PE Java Advanced 2022- 2023: Rollercoaster Tycoon

Tijdens deze PE implementeer een eenvoudige cli[1]-versie van het spel Rollercoaster tycoon. Gebruikers kunnen een pretpark bouwen en configureren. Je schrijft unit testen en zult werken met exceptions. Je zal bestanden moeten lezen en schrijven. Je gebruikt generics, collections, streams en lambda expressies.
Ook kan je het pretpark openen zodat bezoekers het pretpark kunnen bezoeken. Hiervoor maak je gebruik van multithreading.
Je werkt in een groep met 1, 2 of 3 studenten. De startcode is beschikbaar via github classroom. Eén teamlid accepteert de opdracht en geeft de andere groepsleden toegang tot de github repository.

Opdracht beschikbaar op: maandag 21/11/2022

Deadline: **zondag** 18/12/2022 23:59

Deliverables: **Je oplossing staat op github. ALLE teamleden doen regelmatig een commit en push naar de github repository.** Na de deadline mag er geen nieuwe code meer gepushed worden naar de github repository. Je verdedigt jouw implementatie bij jouw docent tijdens week 13 (19/12/2022 - 23/12/2022). Alle teamleden zijn aanwezig bij de verdediging. Afwezigheid tijdens de verdediging resulteert in een 0 op de PE. Je oplossing voldoet aan de eisen die op het einde van dit document staan opgelijst.

Inhoudsopgave

Opgave **2**
Deel 1: Een pretpark bouwen 3
a. Een nieuw pretpark bouwen 3
b. Inkomprijs van het huidige pretpark aanpassen 3
c. Toon keuzelijst met typen rollercoaster 3
d. Toon keuzelijst voor typen shop 4
e. Een rollercoaster toevoegen 4
f. Een shop toevoegen 4
g. Toon details van het pretpark 5
h. Een pretpark opslaan 5
i. Een bestaand pretpark laden 5
Deel 2: Een pretpark openen 5
Overzicht cli commando's **7**
Extra's **8**
Rubrics **8**
Interessante bronnen **10**

[1] cli = command-line interface

Opgave

Je implementeert een eenvoudige cli-versie van het spel Rollercoaster Tycoon. In je implementatie maak je zoveel mogelijk gebruik van **streams** en **lambda expressies**.

Een maven project met de startcode is voorzien. Deze startcode bevat het hoofdprogramma (RollercoasterTycoon.java) dat volledig is. Hier hoef je niets meer in aan te passen.

Wanneer het spel opstart moet het properties bestand *rct.properties* ingelezen worden. Dit bestand bevat enkele constante waarden van het spel. Ook moet het bestand met de verschillende typen rollercoaster[2] worden ingelezen. Het path naar dit bestand vind je terug in de properties-file.

Een bestand met de verschillende typen rollercoaster bevat steeds een header en 1 lijn per type. De lijnen hebben het volgende formaat:

```
id;name;genre;excitement;excitement_rating;nausea;nausea_rating;
cost;passengers;ride_time
```

Veld	Beschrijving
id	Id van het type rollercoaster
name	Naam van het type rollercoaster
genre	RideGenre
excitement	Opwindingsniveau van het type rollercoaster (double)
excitement_rating	Opwindingsniveau van het type rollercoaster (String)
nausea	Kans om ziek te worden in het type rollercoaster
nausea_rating	Kans om ziek te worden in het type rollercoaster (String)
cost	Kostprijs om dit type rollercoaster te bouwen in je pretpark
passengers	Aantal passagiers die tijdens 1 rit in het type rollercoaster kunnen.
ride_time	Tijdsduur van een rit.

Maak een klasse RollercoasterType die een lijn uit het bestand voorstelt. Maak gebruik van de klasse RideGenre. De klasse Rollercoaster is reeds gedeeltelijk gegeven. Voeg toe dat iedere Rollercoaster een RollercoasterType heeft.

[2] Elke attractie, zelfs een draaimolen of monorail, wordt een rollercoaster genoemd in dit spel.

Tip: Maak gebruik van extra helper-klassen!

