☐ Voorblad in te leveren door student? Zo ja ☒, dan volgende blok invullen

Studentgegevens				
Naam student				
Studentnummer			Klas	
Kruis hiernaast de naam van de eigen vakdocenten aan	☐	Johan Talboom	☐	Ik volgde geen les
	☐	Etiënne Goossens	☐	Ik weet het niet
	☐	Jessica van der Heijden	☐	Hans van der Linden

Vermeld op ieder blad je naam, studentnummer en klas

Algemene gegevens van de toets					
Academie	AE&I	Fase	P	Periode	1.2
Opleiding	Technische Informatica	Onderwijsvorm	VT		
Naam vak	Introductie Software Development in Java	Vakcode	EITI-ISDJ		
Subtitel	Deel B - Praktijk	Auteur(s)	Johan Talboom		
Datum	vr 21 januari 2022		Hans van der Linden		
Tijd	14:00 – 17:20 (200 min)	Review door	Jessica van der Heijden		
Aantal pagina's (inclusief voorblad)	13	Aantal opgaven	10 opgaven, 100 punten		
Zak/slaaggrens	55 punten voor een 5.5				

Bijzonderheden			
Papier (kruis aan wat van toepassing is)	☐ Lijntjes A4	☒ Kladpapier	☐ Ruitjes A4
			☐ Schrapkaart a-e mk ned
Opgave inleveren?	☒ (☒=ja ☐=nee)		
Rekenmachine	☐ Technische Informatica: géén rekenmachine toegestaan		
Toegestane hulpmiddelen	☐ Geen hulpmiddelen toegestaan		
	☒ Boeken (titel, auteur) Praktisch UML, Warmer & Kleppe, vijfde editie		
	☒ Anders nl: **Toegestaan** is gebruik van digitaal lesmateriaal op Brightspace, eigen laptop met ontwikkeltools, eigen aantekeningen, uitwerkingen van oefenopgaven en oefentoetsen, online naslagwerken over UML en Java; **NIET toegestaan** tijdens de toets is uitwisseling van bestanden met anderen, communiceren met anderen (email, chat), code letterlijk overnemen van internetbronnen.		
Opmerkingen	Dit deel B ontvangt de student zodra de antwoorden van deel A op papier zijn ingeleverd. Bij deel B mogen de laptop en andere hulpmiddelen gebruikt worden om de uitwerking te maken en via Brightspace in te leveren.		
Contactgegevens academiebureau AE&I	Gebouw LA, office LA003 Olaf van Maurik: 088-5259293 Marina Hendrickx: 088-5257198 Gebouw CHL 13 Barbara Klijs: 088-5257494		Vakdocent (naam en tel.nr.): Hans van der Linden 088-525 7234

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

Vak	Introductie Software Development in Java
Subtitel	Deel B - Praktijk
Vakcode	EITI-ISDJ
Datum	21 jan 2022
Pagina	1 van 12

Instructie

Dit tentamen maak je in twee delen: deel A en deel B. Voor deel A krijg je een cijfer en voor deel B ook. Beide cijfers tellen voor 50% mee in je eindcijfer.

Het tentamen maak je aan de hand van een casus. Die casus wordt zowel in deel A als deel B gebruikt. Het is daarom belangrijk dat je de casus goed bestudeert.

Dit deel B bevat de praktijkopdrachten. Je uitwerking van dit deel van het tentamen bestaat uit een IntelliJ project en een antwoordblad waarin je je UML model uitwerkingen opneemt. Om die in te leveren maak je een zip-file van jouw project vanuit IntelliJ. Het antwoordblad staat in het template-project, en dus ook in je intelliJ project. Controleer of de zipfile die je inlevert jouw uitwerking en het ingevulde antwoordblad bevat.

Je mag zelf kiezen of je het Astah project of het Visual Paradigm project gebruikt, al naar gelang welke van deze twee tools je voorkeur heeft. De projecten zijn inhoudelijk gelijk. De uitbreidingen en aanpassingen op het UML model maak je in dat project. Tot slot exporteer je vanuit Astah of Visual Paradigm de diagrammen waaraan je gewerkt hebt en importeert die als plaatjes in het antwoordblad op de aangegeven plekken.

Lever als je klaar bent de zip-file van je IntelliJ project met daarin je antwoordblad in via de inleveropdracht op Brightspace in de module Introductie Software Development in Java.

