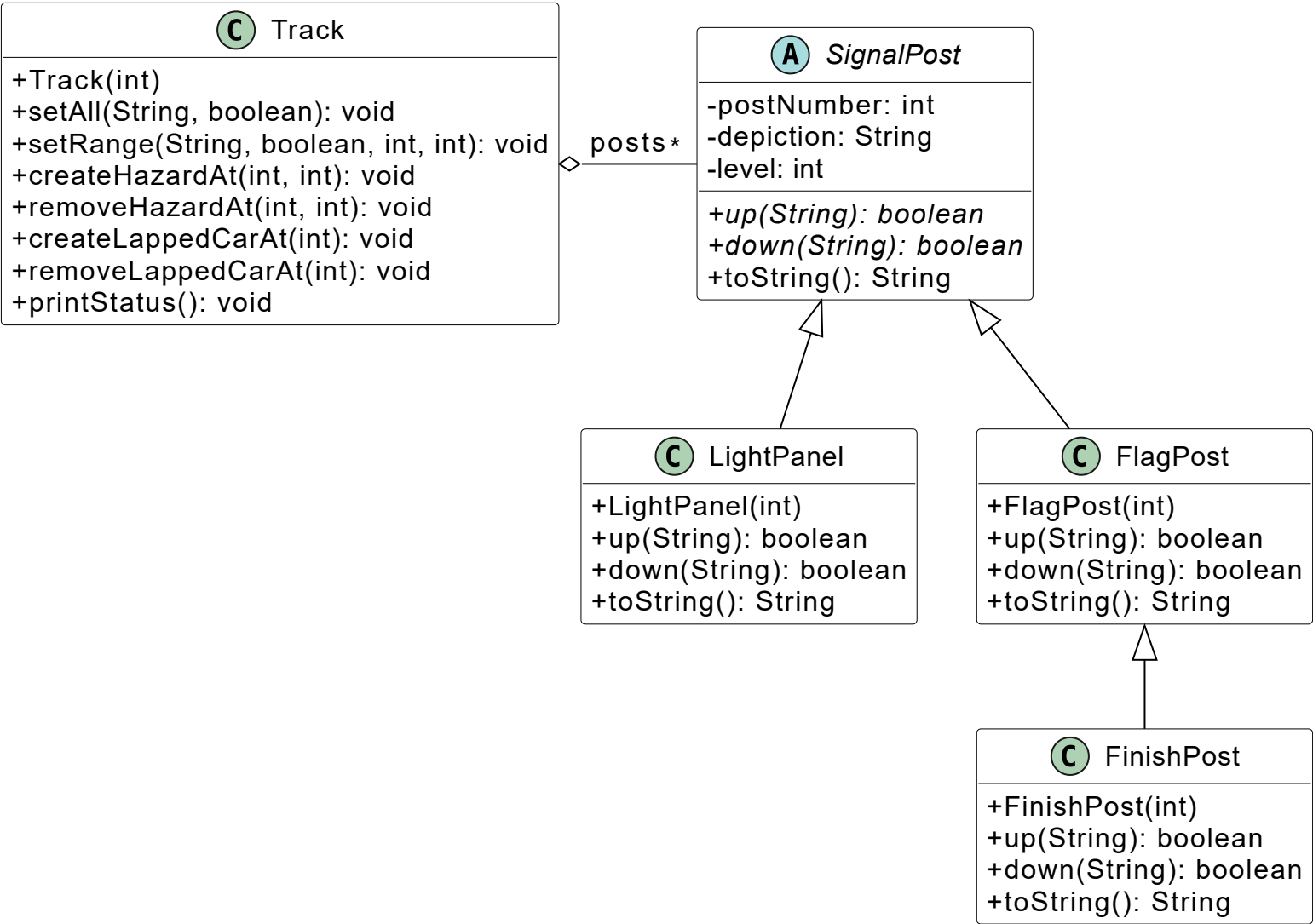
W07H02 - Sicherheit geht vor!