Je werkt ook met **exceptions**. Wanneer er fouten optreden bij het uitvoeren van een cli-commando gooi je een gepaste exception op. Je bent vrij om het type van de exception-klasse te kiezen. Je moet er voor zorgen dat de gebruiker een duidelijke boodschap krijgt waarom een commando niet uitgevoerd kan worden. (bv. commando 'creeaate' is niet gekend, 'abc' is een ongeldige waarde voor entrance-fee,...)

Deel 1: Een pretpark bouwen

Implementeer onderstaande cli commando's in je programma.

a. Een nieuw pretpark bouwen

```
rct> create MyPark
```

Met het commando create wordt een nieuw Themepark aangemaakt. De GameEngine houdt een referentie bij naar het huidige themapark. Ieder themapark krijgt een startkapitaal (waarde uit property file: rct.initial_cash).

b. Inkomprijs van het huidige pretpark aanpassen

```
rct > set entrance-fee 2.5
```

Je bezoekers moeten (waarschijnlijk) betalen om het pretpark te bezoeken. Met dit commando kan je de inkomprijs van je pretpark aanpassen.

c. Toon keuzelijst met typen rollercoaster

```
rct > show -type rollercoaster

rct > show -type rollercoaster -min-cost 500 -max-cost 1000 -sorted name

rct > show -type rollercoaster -ride WATER_RIDE -all
```

De gegevens van alle typen rollercoaster die aan de gegeven voorwaarden voldoen worden getoond. Default worden de typen gesorteerd op id en wordt er een beperkt aantal gegevens getoond (minstens naam, genre en kostprijs). Je kan ook filters meegeven, zoals minimum prijs (min-cost) en maximum kostprijs (max-cost), … Je mag de filters die je aan je gebruikers aanbiedt zelf kiezen (min. 3 verschillende filters). Biedt ook minstens één extra sortering aan (bijv. kostprijs, naam, ...) Maak bij je implementatie zoveel mogelijk gebruik van de Java Streaming API.

```
[1] Bobsleigh Coaster [ROLLER_COASTER] €2700.0
[2] Dinghy Slide      [WATER_RIDE]     €1200.0
[3] Wild Mouse        [ROLLER_COASTER] €1640.0
```

```
...
[12] Bumper boats      [WATER_RIDE]      €850.0
```

Met de optie -all krijg je uitgebreidere informatie (excitement (E) en nausea(N)) te zien. Het formaat van de output mag je zelf kiezen.

d. Toon keuzelijst voor typen shop

```
rct > show -type shop
```

Toon de typen shop die er beschikbaar zijn in het spel.

```
BALLOON_STALL [SOUVENIR]
SOUVENIR_SHOP [SOUVENIR]
BURGER_BAR [FOOD]
DRINKS_STALL [DRINKS]
ICECREAM_STALL [FOOD]
HOT_DOG_STALL [FOOD]
PIZZA_STALL [FOOD]
```

e. Een rollercoaster toevoegen

```
rct> add-rollercoaster 5 Crush's coaster
```

Het commando add-rollercoaster heeft 2 parameters: <id> <name>

Het id is het type van de rollercoaster (uit de keuzelijst die je bij de start van het programma hebt ingeladen). Je geeft een duidelijke foutmelding als de opgegeven id niet bestaat. De nieuwe rollercoaster krijgt ook een unieke naam (spaties zijn mogelijk in de naam). Je moet controleren dat deze naam uniek is, anders geef je een foutmelding. Er wordt een nieuwe rollercoaster met het opgegeven type aangemaakt en toegevoegd aan het huidige pretpark. Het pretpark moet uiteraard betalen voor het bouwen van de rollercoaster. De kostprijs van de rollercoaster wordt afgetrokken van de cash van het pretpark.

f. Een shop toevoegen

```
rct> add-shop BURGER_BAR Top burgers
```

Om inkomsten te genereren kan je shops toevoegen in je pretpark.
Het add-shop commando heeft 2 parameters: <shop-type> <shop-name>. Het shop-type is een waarde uit de enum ShopType. Bij een ongeldige waarde geef je een duidelijke foutboodschap. De shop krijgt ook een unieke naam (controle!). Het bouwen van een shop kost geld, iedere shop heeft een kostprijs. De kostprijs van een shop wordt afgetrokken van de cash van het huidige pretpark.

