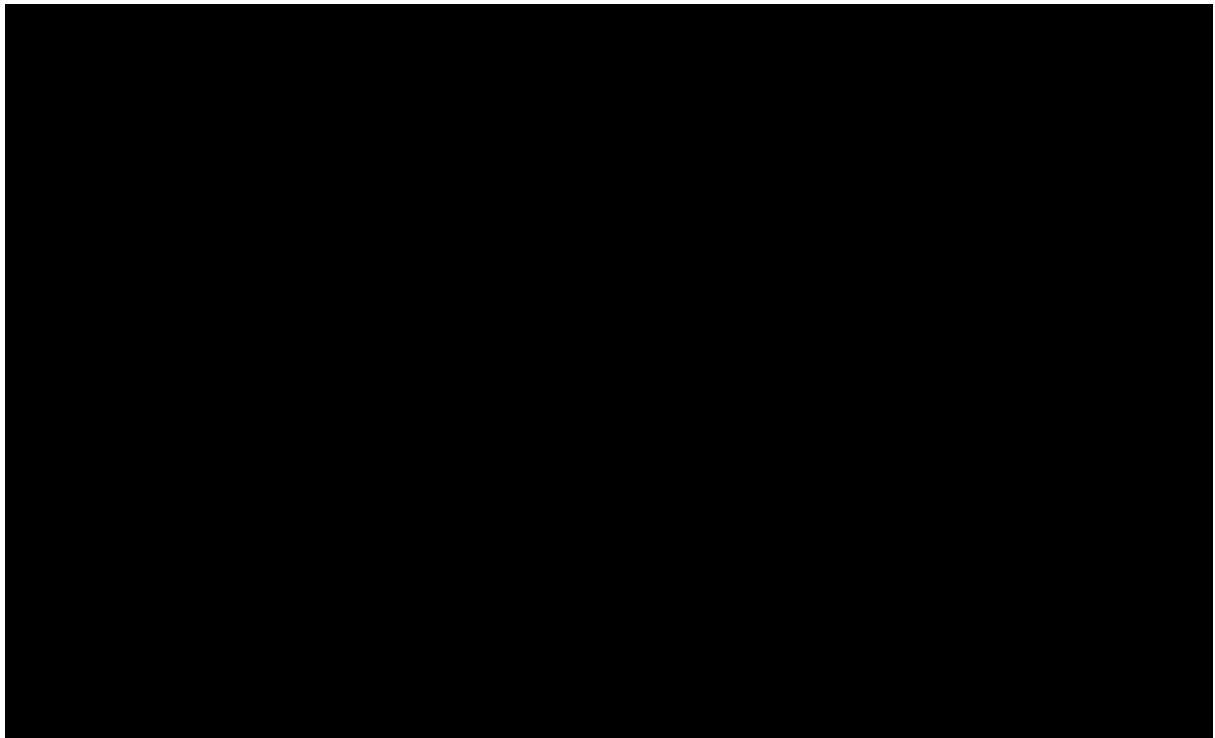


Runnable Sushi

Im Restaurant "Runnable Sushi" gibt es ein F rderband mit mehreren Pl tzen (nummeriert mit 0 - n). An den einzelnen Pl tzen befinden sich

-   K che, die Speisen (z. B. Appetizer, Sushi) zubereiten und am Band abstellen,
-   G ste, die Speisen vom Band nehmen und essen, sowie
-   Abr umer, die  brig gebliebene Speisen vom Band nehmen.

In der folgenden Skizze sehen Sie ein Restaurant mit 12 Pl tzen: an Position 0 steht der Abr umer, an Position 1 und 2 die K che, an den Positionen 3, 5, 7 und 10 haben G ste Platz genommen.



Die Speisen sind auf einem F rderband abgestellt, sodass sie sich im Uhrzeigersinn von Platz zu Platz bewegen. D. h. die Personen bleiben immer an der gleichen Position, w hrend die Speisen an ihnen vorbeiziehen.