Nach der Blamage der FIA beim Regenrennen des Großen Preises von Japan in Suzuka sind sich alle Raceuine einig, dass Änderungen eingeführt werden müssen. Alle Raceuine? Nein, Max Whalestappen (Fahrer für das Gegnerteam Red Polar Bear) ist natürlich der Meinung, dass man im Regen wunderbar sehen kann. Natürlich lässt er dabei außer Acht, dass er ganz vorne war und seine Sicht nicht durch die Gischt anderer Raceuine getrübt wurde. Allerdings schenkt niemand seiner arroganten Meinung Beachtung und du wurdest nun dazu beauftragt, das optische Signalsystem zu überarbeiten. Natürlich ist auch der FIA bewusst, dass es zu viele Flaggen und Regeln gibt. Deswegen wollen sie dich erst mal nur mit den paar Grundfunktionen beauftragen.

Allgemeine Hinweise:

- Im Diagramm kursiv dargestellte Methoden sind abstrakte Methoden.
- Bevor die Behavioral Tests ausgeführt werden, müssen erst die Structural Tests durchlaufen. Die Structural Tests sind alle public und testen, ob alle geforderten Klassen, Methoden und Attribute vorhanden sind. Zudem testen sie, ob Getter, Setter und Konstruktor funktionieren.
- Diese Aufgabe enthält recht viel Erklärungen u.a. zu den beiden Zustandsdiagrammen, damit hoffentlich alle Fragen geklärt werden. Nicht abschrecken lassen!
- Achtung: Es gibt 2 Textpassagen, die fast gleich ausschauen. Sie sind aber nicht gleich, sonst könnte man sie weglassen.
- Im Template sind 2 implementierte Klassen gegeben:
 - Helper** kann mit der Methode `static String changeColors(String depiction)` die übergebene `depiction` passend einfärben, damit das Debugging leichter wird. Ob du diese Methode benutzen willst, bleibt dir überlassen. Die Tests akzeptieren beides. Was die `depiction` ist, erfährst du später.
 - Main** hat eine main-Methode, die ein kleines Beispiel enthält, das du ausführen kannst, wenn du fertig bist. Den korrekten Output findest du zum Vergleich ganz unten.

Folgendes Diagramm soll von dir implementiert werden:

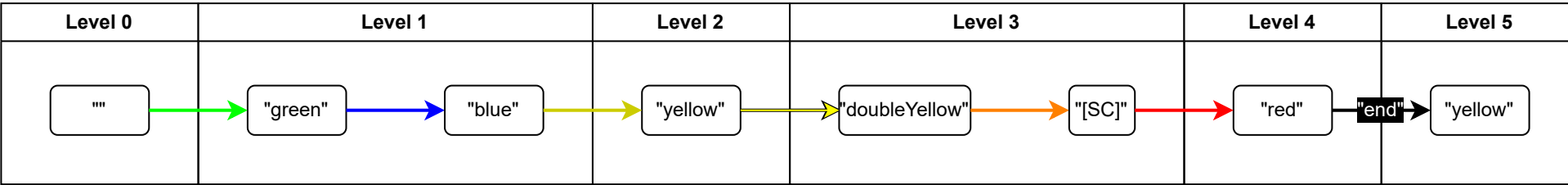


Klasse **SignalPost**:

Die abstrakte Klasse **SignalPost** hat eine **postNumber**, die seine Position (beginnend bei 0, allerdings muss nicht auf negative Werte geachtet werden) speichert, eine **depiction**, die die aktuelle Darstellung des **SignalPosts** speichert, sowie ein **level**, welches das aktuelle Level des **SignalPost** speichert. Was es mit dem Level auf sich hat, wird bei den Unterklassen erklärt. Denke daran, die Getter und Setter zu den Attributen zu implementieren. Die Signatur der Klasse stimmt noch nicht und muss von dir angepasst werden. Ein neuer **SignalPost** wird mit der übergebenen **postNumber**, sowie **level=0** und **depiction=""** initialisiert. Die beiden abstrakten Methoden **up()** und **down()** werden durch die Unterklassen implementiert. Allerdings darfst du weitere Hilfsmethoden implementieren, falls du sie brauchst. Die **toString()** soll einen **String** nach folgendem Schema zurückgeben:
"Signal_Post_<postNumber>:<level><depiction>"

Klasse **LightPanel**:

Die Klasse **LightPanel** ist unsere erste von **SignalPost** erbende Klasse. Sie wird genauso wie ihre Oberklasse initialisiert. Hier kommen jetzt konkret das **level** und die **depiction** zum Einsatz.