Bezoekers kunnen producten kopen in een shop (in iedere shop is er maar 1 product te koop). Ieder product heeft een aankoopprijs (prijs die de bezoeker moet betalen) en een opbrengst voor het pretpark (profitPerItem). Deze shops zijn belangrijk om straks je bezoekers gelukkig te houden: je wilt geen hongerige of dorstige bezoekers in je pretpark.

g. Toon details van het pretpark

```
rct > describe
```

Toon de details van het huidige pretpark. Je toont de naam en het huidige beschikbare geld. Daarnaast toon je alle rollercoasters en shops die in het pretpark zijn. Zorg dat de rollercoasters en shops gesorteerd zijn op naam.

```
Name: My Themepark
Cash: 3615.0
Days open: 3
Number of rollercoasters: 2
  * Happy coaster [3D Cinema, THRILL_RIDE]
  * Wild coaster [Giga Coaster, ROLLER_COASTER]
Number of shops: 3
  * Getting thirsty? [DRINKS_STALL]
  * Happy Burger [BURGER_BAR]
  * Souvenirs [SOUVENIR_SHOP]
```

h. Een pretpark opslaan

```
rct> save my_themepark.ser
```

Wanneer je een pretpark hebt gebouwd, moet het mogelijk zijn om dit op te slaan in een bestand, zodat je het later opnieuw gemakkelijk kan hergebruiken. Je kan hiervoor Object Serialization gebruiken (zie cursustekst), maar je bent ook vrij om een eigen bestandsformaat te maken om geconfigureerde themaparken op te slaan.

i. Een bestaand pretpark laden

```
rct> load my_themepark.ser
```

Een opgeslagen pretpark (naam, cash, rollercoasters, shops) wordt opnieuw geladen.

Deel 2: Een pretpark openen

```
rct > open
```

Met het commando open, doe je het pretpark gedurende 1 dag open. Wanneer het pretpark open is, hoeft je cli niet beschikbaar te zijn (of je mag een boodschap tonen dat het pretpark open is en dat er momenteel geen commando's kunnen worden uitgevoerd). De duur (in milliseconden) van 1 dag wordt meegegeven in het properties bestand. Dit is de duur dat de wachtrijen voor de rollercoasters opengaan en er ook bezoekers gegenereerd worden om het pretpark te bezoeken.

Je maakt 3 Thread-klassen: SimulateDayThread, WaitingLine en Visitor.

1. **WaitingLine**

De klasse WaitingLine is een generieke klasse. Ze moet nl. gebruikt kunnen worden om een wachtrij te simuleren voor iedere klasse die de interface QueueArea<T> implementeert. Een wachtrij krijgt bij het aanmaken een eindtijd mee. Wanneer een wachtrij wordt gestart kunnen objecten van het type T (straks Visitors) zich in de wachtrij plaatsen. Vervolgens worden het toegelaten aantal items (getAllowed) uit de wachtrij genomen, deze items mogen de QueueArea binnengaan (enter). Als het toegelaten aantal items in de QueueArea is binnengegaan, zal de waitingline-thread een poos slapen (getTime). Daarna worden er weer het toegelaten aantal items uit de wachtrij genomen. Dit proces blijft zich herhalen tot de eindtijd van de WaitingLine is aangebroken.

Een Rollercoaster implementeert de interface QueueArea<Visitor>. Het aantal passagiers van de rollercoaster is de waarde voor getAllowed. De getTime is de duurtijd van een rit in de rollercoaster. Wanneer een visitor in de rollercoaster mag, dat stijgt zijn happiness (je mag zelf kiezen met welke waarde). Zorg ook dat je voor iedere bezoeker telt hoeveel ritjes in een rollercoaster hij maakt.

Opmerking: om de implementatie te vereenvoudigen worden voor shops geen wachtrijen gebruikt.