- Zorg ervoor dat jouw project in de root directory van de zip-file staat (dus niet in een subdirectory). Als de zip-file geopend wordt moet direct de projectfile (.iml) te zien zijn, naast de src map. Tevens moet meteen de Astah (.asta) of Visual Paradigm (.vpp) file te zien zijn. Als je je zip-file zo indeelt versnelt dat het nakijken enorm.
- Geef je zip-file de bestandsnaam ‘Praktijktoets ISDJ – <jouw naam>’ met op de plek van <jouw naam> jouw naam, bijvoorbeeld ‘Praktijktoets ISDJ – Hans van der Linden.zip’.
- Gebruik alsjeblief een zip-file en geen rar-file, ook als dat de defaultinstelling van je archiveringstool is.

Je Java code van dit tentamen kun je zelf testen door middel van de aangeleverde testcode (ongeveer zoals bij TMC) maar je kunt NIET de TMC-testknop gebruiken. Je moet de tests vanuit IntelliJ uitvoeren. Om de testcode uit te voeren kun je in het IntelliJ project rechtsklikken op het mapje ‘tests’ en hierna klikken op ‘run all tests’.
Deze testen helpen je wel bij het uitwerken van de programmeeropdrachten maar ze geven NIET aan wat er mis is in je source code want alle foutmeldingen zijn uit de testcode gehaald.

Succes!

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

De casus: RocketLaunch

In dit tentamen werk je aan een simulatie van een raketlancering. Een simulatie werkt in kleine stappen, waarbij ieder stap een korte tijd gesimuleerd wordt.

Het model en de code zijn geschreven in het Engels, dus we zullen hier de Engelse termen gebruiken.

De raket (Engels: rocket) bestaat uit een aantal op elkaar gestapelde trappen die we ‘stages’ noemen. Elke trap bevat raketmotoren die tijdens de lancering na elkaar branden. Als de eerste trap is opgebrand neemt de tweede trap het over, enz. totdat alle trappen zijn opgebrand.Er zijn in deze casus twee verschillende soorten rakettrappen (stages): een solid fuel stage en een liquid fuel stage. Elke stage heeft fuel die tijdens de launch stap voor stap vermindert tot de stage is opgebrand (exhausted).

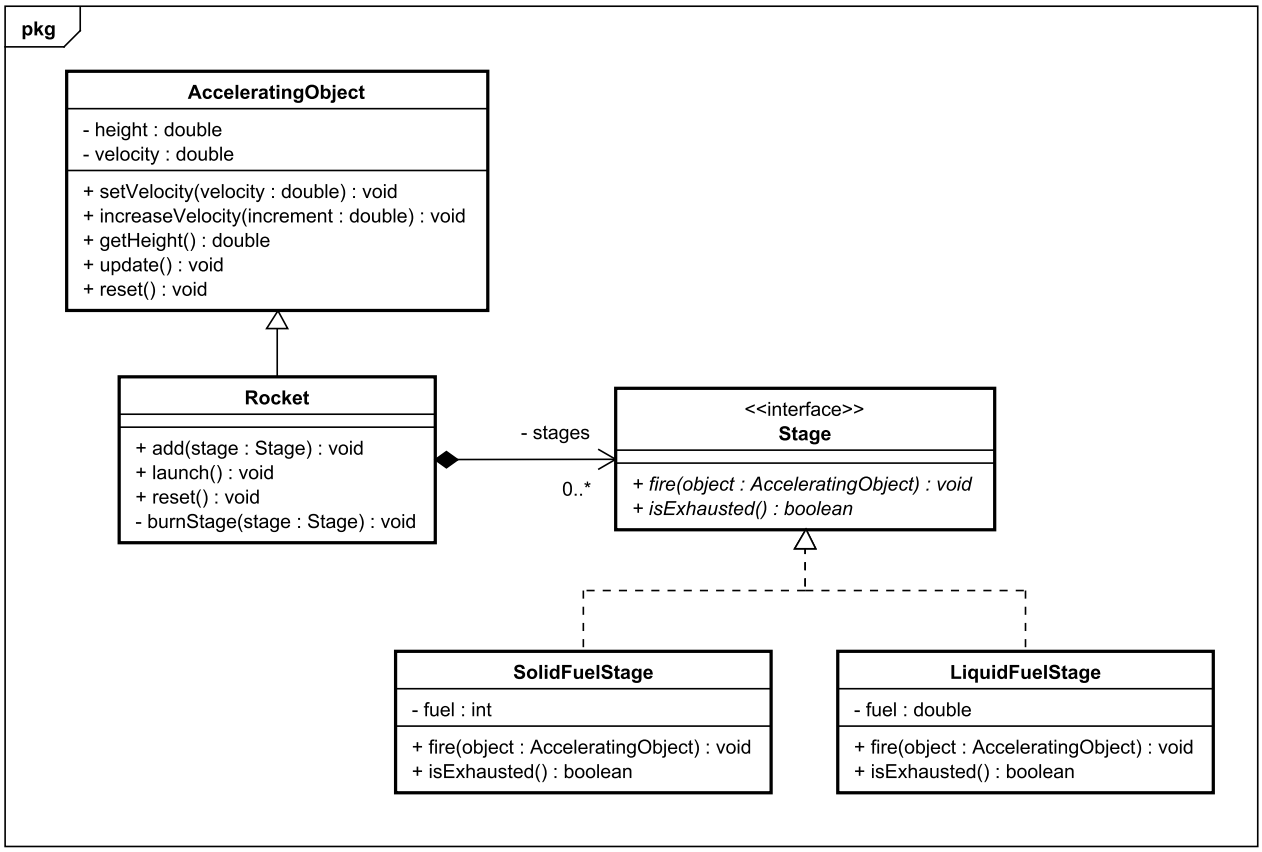
Voor deze casus zijn diverse UML diagrammen beschikbaar waarin de software is gemodelleerd: een klassendiagram van alle klassen, een use case diagram, sequence diagrammen van een aantal scenario’s en een toestandsdiagram van de rocket.