Im oben stehenden Diagramm siehst du alle Zustände, in denen sich ein **LightPanel** befinden kann + die Transitionen für die Methode **up()**. Jeder Zustand besteht aus einem **level** und einer **depiction**. Eine detaillierte Beschreibung der Level findest du nochmals weiter unten. Du kannst davon ausgehen, dass sich das **LightPanel** für die folgenden Methoden immer in einem gültigen Ausgangszustand befindet. Benutze keine weiteren Variablen zur Speicherung der Zustände außer denen, die vorgegeben sind. Sonst könnten Fehler entstehen. Konstanten sollten keine Probleme machen. Kommen wir nun zu den drei Methoden, die wir überschreiben müssen:

- boolean up(String type)** bekommt einen String **type** übergeben. Folgende Eigenschaften soll die Methode (wie im Diagramm zu sehen) erfüllen:
 - Ein gültiger **type** entspricht einem der Strings aus {**green**, **blue**, **yellow**, **doubleYellow**, **[SC]**, **red**, **end**}.
 - Der Zustand des **LightPanel**s wird analog zum Diagramm oben geändert. Die **types** entsprechen demnach ihrem Zielzustand. Mit **type="green"** geht es also in den Zustand **green** in Level 1, mit **type="doubleYellow"** geht es in den Zustand **doubleYellow** in Level 3, usw. Mit **type="end"** geht es in Level 5 (siehe oben).
 - Das Wechseln soll allerdings nur in Pfeilrichtung von einem beliebigen Zustand in einen beliebigen Zustand weiter rechts möglich sein. Also nur, wenn sich das Level erhöht oder innerhalb eines Levels dem Vorrang entsprechend der Zustand geändert wird.
 - Das Diagramm ist komplett transitiv, man kann also bspw. aus dem Zustand **green** (Level 1) mit **type="red"** direkt zu **red** (Level 4) gehen. Vor **blue** zu **green** oder von **red** zu **yellow** sollen beispielsweise nicht möglich sein, da sie entgegen der Pfeilrichtung sind.
 - Setze nicht nur die **depiction** korrekt, sondern auch das **level**.
 - Gebe dabei **true** zurück, falls sich der Zustand geändert hat, ansonsten **false**.
 - Falls der Parameter einem anderen String als den gültigen Strings entspricht, so soll nichts passieren und **false** zurückgegeben werden.
- boolean down(String type)** bekommt ebenfalls einen String **type** übergeben, der allerdings nur einem der gültigen Strings aus {**clear**, **green**, **blue**, **danger**} entsprechen kann. Bei allen anderen Strings soll die Methode nichts machen und **false** zurückgeben. Gebe auch hier **true** zurück, falls sich der Zustand ändert, ansonsten **false**. Außerdem soll sie sich folgendermaßen verhalten:

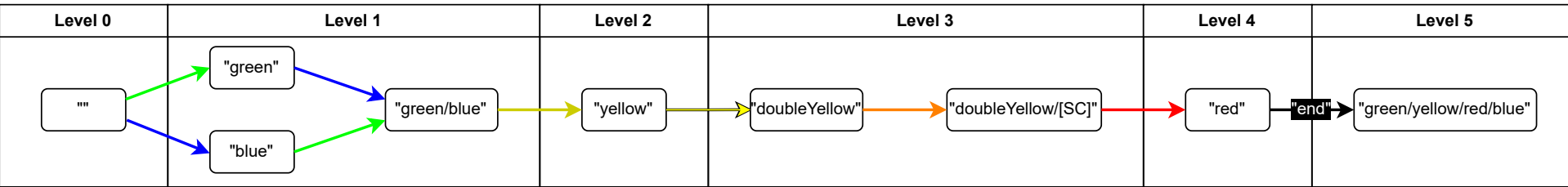
- mit `type="clear"` kann aus jedem Zustand in das `level=0` mit `depiction=""` gewechselt werden.
- mit `type="green"` wird in das `level=0` mit `depiction=""` gewechselt, sofern sich das `LightPanel` zuvor im Zustand `green` befunden hat.
- mit `type="blue"` wird in das `level=0` mit `depiction=""` gewechselt, sofern sich das `LightPanel` zuvor im Zustand `blue` befunden hat.
- mit `type="danger"` wird in das `level=1` mit `depiction=green` gewechselt, sofern sich das `LightPanel` zuvor im Zustand `yellow`, `doubleYellow`, `[SC]` oder `red` befunden hat. Beachte hierbei, dass das `yellow` aus Level 5 nicht gemeint ist.
- `String toString()` soll einen `String` nach folgendem Schema zurückgeben. Ob du dafür die `depiction` mit der vorgegebenen Hilfsmethode färbst oder nicht, bleibt dir überlassen. Die Tests akzeptieren beides:
 - falls `level=0`: `Signal_post_<postNumber>_of_type_light_panel_is_in_level_<level>_and_is_switched_off`.
 - sonst: `Signal_post_<postNumber>_of_type_light_panel_is_in_level_<level>_and_is_blinking_<depiction>`.