2. **SimulateDayThread**

Stap 1: bepaal de sluitingstijd van het themapark
Stap 2: maak voor iedere Rollercoaster een WaitingLine aan en start de threads
Stap 3: maak op willekeurige tijdstippen bezoeker-threads aan (gebruik VisitoryFactory) en start deze op (de bezoekers blijven een willekeurige tijd, ten laatste tot sluitingstijd in het pretpark).
Stap 4: als de dag is afgelopen, schrijf je een bestand (log/pretparknaam_dag.txt) aan met de volgende gegevens:

```
<naam_pretpark> @ dag <dag>
Cash: <cash_themepark>
Aantal bezoekers: <aantal_bezoekers>
Gemiddelde happiness: <gemiddelde_happiness>
Gelukkigste bezoeker: <naam_bezoeker> <happiness_bezoeker>
Ongelukkigste bezoeker: <naam_bezoeker> <happiness_bezoeker>
Gemiddelde aantal ritjes/bezoeker: <gemiddeld_aantal_ritten>
Totaal bedrag uitgegeven: <totaal_bedrag_bezoekers>
Gemiddelde tijd: <gemiddelde_tijd_in_pretpark>
Langste bezoek: <naam_bezoeker> <tijd>
```

Maak zoveel mogelijk gebruik van streams en lambda expressies om de waarden in het bestand te berekenen.

3. **Visitor (Thread)**

Bezoekers starten met het betalen van de inkomprijs van het pretpark.

Een bezoeker herhaalt de volgende stappen tijdens de duur van zijn bezoek:

Stap 1: bezoeker heeft 2% kans om dorstig of hongerig te worden

Stap 2:

Als de bezoeker in de wachtrij van een rollercoaster
-> happiness -2 (indien honger of dorstig -5)

Als de bezoeker niet in een wachtrij, dan wordt max. 1 van onderstaande acties uitgevoerd:

A. de bezoeker is hongerig
-> kies willekeurig een shop met food en koop iets (happiness +15)*

B. de bezoeker is dorstig
-> kies willekeurig een shop met drank en koop iets (happiness +10)*

C. kies random een rollercoaster wachtrij en ga in de wachtrij staan.

Stap 3: sleep 200 ms

*Als de bezoeker het bedrag niet kon betalen, zal hij uiteraard hongerig en/of dorstig blijven en zal zijn happiness niet verhogen.

Opmerkingen:het is toegelaten om tijden en happiness-waarden aan te passen in je implementatie.

Overzicht cli commando's

Commando	Betekenis
create <name>	Een themapark met de opgegeven naam wordt aangemaakt.
save <filename>	Het huidige themapark wordt weggeschreven naar het opgegeven bestand.
load <filename>	Het huidige themapark wordt geladen van het opgegeven bestand.
set <property> <value>	Geef de opgegeven property een waarde. Mogelijk properties: entrance-fee.
show -type <type>	Geef een overzicht van typen rollercoaster of shop.
add-shop <type> <name>	Voeg een shop toe in het huidige pretpark.
add-rollercoaster <type> <name>	Voeg een rollercoaster toe in het huidige pretpark.
describe	Geef een beschrijving van het huidige pretpark.
open	Open het huidige pretpark voor 1 dag.
quit	Beëindig de toepassing.

Extra's

Teams met 3 studenten zijn verplicht om extra functionaliteit te implementeren. Bespreek met de docent welke extra('s) jullie implementeren.

Hier zijn alvast enkele extra's die voor het spel:

- Tijd bijhouden die de Visitor-threads in een wachtrij doorbrengen. Hoeveel tijd van hun bezoek hebben ze in een wachtrij gestaan?
- Je moet voor iedere shop en rollercoaster personeel aannemen. Uiteraard moet je personeel aan het einde van de dag ook betaald worden.
- Rollercoaster kunnen stuk gaan, enkel als je onderhoudspersoneel hebt aangenomen kan de rollercoaster blijven werken.
- Iedere shop heeft maar een beperkte voorraad van items.
- Voor rollercoasters moet er betaald worden i.p.v. of bovenop de inkomprijs.
- Bezoekers moeten naar het toilet kunnen gaan. Indien er geen toilet is worden bezoekers die dringend moeten ongelukkig. Zorg voor toiletten.
- Zorg voor een bankautomaat waar jouw bezoekers geld kunnen afhalen.
- help-commando implementeren zodat de gebruiker weet welke commando's er uitgevoerd kunnen worden.
- Bezoekers kunnen ziek worden in een rollercoaster en naar huis gaan.

Jullie zijn uiteraard vrij om zelf extra functionaliteit toe te voegen.