Vorg nge im Restaurant:

-   Wenn das Restaurant  ffnet, beginnt das F rderband zu laufen.
-   Die K che nehmen ihre Position ein und beginnen mit der Zubereitung der Speisen, die etwas Zeit in Anspruch nimmt. Fertige Speisen werden auf das F rderband gelegt.
-   Die G ste nehmen eine Speise vom Band und essen diese. Nach einiger Zeit wiederholen sie diesen Vorgang.
-   Wenn das Restaurant schlie t, beenden die K che ihre Arbeit.
-   Anschlie end werden die G ste gebeten, das Restaurant zu verlassen.
-   Der Abr umer beginnt seine Arbeit und entfernt die  brig gebliebenen Speisen vom Band.
-   Schlie lich wird das Band gestoppt.

Aufgabenstellung

Implementieren Sie eine Simulation des Restaurants "Runnable Sushi" mit unten angeführten Anforderungen. Orientieren Sie sich an folgendem Klassendiagramm:



- ⌘ Die Klassen können im Bedarfsfall um weitere Felder und Methoden ergänzt werden. Zudem können Sie zusätzliche Klassen erstellen, wenn dies erforderlich ist, z. B. bei Umsetzung des Strategy-Pattern.
- ⌘ Erstellen Sie Getter- und Setter-Methoden nur, wenn diese benötigt werden.
- ⌘ Thread-bezogene Klassen können wahlweise von `Thread` abgeleitet werden oder das `Runnable`-Interface implementieren.

FoodType und Food

- ⌘ Die Enumeration `FoodType` enthält die verschiedenen Arten von Speisen (z. B. "Sushi", "Appetizer").
- ⌘ Die Klasse `Food` repräsentiert die Speisen. Speisen haben eine `id` und einen `foodType`. Die `id` wird vom Koch vergeben und besteht aus einer Kombination des Namens des Produzenten und einer laufenden Nummer je Koch ("S-1" ist z. B. das Produkt Nr 1 vom Sushikoch "S").

Belt

- ⌘ Die Klasse `Belt` repräsentiert das Förderband. Realisieren Sie dieses als Food-Array, wobei der Index eines Elements die Position des Platzes darstellt.
- ⌘ Es sollen folgende Methoden implementiert werden:
 - ! `isValidPosition(int pos)`: überprüft, ob die Position `pos` auf dem Band existiert.
 - ! `isFreePosition(int pos)`: überprüft, ob die Position `pos` auf dem Band frei ist.
 - ! `isEmpty()`: überprüft, ob das gesamte Band leer ist.
 - ! `add(Food food, int pos)`: Stellt eine Speise `food` an Position `pos` am Band ab. Wenn die Position belegt ist, wird `false` zurückgegeben.
 - ! `remove(int pos)`: Entfernt die Speise an Position `pos` vom Band und gibt sie zurück.
 - ! `move()`: Bewegt die Speisen um eine Position weiter. Beachten Sie: Das letzte Element muss

an die erste Stelle des Bands gesetzt werden.

! `toString()`: Gibt den aktuellen Zustand des Bands als String zurÜck (siehe Konsolenausgabe).

¥ Das Band soll in einem Thread laufen, der die Speisen alle 0.5 Sekunden weiterbewegt.

!

Immer wenn das Band bewegt wird, müssen wartende Kšche oder GŠste benachrichtigt werden.

¥ Nach jedem Bewegen des Bandes soll der aktuelle Zustand des Bandes (abrufbar Über `toString()`) in der Konsole ausgegeben werden.

Konsolenausgabe: Zustand des Fšrderbands

```
- (0: S-1) - (1: ...) - (2: A-5) - (3: ...) - (4: A-4) - (5: ...) - (6: ...) - (7: A-3) - (8: ...) - (9: ...)
- (10: ...) - (11: ...)
- (0: ...) - (1: S-1) - (2: ...) - (3: A-5) - (4: ...) - (5: A-4) - (6: ...) - (7: ...) - (8: A-3) - (9: ...)
- (10: ...) - (11: ...)
```

Producer

¥ Die Klasse `Producer` repršsentiert die Kšche. Jeder Koch bereitet eine bestimmte Speiseart zu (z. B. "Sushi", "Appetizer").

¥ Das Zubereiten einer Speise dauert 1 bis 2 Sekunden. Anschließend wird sie auf das Band gelegt. Wenn die Position auf dem Band belegt ist, wartet der Koch, bis die Position frei ist. Dann legt er die Speise ab und beginnt mit der nŠchsten Zubereitung.

¥ Jeder Koch lšuft in einem eigenen Thread und produziert Speisen, bis er gestoppt wird. Beim Stoppen soll der Koch alle produzierten und abgelegten Speisen in der Konsole ausgeben.