Die `level` mit den zugehörigen `depictions` (die relevanten Informationen stehen vor den "-----"):

- Level 0: `depiction=""`. ----- Die Strecke ist an diesem `LightPanel` frei, die Raceuine können Gas geben.
- Level 1: `depiction="green"` oder `"blue"`. Dabei hat "blue" Vorrang vor "green". ----- Die Strecke ist hier entweder nach einem Hindernis/einer Störung wieder freigegeben (green) oder ein langsamer Raceuin wird zum Überholen lassen aufgefordert (blue).
- Level 2: `depiction="yellow"`. ----- Hier befindet sich auf der Strecke ein Hindernis/eine Störung und die Raceuine müssen vorsichtig vorbeifahren und dürfen nicht überholen.
- Level 3: `depiction="doubleYellow"` oder `"[SC]"`. Dabei hat "[SC]" Vorrang vor "doubleYellow". ----- Eine noch strengere Form von Level 2. Die Raceuine müssen zusätzlich vom Gas gehen (doubleYellow). Wenn zudem das Safety Car rausgeschickt wird, so müssen alle Raceuine diesem langsam hinterherfahren ([SC]).
- Level 4: `depiction="red"`. ----- Das Hindernis/die Störung ist so groß, dass das Rennen unterbrochen werden muss.
- Level 5: `depiction="yellow"`. ----- Das Ende des Rennens ist erreicht, der Sieger steht fest.

Klasse `FlagPost`:

Die Klasse `FlagPost` ist unsere andere Klasse, die von `SignalPost` erbt. Sie funktioniert ähnlich wie das `LightPanel`, aber hat kleine Unterschiede, da diese Klasse imstande ist, mehrere Farben gleichzeitig zu zeigen. Ein `FlagPost`-Objekt soll wie die Oberklasse implementiert werden.



Im oben stehenden Diagramm siehst du alle Zustände, in denen sich ein `FlagPost` befinden kann + die Transitionen für die Methode `up()`. Eine detaillierte Beschreibung der Level findest du nochmals weiter unten (es gibt kleine Änderungen, wie du schon dem Diagramm entnehmen kannst). Du kannst davon ausgehen, dass sich der `FlagPost` für die folgenden Methoden immer in einem gültigen Ausgangszustand befindet. Benutze keine weiteren Variablen zur Speicherung der Zustände außer denen, die vorgegeben sind. Sonst könnten Fehler entstehen. Konstanten sollten keine Probleme machen. Kommen wir nun zu den drei Methoden, die wir überschreiben müssen:

- `boolean up(String type)` bekommt einen `String type` übergeben. Im Vergleich zum `LightPanel` gibt es folgende Änderungen:
 - Das Diagramm ist auch hier transitiv, allerdings gibt es eine Ausnahme: Der Übergang von Level 0 zu `green/blue` ist als einziger **nicht** direkt möglich. Allerdings kann man durch mehrfaches Aufrufen von `up()` mit `blue` zu `blue` und dann mit `green` zu `green/blue` gehen, oder alternativ mit `green` zu `green` und dann mit `blue` zu `green/blue` gehen.
 - Zudem gibt es zur `depiction=doubleYellow/[SC]` kein zugehöriges `type`. Hier soll man mit `type=[SC]` zu `doubleYellow/[SC]` wechseln können.
- `boolean down(String type)` bekommt ebenfalls einen `String type` übergeben. Sie soll sich ähnlich wie in `LightPanel` verhalten, aber mit paar Ausnahmen bei bestimmten Zuständen:
 - mit `type="clear"` kann aus jedem Zustand in das `level=0` mit `depiction=""` gewechselt werden.
 - mit `type="green"` wird in das `level=0` mit `depiction=""` gewechselt, sofern sich der `FlagPost` zuvor im Zustand `green` befunden hat. Oder aber die `depiction` wird zu `blue` und das Level bleibt bei 1, sofern sich der `FlagPost` in Zustand `green/blue` befunden hatte.
 - mit `type="blue"` wird in das `level=0` mit `depiction=""` gewechselt, sofern sich das `FlagPost` zuvor im Zustand `blue` befunden hat. Oder aber die `depiction` wird zu `green` und das Level bleibt bei 1, sofern sich der `FlagPost` in Zustand `green/blue` befunden hat.
 - mit `type="danger"` wird in das `level=1` mit `depiction=green` gewechselt, sofern sich das `FlagPost` zuvor im Zustand `yellow`, `doubleYellow`, `doubleYellow/[SC]` oder `red` befunden hat.
- `String toString()` soll einen `String` nach folgendem Schema zurückgeben. Ob du dafür die `depiction` mit der vorgegebenen Hilfsmethode färbst oder nicht, bleibt dir überlassen. Die Tests akzeptieren beides. **Achte auf die doppelten Leerzeichen um "flag post", "waving" herum und vor "doing"**. Sie dienen der Formatierung für bessere Lesbarkeit auf der Konsole:
 - falls `level=0`: `Signal_post_<postNumber>_of_type_flag_post_is_in_level_<level>_and_is_doing_nothing`.
 - sonst: `Signal_post_<postNumber>_of_type_flag_post_is_in_level_<level>_and_is_waving_<depiction>`.

Auch hier gibt es wieder die `level` und die zugehörigen `depictions` mit kleinen Unterschieden bei Level 1, 3 und 5 (die relevanten Informationen stehen vor den "-----"):

- Level 0: `depiction=""`. ----- Die Strecke ist an diesem `FlagPost` frei, die Raceuine können Gas geben.
- Level 1: `depiction="green"` oder `"blue"` oder `"green/blue"`. ----- Die Strecke ist hier entweder nach einem Hindernis/einer Störung wieder freigegeben (green) oder ein langsamer Raceuin wird zum Überholen lassen aufgefordert (blue), oder sogar beides gleichzeitig (green/blue).

- Level 2: `depiction="yellow"`. ----- Hier befindet sich auf der Strecke ein Hindernis/eine Störung und die Raceuine müssen vorsichtig vorbeifahren und dürfen nicht überholen.
- Level 3: `depiction="doubleYellow"` oder `"doubleYellow/[SC]"`. Dabei kann "[SC]" nicht ohne "doubleYellow" sein und hat Vorrang vor nur "doubleYellow". ----- Eine noch strengere Form von Level 2. Die Raceuine müssen zusätzlich vom Gas gehen (doubleYellow). Wenn zudem das Safety Car rausgeschickt wird, so müssen alle Raceuine diesem langsam hinterher fahren (doubleYellow/[SC]).
- Level 4: `depiction="red"`. ----- Das Hindernis/die Störung ist so groß, dass das Rennen unterbrochen werden muss.
- Level 5: `depiction="green/yellow/red/blue"`. ----- Das Ende des Rennens ist erreicht, der Sieger steht fest.

Klasse `FinishPost`:

Die Klasse `FinishPost` erbt von `FlagPost`, da sie bis auf 2 kleine Unterschiede genau die gleiche Funktionalität wie `FlagPost` hat. Folgende Methoden musst du überschreiben:

- falls `type=end`, soll `boolean up(String type)` die `depiction` zu `"chequered"` ändern und das Level zu 5 ändern. In allen anderen Fällen verhält sich die Methode wie in der Oberklasse `FlagPost`.
- `String toString()` soll einen `String` nach folgendem Schema zurückgeben. Ob du dafür die `depiction` mit der vorgegebenen Hilfsmethode färbst oder nicht, bleibt dir überlassen. Die Tests akzeptieren beides. **Achte auf die doppelten Leerzeichen um "waving" herum und vor "doing"**. Sie dienen der Formatierung:
 - falls `level=0`: `"Signal_post_<postNumber>_of_type_finish_post_is_in_level_<level>_and_is__doing_nothing"`.
 - sonst: `"Signal_post_<postNumber>_of_type_finish_post_is_in_level_<level>_and_is__waving__<depiction>"`.