Rubrics

Zorg dat elk teamlid regelmatig code toevoegt in de repository. De punten van de teamleden kunnen individueel aangepast worden, waarbij er rekening wordt gehouden met de bijdrage aan het project en inzicht in de implementatie van het project.

	Excellent 8 - 10	**Good 5 - 7**	**Poor 0 - 3**
Programma uitvoeren (10%)	Het spel kan gespeeld worden zonder foutmelding	Het programma kan uitgevoerd worden en het spel kan gespeeld worden, maar max. 2 commando's werken niet correct.	Programma kan niet uitgevoerd worden (compile time or runtime errors) Meer dan 2 commando's werken niet correct.
Unit testen (15%)	50%[(1)] van de code is voorzien van zinvolle en relevante unit testen (eenvoudige getters en setters hoeven niet getest te worden en tellen niet mee bij het percentage code coverage).	35%[(1)] van de code is voorzien van zinvolle en relevante unit testen (eenvoudige getters en setters hoeven niet getest te worden en tellen niet mee bij het percentage code coverage).	Minder dan 35%[(1)] van de code is voorzien van unit testen. De unit testen zijn niet correct geschreven.

Werken met bestanden (10%)	Alle bestanden worden correct ingelezen en gevraagde data wordt correct weggeschreven. Paden zijn relatief. Properties worden correct ingelezen en gebruikt.	Paden naar bestanden zijn niet relatief. De code om data in te lezen of weg te schrijven bevat werkt correct mits enkele kleine tekortkomingen. Alle opgegeven bestanden worden ingelezen. Gevraagde data wordt weggeschreven.	Paden zijn niet relatief. Code om bestanden te lezen en te schrijven bevat fouten. Niet alle opgegeven bestanden worden ingelezen of weggeschreven. De inhoud van de bestanden komt niet overeen met het gevraagde.
Exception handling (10%)	Exceptions worden correct geïmplementeerd, opgegooid en afgehandeld. Er zijn geen lege catch-blokken. Er zijn unit testen voor het testen van de exception handling.	Er zijn enkele kleine tekortkomingen bij de implementatie en het gebruik van exceptions. Er zijn unit testen voor het testen van de exception handling.	Fout gebruik van exceptions. Er zijn geen unit testen voor het testen van de exception handling.
Streams en lambda expressies (10%)	Er wordt correct en gebruikgemaakt van streams en lambda expressies waar dit mogelijk is.(2)	Er zijn enkele correct geïmplementeerde, relevante streams met lambda expressies aanwezig in de oplossing.	Streams en lambda expressies worden heel beperkt gebruikt of zijn fout geïmplementeerd.
Multithreading (10%)	Multithreading wordt correct geïmplementeerd. Synchronisatie tussen de threads werkt perfect.	Multithreading wordt correct geïmplementeerd. De thread synchronisatie heeft nog kleine tekortkomingen.	Multithreading is niet correct geïmplementeerd. Thread synchronisatie bevat fouten.
Implementatie en clean code (15%)	De implementatie-richtlijnen werden opgevolgd. De code is netjes geformatteerd en volgens de code conventies. De code is goed leesbaar, makkelijk onderhoudbaar en uitbreidbaar.	De opgelegde implementatie-richtlijnen zijn grotendeels opgevolgd. Een enkele keer worden code conventies niet nageleefd. De code is goed leesbaar, makkelijk onderhoudbaar en uitbreidbaar.	De implementatie-richtlijnen worden niet of amper opgevolgd. Code conventies worden niet gerespecteerd. Code is niet makkelijk leesbaar en daardoor niet onderhoudbaar.
Demo en verdediging (20%)	De demo is duidelijk en toont aan dat het	De demo toont aan dat het programma	De demo verloopt chaotisch en er doen

	programma correct werkt. Vragen over implementatie en werking van het programma worden correct beantwoord.	(grotendeels) correct werkt. Vragen over implementatie en werking van het programma worden meestal correct beantwoord.	zich problemen voor. Vragen over implementatie en werking van het programma worden foutief beantwoord.

[1] Run with Coverage.
[2] For-each loops zijn nog toegestaan.

Interessante bronnen

- https://www.baeldung.com/java-faker
- https://www.baeldung.com/java-clean-code
- https://www.baeldung.com/java-stop-execution-after-certain-time