Konsolenausgabe: Starten eines Kochs

```
Producer A starts producing at position 1 ...
```

Konsolenausgabe: Produzieren eines Kochs

```
*** A placed A-1 at position 1
```

Konsolenausgabe: Stoppen eines Kochs

```
Producer A stopped
Producer A produced: A-1 | A-2 | A-3 | A-4 | A-5 | A-6 |
```

!

Produzenten und Konsumenten sollen intern eine Liste fÜhren, in der sie alle produzierten bzw. konsumierten Produkte eintragen können, damit diese am Ende ausgegeben werden können. Verwenden Sie dazu eine geeignete Datenstruktur

ConsumerType und Consumer

- ¥ Die Enumeration `ConsumerType` definiert die verschiedenen Arten von Consumer, die Speisen vom Band nehmen, nämlich Gäste ("GUEST") und Abräumer ("CLEANER").
- ¥ Die beiden Consumer-Typen unterscheiden sich lediglich durch ihre Konsum-Strategien, welche mittels des Strategy-Pattern umgesetzt werden sollen:
 - ! Gäste warten 1 bis 5 Sekunden, bevor sie eine Speise vom Band nehmen.
 - ! Ein Abräumer nimmt jede Speise an seiner Position vom Band.
- ¥ Jeder Consumer läuft in einem eigenen Thread und nimmt Speisen vom Band, bis er gestoppt wird. Beim Stoppen sollen alle konsumierten Speisen in der Konsole ausgegeben werden.

Konsolenausgabe: Starten eines Gastes

```
Consumer Ann starts consuming at position 3 ...
```

Konsolenausgabe: Konsumieren eines Gastes

```
*** Ann consumed S-3 at position 3
```

Konsolenausgabe: Stoppen eines Gastes

```
Consumer Ann stopped.  
Ann took: S-3 | A-5 | A-3 |
```

RunnableSushi

- ¥ Die Klasse `RunnableSushi` enthält die `main`-Methode und steuert den Ablauf im Restaurant.
 1. Erstellung und Starten des Förderbandes
 2. Erstellung und Starten der Küche
 3. Erstellung und Starten der Gäste
 4. Warten auf das Schließen des Restaurants
 5. Stoppen der Küche
 6. Stoppen der Gäste
 7. Abräumen des Bandes
 8. Stoppen des Bandes

Wählen Sie für Ihr Restaurant folgendes Setting:

- ¥ 2 Küchen (S, A) bereiten Speisen zu (Sushi, Appetizer) auf Position 1 und 2.

¥ 3 GŠste (Ann, Bob, Joe) auf Positionen 3, 5 und 7.

¥ 1 AbrŠumer auf Position 0.