De diagrammen en de beschrijving van de casus zijn zowel in deel A als deel B opgenomen dus je hoeft de casus niet uit je hoofd te leren.

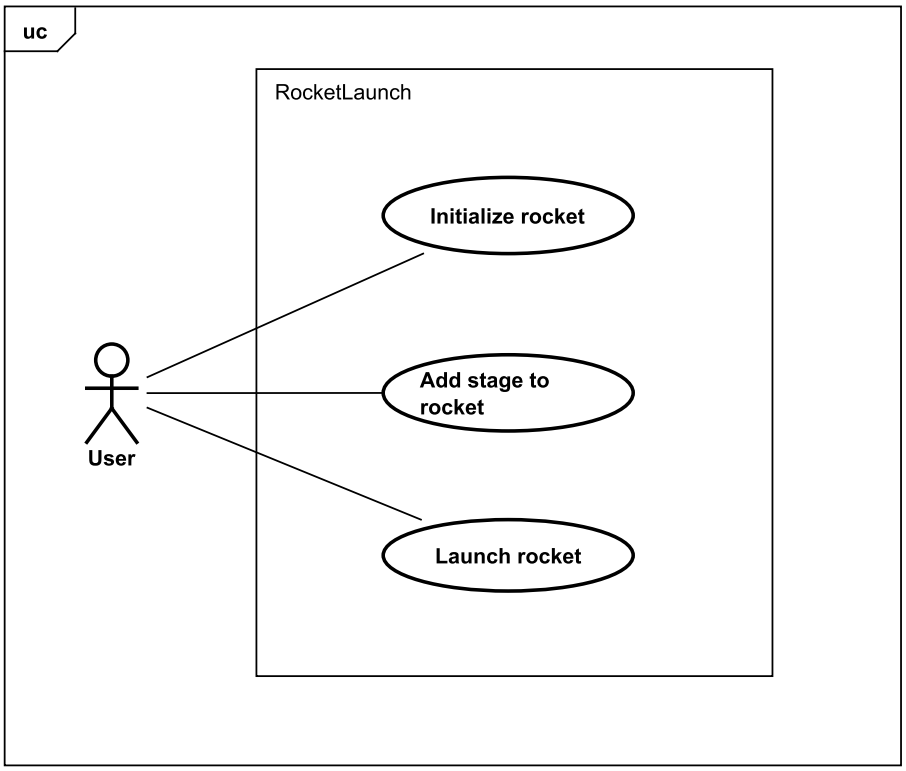
Bestudeer de beschrijving en de diagrammen goed en werk daarna in het IntelliJ project en in het Astah project of het Visual Paradigm project de opdrachten uit dit deel B uit.