Klasse `Track`:

Nun wollen wir unsere ganzen `SignalPosts` in der Klasse `Track` steuern. Ein neues `Track`-Objekt initialisiert ein `SignalPost[]` mit der übergebenen Größe nach folgendem Schema: Beginnend mit der ersten Stelle im Array wird alle 3 Stellen der `SignalPost` als `LightPanel` initialisiert, sonst als `FlagPost`. Die `postNumbers` sollen bei 0 beginnen und aufsteigend zählen. Der allerletzte `SignalPost` wird aber als `FinishPost` initialisiert. Sollte der Parameter des Konstruktors `<= 0` sein, soll das Array mit 10 initialisiert werden. Implementiere auch hier Getter und Setter. Im weiteren Verlauf können aber beliebige Anordnungen von `SignalPosts` durch den Setter gegeben werden. Es ist aber garantiert, dass das Array nicht `null` ist, dass es keinen `SignalPost` gibt, der `null` ist und dass das Array auch nicht `leer` ist.

▼ Beispiel zum Initialisieren

```
Array wird mit 10 initialisiert.
SignalPost (kurz SP) 0 ist ein LightPanel
SP 1 ist ein FlagPost
SP 2 ist ein FlagPost
SP 3 ist ein LightPanel
SP 4 ist ein FlagPost
SP 5 ist ein FlagPost
SP 6 ist ein LightPanel
SP 7 ist ein FlagPost
SP 8 ist ein FlagPost
SP 9 ist ein FinishPost
```

Zudem soll die Klasse folgende Methoden implementieren. Du kannst davon ausgehen, dass die übergebenen String-Parameter `!= null` sind und die int-Parameter innerhalb der Array-Grenzen sind (`>= 0` und `< posts.length`):

- `void setAll(String type, boolean up)`: Wenn `up=true` ist, soll auf allen `SignalPosts` `up(type)` mit dem übergebenen `type` aufgerufen werden, ansonsten `down(type)`.
- `void setRange(String type, boolean up, int start, int end)`: Wenn `up=true` ist, soll auf allen `SignalPosts` im Bereich zwischen `start` und `end` (beide inklusive) `up(type)` mit dem übergebenen `type` aufgerufen werden, ansonsten `down(type)`. Beachte, dass `start` auch nach `end` liegen kann. In dem Fall beginnt man bei `start`, geht bis zum Ende vom Array und fängt dann vorne wieder an, bis man bei `end` angekommen ist (Ein Kreis hat keinen Anfang und kein Ende).
- `void createHazardAt(int start, int end)` ruft auf allen `SignalPosts` im Bereich zwischen `start` und `end` (beide inklusive) `up("yellow")` auf. Auf dem `SignalPost` an der Stelle `end` soll allerdings direkt `up("green")` aufgerufen werden. Die Regeln für die Grenzen aus `setRange()` gelten hier auch.
- `void removeHazardAt(int start, int end)` ruft auf allen `SignalPosts` im Bereich zwischen `start` und `end` (beide inklusive) `down("danger")` auf. Auch hier gelten dieselben Bedingungen für die Grenzen.
- `void createLappedCarAt(int post)` ruft auf dem an `post` liegenden `SignalPost` sowie auf den 3 folgenden `SignalPosts` `up("blue")` auf. Falls das am Ende vom Array passiert ... nun ja, mittlerweile solltest du wissen, was zu tun ist.
- `void removeLappedCarAt(int post)` ruft auf dem an `post` liegenden `SignalPost` sowie auf den 3 folgenden `SignalPosts` `down("blue")` auf.
- `void printStatus()` printet nach folgendem Schema einen String auf die Konsole: Für jeden `SignalPost` im Array werden die Strings der einzelnen `SignalPosts` getrennt durch `"\n"` konkateniert und geprintet (den Zeilenumbruch auch am Ende einfügen, sodass eine Leerzeile entsteht, wenn man `printStatus` mehrmals aufrufen würde. Im Beispiel unten ist das nochmal gut zu sehen). Ob die Strings durch die Hilfsmethode gefärbt sind oder nicht, bleibt dir überlassen.

Aufgabe:

? **Structure** No results

Implementiere das gegebene UML-Diagramm.

? **SignalPost** No results

Implementiere die Klasse **SignalPost**, wie oben beschrieben. Hier wird nur die **toString()** getestet, da Konstruktor, Getter und Setter bei Structure untergebracht sind und die anderen beiden Methoden durch die Unterklassen implementiert werden.