Konsolenausgabe eines vollstŠndigen Testlaufs

```
Runnable Sushi opens ...
Producer A starts producing at position 1 ...
-(0:...)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
Producer S starts producing at position 2 ...
Consumer Ann starts consuming at position 3 ...
Consumer Bob starts consuming at position 5 ...
Consumer Joe starts consuming at position 7 ...
-(0:...)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
-(0:...)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
*** A placed A-1 at position 1
-(0:...)-(1:...)-(2:A-1)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
-(0:...)-(1:...)-(2:...)-(3:A-1)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
*** S placed S-1 at position 2
-(0:...)-(1:...)-(2:...)-(3:S-1)-(4:A-1)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
*** A placed A-2 at position 1
-(0:...)-(1:...)-(2:A-2)-(3:...)-(4:S-1)-(5:A-1)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
-(0:...)-(1:...)-(2:...)-(3:A-2)-(4:...)-(5:S-1)-(6:A-1)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)
*** S placed S-2 at position 2
-(0:...)-(1:...)-(2:...)-(3:S-2)-(4:A-2)-(5:...)-(6:S-1)-(7:A-1)-(8:...)-(9:...)-
-(10:...)-(11:...)
*** Joe consumed A-1 at position 7
*** A placed A-3 at position 1
-(0:...)-(1:...)-(2:A-3)-(3:...)-(4:S-2)-(5:A-2)-(6:...)-(7:S-1)-(8:...)-(9:...)-
-(10:...)-(11:...)
*** Bob consumed A-2 at position 5
-(0:...)-(1:...)-(2:...)-(3:A-3)-(4:...)-(5:S-2)-(6:...)-(7:...)-(8:S-1)-(9:...)-
-(10:...)-(11:...)
-(0:...)-(1:...)-(2:...)-(3:...)-(4:A-3)-(5:...)-(6:S-2)-(7:...)-(8:...)-(9:S-1)-
-(10:...)-(11:...)
*** S placed S-3 at position 2
*** A placed A-4 at position 1
-(0:...)-(1:...)-(2:A-4)-(3:S-3)-(4:...)-(5:A-3)-(6:...)-(7:S-2)-(8:...)-(9:...)-(10:S-
1)-(11:...)
*** Ann consumed S-3 at position 3
*** Joe consumed S-2 at position 7
-(0:...)-(1:...)-(2:...)-(3:A-4)-(4:...)-(5:...)-(6:A-3)-(7:...)-(8:...)-(9:...)-
```

```

-(10:...)-(11:S-1)
*** A placed A-5 at position 1
-(0:S-1)-(1:...)-(2:A-5)-(3:...)-(4:A-4)-(5:...)-(6:...)-(7:A-3)-(8:...)-(9:...)-
-(10:...)-(11:...)-
-(0:...)-(1:S-1)-(2:...)-(3:A-5)-(4:...)-(5:A-4)-(6:...)-(7:...)-(8:A-3)-(9:...)-
-(10:...)-(11:...)-
*** S placed S-4 at position 2
Producer S stopped
*** Ann consumed A-5 at position 3
Producer S produced: S-1 | S-2 | S-3 | S-4 |
-(0:...)-(1:...)-(2:S-1)-(3:S-4)-(4:...)-(5:...)-(6:A-4)-(7:...)-(8:...)-(9:A-3)-
-(10:...)-(11:...)-
*** A placed A-6 at position 1
Producer A stopped
Producer A produced: A-1 | A-2 | A-3 | A-4 | A-5 | A-6 |
-(0:...)-(1:...)-(2:A-6)-(3:S-1)-(4:S-4)-(5:...)-(6:...)-(7:A-4)-(8:...)-(9:...)-(10:A-3)-
-(11:...)-
-(0:...)-(1:...)-(2:...)-(3:A-6)-(4:S-1)-(5:S-4)-(6:...)-(7:...)-(8:A-4)-(9:...)-
-(10:...)-(11:A-3)-
*** Bob consumed S-4 at position 5
Consumer Bob stopped.
Bob took: A-2 | S-4 |
-(0:A-3)-(1:...)-(2:...)-(3:...)-(4:A-6)-(5:S-1)-(6:...)-(7:...)-(8:...)-(9:A-4)-
-(10:...)-(11:...)-
-(0:...)-(1:A-3)-(2:...)-(3:...)-(4:...)-(5:A-6)-(6:S-1)-(7:...)-(8:...)-(9:...)-(10:A-4)-
-(11:...)-
-(0:...)-(1:...)-(2:A-3)-(3:...)-(4:...)-(5:...)-(6:A-6)-(7:S-1)-(8:...)-(9:...)-
-(10:...)-(11:A-4)-
-(0:A-4)-(1:...)-(2:...)-(3:A-3)-(4:...)-(5:...)-(6:...)-(7:A-6)-(8:S-1)-(9:...)-
-(10:...)-(11:...)-
*** Ann consumed A-3 at position 3
Consumer Ann stopped.
Ann took: S-3 | A-5 | A-3 |
*** Joe consumed A-6 at position 7
Consumer Joe stopped.
Joe took: A-1 | S-2 | A-6 |
Start clean up
Consumer Cleaner starts consuming at position 0 ...
*** Cleaner consumed A-4 at position 0
-(0:...)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:S-1)-
-(10:...)-(11:...)-
-(0:...)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-(10:S-1)-
-(11:...)-
-(0:...)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:S-1)-
-(0:S-1)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)-
*** Cleaner consumed S-1 at position 0
-(0:...)-(1:...)-(2:...)-(3:...)-(4:...)-(5:...)-(6:...)-(7:...)-(8:...)-(9:...)-
-(10:...)-(11:...)-
Consumer Cleaner stopped.

```

Cleaner took: A-4 | S-1 |

Belt stopped

Runnable Sushi closes

Process finished with exit code 0