In onderstaand klassendiagram worden de klassen (en interfaces) van het RocketLaunch systeem getoond, met hun onderlinge relaties. Zoals te zien is in dit diagram bestaat de Rocket uit Stages, waarvan er twee soorten zijn. De Rocket is een AcceleratingObject en die klasse heeft zijn eigen attributen en methoden om de hoogte en de snelheid te bepalen.

De twee soorten Stages houden ieder hun hoeveelheid brandstof (fuel) bij. Elke stage vermindert tijdens de fire() methode op zijn eigen wijze de hoeveelheid fuel en verandert daarbij ook op zijn eigen wijze de snelheid (velocity) en hoogte (height) van het AcceleratingObject waarvan de stage deel uitmaakt.



Er zijn drie use-cases voor het RocketLaunch systeem, zoals aangegeven in het use-case diagram.



Van elke use-case is hier een use-case-beschrijving opgenomen.

Naam	Launch rocket
Samenvatting	De user simuleert de launch van de rocket waarbij de rocket na elkaar al zijn stages opbrandt
Actoren	User
Precondities /aannamen	GUI is opgestart De rocket is aangemaakt
Beschrijving	1. De user start de launch 2. Het systeem simuleert de lancering 3. Zodra de lancering klaar is toont het systeem een popup met de bereikte hoogte
Uitzonderingen	2a. Er zijn geen stages: er komt een popup met de melding dat de lancering niet mogelijk is omdat er geen stages zijn en het scenario eindigt hier
Postcondities /resultaat	De rocket heeft door de simulatie een bepaalde hoogte en snelheid gekregen en alle stages zijn opgebrand OF De rocket is ongewijzigd omdat er geen stages zijn