? **LightPanel** No results

Implementiere die Klasse **LightPanel** wie oben beschrieben.

? **FlagPost** No results

Implementiere die Klasse **FlagPost** wie oben beschrieben.

? **FinishPost** No results

Implementiere die Klasse **FinishPost** wie oben beschrieben.

? **Track** No results

Implementiere die Klasse **Track** wie oben beschrieben.

▼ Beispiel aus der main-Methode

Signal post 0 of type light panel is in level 0 and is switched off
Signal post 1 of type flag post is in level 0 and is doing nothing
Signal post 2 of type flag post is in level 0 and is doing nothing
Signal post 3 of type light panel is in level 0 and is switched off
Signal post 4 of type flag post is in level 0 and is doing nothing
Signal post 5 of type flag post is in level 0 and is doing nothing
Signal post 6 of type light panel is in level 0 and is switched off
Signal post 7 of type flag post is in level 0 and is doing nothing
Signal post 8 of type flag post is in level 0 and is doing nothing
Signal post 9 of type finish post is in level 0 and is doing nothing

Signal post 0 of type light panel is in level 0 and is switched off
Signal post 1 of type flag post is in level 0 and is doing nothing
Signal post 2 of type flag post is in level 1 and is waving blue
Signal post 3 of type light panel is in level 1 and is blinking blue
Signal post 4 of type flag post is in level 1 and is waving blue
Signal post 5 of type flag post is in level 1 and is waving blue
Signal post 6 of type light panel is in level 0 and is switched off
Signal post 7 of type flag post is in level 0 and is doing nothing
Signal post 8 of type flag post is in level 0 and is doing nothing
Signal post 9 of type finish post is in level 0 and is doing nothing

Signal post 0 of type light panel is in level 0 and is switched off
Signal post 1 of type flag post is in level 0 and is doing nothing
Signal post 2 of type flag post is in level 0 and is doing nothing
Signal post 3 of type light panel is in level 0 and is switched off
Signal post 4 of type flag post is in level 0 and is doing nothing
Signal post 5 of type flag post is in level 0 and is doing nothing
Signal post 6 of type light panel is in level 0 and is switched off
Signal post 7 of type flag post is in level 0 and is doing nothing
Signal post 8 of type flag post is in level 0 and is doing nothing
Signal post 9 of type finish post is in level 0 and is doing nothing

Signal post 0 of type light panel is in level 0 and is switched off
Signal post 1 of type flag post is in level 0 and is doing nothing
Signal post 2 of type flag post is in level 0 and is doing nothing
Signal post 3 of type light panel is in level 2 and is blinking yellow
Signal post 4 of type flag post is in level 2 and is waving yellow
Signal post 5 of type flag post is in level 2 and is waving yellow
Signal post 6 of type light panel is in level 1 and is blinking green
Signal post 7 of type flag post is in level 0 and is doing nothing
Signal post 8 of type flag post is in level 0 and is doing nothing
Signal post 9 of type finish post is in level 0 and is doing nothing

Signal post 0 of type light panel is in level 0 and is switched off
Signal post 1 of type flag post is in level 0 and is doing nothing
Signal post 2 of type flag post is in level 0 and is doing nothing
Signal post 3 of type light panel is in level 1 and is blinking green
Signal post 4 of type flag post is in level 1 and is waving green
Signal post 5 of type flag post is in level 1 and is waving green
Signal post 6 of type light panel is in level 1 and is blinking green
Signal post 7 of type flag post is in level 0 and is doing nothing
Signal post 8 of type flag post is in level 0 and is doing nothing
Signal post 9 of type finish post is in level 0 and is doing nothing

Signal post 0 of type light panel is in level 3 and is blinking [SC]
Signal post 1 of type flag post is in level 3 and is waving doubleYellow/[SC]
Signal post 2 of type flag post is in level 3 and is waving doubleYellow/[SC]
Signal post 3 of type light panel is in level 3 and is blinking [SC]

Exercise details

Release date:	Dec 1, 2022 18:30
Submission due:	Dec 18, 2022 18:00
Complaint due:	Dec 25, 2022 18:00

Every student is allowed to complain once per exercise. In total 1000 complaints are possible in this course. You still have 998 complaints left. ⓘ