**EINDONTWERP**

|  |  |
| --- | --- |
| **Naam** | Vinay Mahadew |
| **Studentnummer** | 21022208 |
| **Klas** | 6 |

**OPDRACHT**

Geef hieronder een korte beschrijving van je opdracht (een samenvatting van je productvisie) zodat je docent OPT3 ook begrijpt met welke opdracht jij bezig bent (neem dit onderdeel over uit je laatst ingeleverde opdracht en pas het aan aan de hand van je eigen bevindingen en aan de hand van feedback van medestudenten en je docent aan):

Zie Casus op BlackBoard

**GITHUB**

Maak je GitHub-repo Private en geef je docent voor het werkcollege toegang tot je repo (vraag hem om zijn account als hij dat volgens jou nog niet aan jou gegeven heeft). Stuur niet steeds een nieuwe link/uitnodiging, maar bouw in feature-branches verder aan de applicatie die je eerder hebt opgeleverd:

https://github.com/VIMAH/opt3-2de-3de-opdr (in branch <Naam van de branch met de oplevering voor deze opdracht>)

Bovenstaande repo is m.b.v. .gitignore ontdaan van IntelliJ-files en er is een Maven POM-file toegevoegd waarmee je applicatie in één keer gerund kan worden.

**UML-ONTWERP (PORTFOLIO OPT2)**

Bij de oplevering van het ontwerp en de realisatie hieronder ben ik uitgegaan van de volgende User Stories (gebruik de ‘+’ rechts van de tabel om een User Story toe te voegen):

|  |
| --- |
| Als <rol> wil ik <wens> zodat ik <reden of doel>. |

Kopieer hieronder (een kopie van) het (aangepaste) UML-model dat je als resultaat hebt opgeleverd voor je portfolio-opdracht voor OPT2 (kopieer het model uit je vorige opdracht of kopieer het aangepaste model):

<Copy-Paste hier een kopie van het UML-ontwerp dat je hebt gemaakt voor OPT2 >

\* Controleer of het plaatje goed leesbaar is. Als je wilt dat je docent het ontwerp van OPT2 opnieuw beoordeelt, lever je de opdracht voor OPT2 opnieuw in als herkansing.

**EINDONTWERP**

Vanuit dit UML-ontwerp (dat je hebt gemaakt voor OPT2) heb je Patterns en SOLID-principes toegepast op je ontwerp en je hebt Classes toegevoegd die je nodig hebt om je applicatie te laten werken (zoals bijv. voor het tonen van een GUI of Command Line voor het tonen en afhandelen van een menu). Je hebt er kortom een technisch ontwerp van gemaakt waarin je ook de types van je properties, parameters en methodes hebt toegevoegd. Zie het volgende document voor voorbeelduitwerking en aanwijzingen (hiervoor moet je wel eerst inloggen in BlackBoard):

‘[OPT3 16.1.3.2 - Ontwerp en realisatie met patterns - Toelichting.pdf](https://blackboard.hhs.nl/bbcswebdav/xid-30272989_2)’

<Copy-Paste hier een kopie van je ontwerp (met patterns en SOLID)>

\* Controleer of het plaatje goed leesbaar is.

**PATTERNS**

Beschrijf hieronder hoe je het Template Method Pattern en een tweede Pattern hebt toegepast in bovenstaand eindontwerp:

|  |  |
| --- | --- |
| **Pattern** | **Beschrijving** |
| Template Method Pattern | Na(a)m(en) van template method (s): Producten  In Producten wordt de Huur prijs per dag en verzekering apart en gezamenlijk berekent. Daarnaast wordt er een boolean verhuurd op false gezet en de klant naam opgeslagen. Personenauto, VrachtAuto en Boormachine extenden Product, zodat deze 3 classes de methodes overerven van de abstract class Product. In de abstract class Producten zit de methode AllesIn(), in deze methode wordt bepaald of de verzekeringsprijs mee moet worden genomen in totale berekening of niet. |
| Abstract Factory Pattern | Naam van factory method: PersonenAutoFactory  Omdat we in PersonenAuto binnen dit project zullen vervangen door een andere PersonenAuto(Mercedes, Bmw of Audi) is de relatie voor het aanmaken van de PersonenAuto vanuit OverzichtController vervangen door een abstract PersonenAutoFactory. |
| Observer Pattern | Het Observer Pattern wordt gebruikt bij het verhuren van een product. Wanneer een klant een product een aantal dagen huurt, wordt de boolean gezet op true, hierdoor wordt in de HuurStatus() aangegeven dat het product niet verhuurd kan worden. |

**SOLID-PRINCIPES**

Beschrijf hieronder voor twee SOLID-principes hoe je die hebt toegepast in Bovenstaand eindontwerp:

|  |  |
| --- | --- |
| **SOLID-principe** | **Beschrijving (gebruik een spatie als je geen toelichting geeft)** |
| Single Responsibility | Elke class heeft zijn eigen taak/verantwoordelijk. Ik heb in mijn code de functionaliteiten verspreid in verschillende classes. Bijv. MercedesFactory maakt alleen personauto’s van het type Mercedes. |
| Open-Closed | Als ik een nieuw product wil toevoegen, hoef ik hiervoor alleen maar een nieuwe class aan te maken. En dit toe te voegen aan de code. Een nieuwe Porsche class toevoegen en die extends dan de class PersonenAuto’s. |
| Liskov | De class Producten is abstract. PersonenAuto, BoorMachine en VrachtAuto extenden de class Producten. Producten zou kunnen vervangen door de VrachtAuto of BoorMachine. |
| Interface Segragation | <Toepassing van het principe> |
| Dependency-Inversion | <Toepassing van het principe> |

**REALISATIE**

Als laatste onderdeel voor deze opdracht kopieer je hieronder de tweede versie van de Javacode waarmee je het eindontwerp realiseert (hoewel je de code smells al wel uit deze code mag verwijderen wordt dat pas beoordeeld bij de laatste opdracht):

https://github.com/VIMAH/opt3-2de-3de-opdr