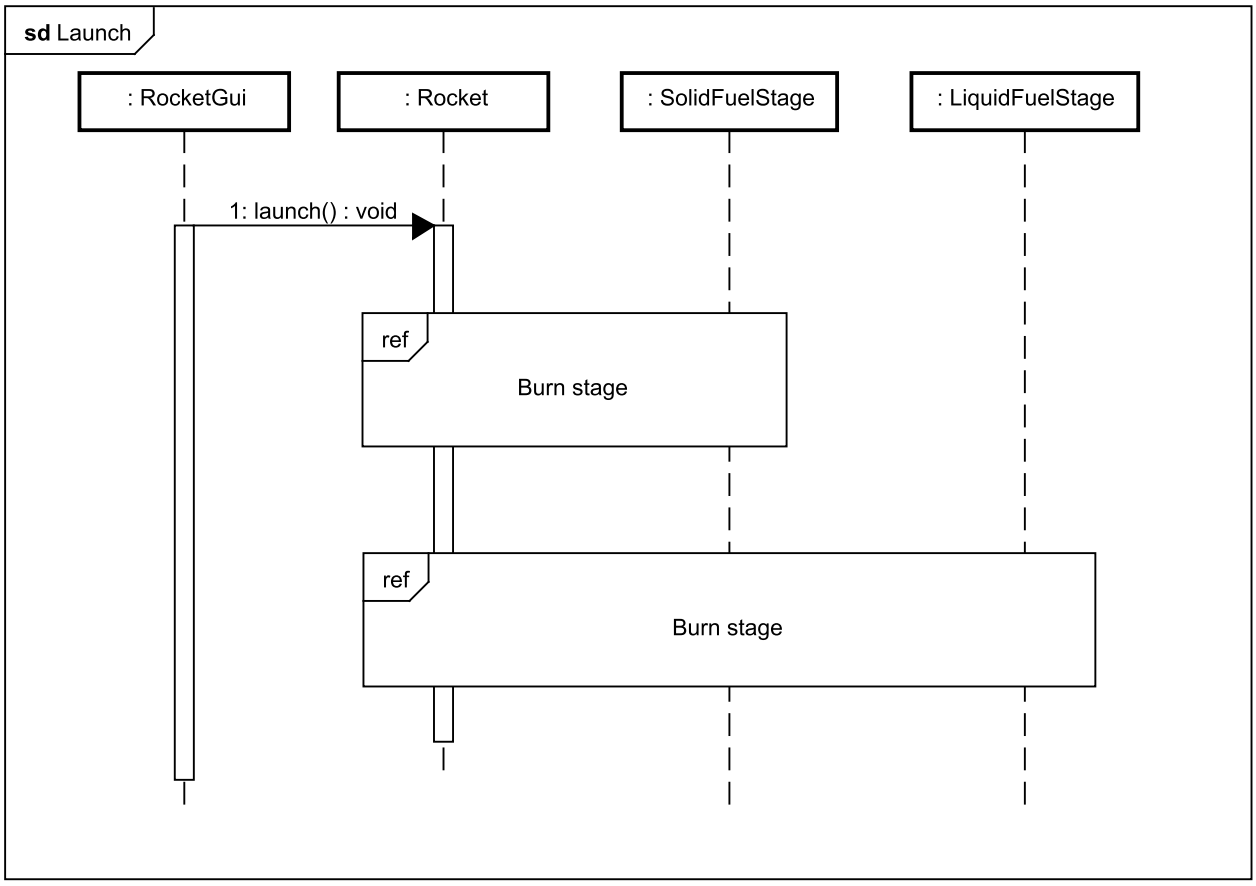
---

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

Naam	Add stage to rocket
Samenvatting	De user voegt een stage naar keuze toe aan de rocket, met parameters die passend zijn voor dat type stage
Actoren	User
Precondities /aannamen	GUI is opgestart Er is een rocket met nul of meer stages
Beschrijving	1. De user selecteert het type brandstof (solid fuel of liquid fuel) voor de stage die toegevoegd moet worden 2. De user voert de hoeveelheid fuel in 3. De user geeft aan dat de stage moet worden toegevoegd 4. Het systeem voegt een stage van het aangegeven type met de ingevoerde hoeveelheid fuel toe aan de rocket 5. Het systeem wist de selectie van het brandstoftype en de hoeveelheid fuel zodat een nieuwe stage kan worden ingevoerd
Uitzonderingen	4a. Niet alle gegevens worden ingevoerd: er komt een popup met de melding dat de stage niet toegevoegd kan worden
Postcondities /resultaat	De rocket is voorzien van één of meer stages en gereed voor launch

Naam	Initialize rocket
Samenvatting	De gebruiker brengt de rocket in de begintoestand
Actoren	User
Precondities /aannamen	GUI is opgestart Er is een rocket aangemaakt
Beschrijving	1. De user geeft aan dat de rocket naar de begintoestand terug moet gaan 2. Het systeem verwijdert alle stages van de rocket 3. Het systeem brengt de attributen van de rocket in de begintoestand voor een lancering
Uitzonderingen	Geen
Postcondities /resultaat	De rocket is gereed om er stages aan toe te voegen

In onderstaand sequence diagram wordt getoond hoe de simulatie van een lancering wordt gestart. Via een Graphical User Interface klasse RocketGui krijgt de Rocket een launch bericht. In reactie daarop zal de Rocket na elkaar elk van zijn stages laten ontbranden. In dit specifieke geval is dat eerst een SolidFuelStage en daarna een LiquidFuelStage.



Voorbereiding

Begin eerst met het downloaden van het IntelliJ project van Brightspace. Hierin staat al een aantal klassen, de Astah en Visual Paradigm projecten (kies zelf met welk van deze twee tools je wilt werken) en het antwoordblad.

Hernoem tenslotte het antwoordblad met als bestandsnaam ‘Antwoordblad ISDJ *<jouw voor- en achternaam>*.docx’. Vul meteen op het eerste blad je naam en studentnummer in.
Op dat antwoordblad zet je alle diagrammen uit het UML model waaraan je in de opdrachten hebt gewerkt. Je vult daarin ook de overgangstabel in die je bij één van de opdrachten moet afmaken. Als je deze file in het IntellIJ project laat staan, komt dit bestand met het zippen ook mee.

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

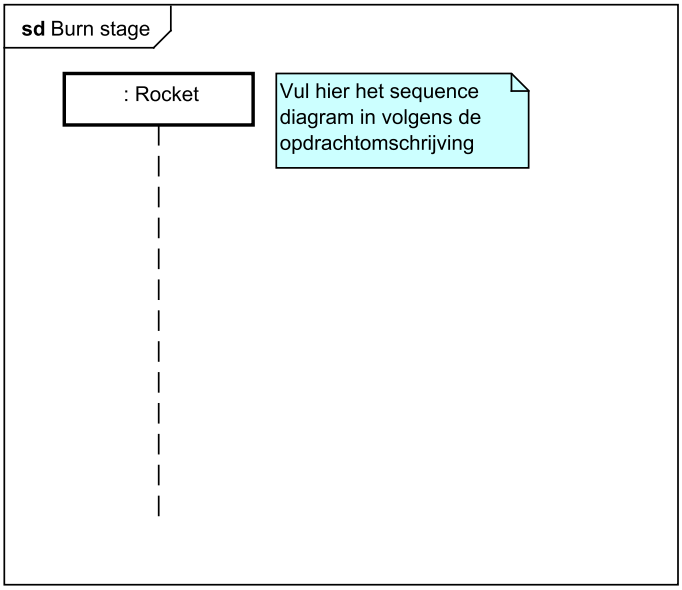
Opdrachten deel B

1. Wie ben je? 0 punten

Zet allereerst je eigen studentnummer in de klasse **`StudentNummer`**. Deze klasse staat al in het IntelliJ project.

2. Sequence diagram 'Burn stage' uitwerken 15 punten

In het sequence diagram 'Launch' wordt verwezen naar het sequence diagram 'Burn stage', waarmee het volledig opbranden van een stage gemodelleerd gaat worden.



In het project vind je dit nog grotendeels leeg sequence diagram.
De opdracht is om dit sequence diagram zodanig in te vullen dat de interactie tussen de betrokken objecten correct wordt weergegeven. In het diagram staat al een Rocket object van waaruit het eerste bericht wordt verstuurd.

In dit sequencediagram kiezen we als stage voor een SolidFuelStage om het burn proces mee te modelleren.
Een burn van een stage verloopt als volgt:

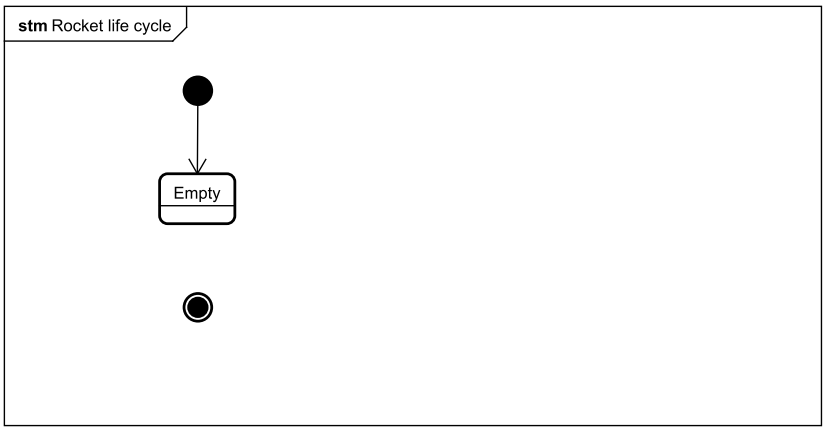
- Zolang de stage niet exhausted is wordt telkens een fire bericht naar de stage gestuurd om een volgende kleine stap in de simulatie uit te voeren. Onderdeel van de afhandeling van dat fire bericht is om het AcceleratingObject verder een klein beetje te versnellen. Echter, omdat de stage geen eigen referentie naar het AcceleratingObject heeft krijgt de stage als onderdeel van het fire bericht een verwijzing naar het AcceleratingObject mee. Rocket is zelf zo'n AcceleratingObject dus kan Rocket zichzelf meegeven.
- SolidFuelStage stuurt als afhandeling van het fire bericht aan het AcceleratingObject een increaseVelocity bericht om de velocity met 1 te verhogen.
- SolidFuelStage verlaagt vervolgens intern de hoeveel fuel.
- Als het fire bericht op deze wijze is afgehandeld voert de rocket een interne update uit om de hoogte en snelheid te updaten.
- Zoals aan het begin al gesteld: dit hele proces wordt herhaald zolang de stage niet exhausted is. Zodra de stage exhausted is, is de burn afgelopen.

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

3. Toestandsdiagram ‘Rocket life cycle’ afmaken 15 punten

In het project vind je een toestandsdiagram ‘Rocket life cycle’ dat de toestanden en toestandsovergangen van klasse **`Rocket`** modelleert.

In dat toestandsdiagram is een heel klein deel van het gedrag al gemodelleerd: de overgang naar de **Empty** toestand.



Modelleer alle toestanden, transities en acties die hieronder worden beschreven:

- Als in de **Empty** toestand een *launch* plaatsvindt wordt een exception gegooid en gaat het **`Rocket`** object naar de eindtoestand.
- Vanuit **Empty** is een transitie mogelijk naar de toestand **Assembled** zodra een *add* van een stage optreedt.
- In **Assembled** leidt elk volgende stage die toegevoegd wordt tot een overgang terug naar dezelfde toestand **Assembled**.
- Als de rocket **Assembled** is kan hij gelanceerd worden. Zo’n *launch* zorgt voor het opbranden van alle stages na elkaar en het bereiken van de **Exhausted** toestand.
- Een rocket die **Exhausted** is kan definitief weggegooid (*discarded*) worden, of hij kan *gereset* worden waarbij alle stages gewist worden. De rocket is dan weer **Empty** en klaar om er opnieuw stages aan toe te voegen.

4. Overgangstabel ‘Rocket life cycle’ afmaken 10 punten

Op het antwoordblad (Word document) vind je een gedeeltelijk ingevulde overgangstabel met de toestanden, events en transities van Rocket life cycle. Maak die overgangstabel af door de ontbrekende events, toestanden en transities in te vullen. Zorg dat ze kloppen met het toestandsdiagram ‘Rocket life cycle’.

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

In de volgende opdrachten ga je de Java code voor deze casus maken. Baseer je daarbij op wat het klassendiagram aan informatie biedt en verwerk dat zo volledig mogelijk in je Java code.

5. Klasse AcceleratingObject — 5 punten

Een **`AcceleratingObject`** stelt een object voor met een snelheid en een versnelling. Iedere simulatiestap wordt de versnelling bij de snelheid opgeteld, wordt wrijving en zwaartekracht (zeer versimpeld) toegepast.

Een korte beschrijving van de functionaliteit van de methoden:

- **`setVelocity(double velocity)`**
  deze methode zet de **`velocity`** op de waarde die als parameter is meegegeven
- **`increaseVelocity(double increment)`**
  deze methode verhoogt de **`velocity`** met de waarde die als parameter is meegegeven
- **`getHeight()`**
  deze methode geeft de waarde van **`height`** terug
- **`reset()`**
  deze methode initialiseert de attributen terug naar de beginwaarden
- **`void update()`**
  deze methode doet 1 stap in de simulatie van dit object. Gebruik hiervoor de volgende code:

```
this.height += this.velocity;
this.velocity -= 0.5;
```

Maak de klasse **`AcceleratingObject`** aan. Geef die klasse de attributen en methoden die aangegeven zijn in het klassendiagram. Implementeer de methoden zoals hierboven aangegeven.

6. Interface Stage — 5 punten

Een raket heeft een aantal verschillende motoren, ook wel stages genoemd. Een stage kan de raket versnellen, en is op een gegeven moment opgebrand.

Een korte beschrijving van de functionaliteit van de methoden:

- **`fire`**, deze methode geeft niets terug en heeft een **`AcceleratingObject`** als parameter
- **`isExhausted`**, deze methode geeft terug of deze stage opgebrand is of niet, en heeft geen parameters

Maak de interface **`Stage`** aan in overeenstemming met het klassendiagram en met de beschrijving die hier is gegeven.

Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

7. Klasse Rocket — 10 punten

Maak de klasse **`Rocket`** zoals in het klassendiagram staat aangegeven.
Een korte beschrijving van de functionaliteiten van de methoden van deze klasse:

- **`launch`**, deze methode gaat alle stages af, en roept per stage de **`burnStage`** methode aan. Als er geen stages in de raket zitten, laat deze methode dan een **`exception`** geven.
- **`burnStage`**, deze methode roept zolang de stage nog niet uitgebrand is, de **`fire`** methode van de stage aan en **`update`** hierna de waarden van de raket (zoals hoogte en snelheid). Denk hierbij ook aan het sequencediagram dat je eerder hebt gemaakt
- **`add`**, deze methode voegt een stage toe
- **`reset`**, deze methode zorgt dat de raket weer geïnitialiseerd wordt en de stages leeg zijn.

In de **`TestConsole`** klasse staat een stukje testcode om de functionaliteit te testen.

8. Fuel Stages — 10 punten

Maak de klassen **`LiquidFuelStage`** en **`SolidFuelStage`**. Deze klassen beginnen met een hoeveelheid brandstof die in de constructor wordt meegegeven.
De **`LiquidFuelStage`** brandt iedere keer dat **`fire`** aangeroepen wordt 10% van de huidige brandstof, en zet de snelheid op 10. Als er minder dan 1 liter brandstof over is, is deze stage exhausted.
De **`SolidFuelStage`** brandt iedere keer dat **`fire`** aangeroepen wordt 1 kilo brandstof op, en verhoogt de acceleratie met 1. Als de brandstof 0 is, is deze stage exhausted.

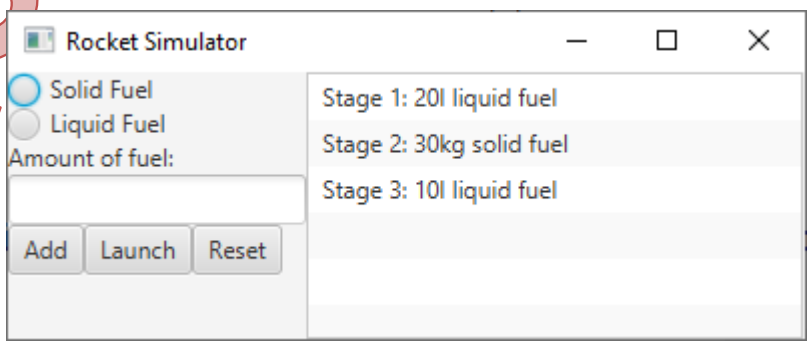
9. GUI — 20 punten

We gaan een user interface maken om deze raket te simuleren. In deze GUI kunnen we stages toevoegen, de raket lanceren en een nieuwe raket maken als deze opgebrand is.

Maak de klasse **`RocketGui`** om een JavaFX GUI te laten zien. Sla in deze klasse een **`Rocket`** op als attribuut, waar straks de gesimuleerde raket in wordt opgeslagen.

De layout

Voeg componenten en layoutmanagers toe om een layout te maken. Zorg ervoor dat de layout ongeveer zoals in de screenshot is.
Zorg dat maar 1 bolletje tegelijk geselecteerd kan worden (solid fuel en liquid fuel).



Elke student wordt geacht de bepalingen m.b.t. het afleggen van de toetsen te kennen.

Functionaliteit buttons

Implementeer de buttons volgens de use-case beschrijvingen. In de use-case beschrijvingen wordt gesproken over het tonen van een popup. Dit kun je doen met de volgende code voor een foutmelding:

```
new Alert(Alert.AlertType.ERROR, "message").showAndWait();
```

of voor een succesvolle melding:

```
new Alert(Alert.AlertType.INFORMATION, "message").showAndWait();
```

Je kunt hulpmethoden toevoegen aan de rest van het project om deze functionaliteit te maken, zoals bijvoorbeeld een **`toString`**-methode.

10. Klassendiagram ‘Stage Engine’ invullen 10 punten

In het project vind je een leeg klassendiagram ‘Stage Engine’. Modelleer in dat klassendiagram jouw uitbreiding op deze casus volgens onderstaande omschrijving.

We willen elke stage beter kunnen simuleren door ook de engine (raketmotor) van de stage op te nemen in het model. Zo’n engine kan ignited (ontstoken, gestart) worden en heeft een bepaalde thrust (stuwkracht), die we kunnen weergeven als een geheel getal.
De engines van een **`SolidFuelStage`** en een **`LiquidFuelStage`** zijn in bepaalde opzichten gelijk, maar verschillen in andere opzichten. De zojuist genoemde thrust is gemeenschappelijk, evenals het kunnen ontsteken van de engine. Verschillend is dat een liquid fuel engine een instelbare thrust heeft en tevens uitgezet kan worden. Een solid fuel engine heeft geen instelbare thrust en kan ook niet uitgezet worden.
Een solid fuel stage bezit precies 1 engine. Een liquid fuel stage heeft minstens 1 engine en kan tot maximaal 4 engines bezitten. Uiteraard is elke engine onderdeel van slechts 1 stage.