**TESTONTWERP & -REALISATIE**

|  |  |
| --- | --- |
| **Naam** | Timo Brouwer |
| **Studentnummer** | 23055987 |
| **Klas** | 1 |

**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.

“Freezer" is een app die 65-plussers helpt hun vriezerinhoud te beheren door het scannen van producten en het uitprinten van stickers met QR-codes, namen, en houdbaarheidsdata. Dit vereenvoudigt het bijhouden van voorraden en vermindert voedselverspilling.

**GITHUB**

Maak je GitHub-repo **private**, geef je eigen docent toegang tot je repo (zie paragraaf 2 in [de uitleg bij deze opdracht op Brightspace](https://brightspace.hhs.nl/d2l/le/lessons/56739/topics/736360)) en lever je repo *clean* en Maven-*aware* op (zie paragraaf 3 in [de uitleg bij opdracht 2](https://dehaagsehogeschool.sharepoint.com/:b:/r/sites/AgileOOProgrammingSemesterSE-S2_groups-Docenten/Gedeelde%20documenten/Docenten/H-SE-S2OPT3/Lesmateriaal%20(publiek)/OPT3%2014.1%20-%20Realisatie%20zonder%20code%20smells%20-%20Toelichting.pdf?csf=1&web=1&e=MHo6uM)). Stuur niet steeds een nieuwe link/uitnodiging, maar bouw in feature-branches verder aan de applicatie die je eerder hebt opgeleverd:

https://github.com/TimoRoggel/OPT3.git (in branch: main)

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.

**GESLAAGDE TESTS**

Toon met een relevante screenshot aan dat alle onderstaande tests inderdaad zijn geslaagd:

Afbeelding met tekst, schermopname, software

Automatisch gegenereerde beschrijving

**SAMENGESTELDE DECISION**

In mijn software neem ik een besluit op basis van de volgende voorwaarden (maximaal 3 te testen):

Voorwaarde A: Het product is beschikbaar in de lijst van producten.

Voorwaarde B: Het product heeft een geldige naam.

Voorwaarde C: Het product heeft een geldige ID.

Het besluit is gebaseerd op de volgende combinatie van deze voorwaarden (bijv. D = A && (B || C)):

D = Het product kan worden true als het beschikbaar is in de lijst van producten en een geldige naam heeft, ofwel als het een geldige ID heeft.

**CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| true | true | true | true |
| true | true | false | true |
| true | false | true | true |
| true | false | false | false |
| false | true | true | false |
| false | true | false | false |
| false | false | true | false |
| false | false | false | false |
|  |  |  |  |

**DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| true | true | true | true |
| true | true | false | true |
| true | false | true | true |
| true | false | false | true |
| false | true | true | true |
| false | true | false | true |
| false | false | true | true |
| false | false | false | true |
|  |  |  |  |

**CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition/Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| true | true | true | true |
| false | true | true | False |
| true | false | true | true |
| false | false | true | False |
| true | true | false | true |
| false | true | false | False |
| true | false | false | true |
| false | false | false | False |
|  |  |  |  |

**MODIFIED CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Modified Condition/Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie). Geef met <l>/<f> aan met welke logische en fysieke test cases je werkt:

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| true | true | true | True <L> |
| false | true | true | False <L> |
| true | false | true | True <L> |
| false | false | true | False <L> |
| true | true | False | True <L> |
| false | true | false | False <L> |
| true | false | false | True <L> |
| false | false | false | False <L> |
|  |  |  |  |

**MULTIPLE CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Multiple Condition Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| Voorwaarde A | Voorwaarde B | Voorwaarde C | D = <Decision> |
| true | true | true | true |
| true | true | false | true |
| true | false | true | true |
| true | false | false | false |
| false | true | true | false |
| false | true | false | false |
| false | false | true | false |
| false | false | false | false |
|  |  |  |  |

**MODIFIED CONDITION/DECISION COVERAGE (JUNIT)**

Ik heb de testcases voor Modified Condition/Decision Coverage omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

**package** org.freezer;  
  
**import** org.junit.jupiter.api.Test;  
**import static** org.junit.jupiter.api.Assertions.\*;  
  
**public class** ProductDecisionTest {  
  
 **public class** Product {  
 **private boolean** isAvailable;  
 **private boolean** hasValidName;  
 **private boolean** hasValidId;  
  
 **public** Product(**boolean** isAvailable, **boolean** hasValidName, **boolean** hasValidId) {  
 **this**.isAvailable = isAvailable;  
 **this**.hasValidName = hasValidName;  
 **this**.hasValidId = hasValidId;  
 }  
  
 **public boolean** canBePrinted() {  
 **return** isAvailable && (hasValidName || hasValidId);  
 }  
 }  
  
 // Test cases voor Modified Condition/Decision Coverage  
 @Test  
 **public void** testTrueTrueTrue() {  
 Product product = **new** Product(**true**, **true**, **true**);  
 assertTrue(product.canBePrinted());  
 }  
  
 @Test  
 **public void** testTrueTrueFalse() {  
 Product product = **new** Product(**true**, **true**, **false**);  
 assertTrue(product.canBePrinted());  
 }  
  
 @Test  
 **public void** testTrueFalseTrue() {  
 Product product = **new** Product(**true**, **false**, **true**);  
 assertTrue(product.canBePrinted());  
 }  
  
 @Test  
 **public void** testTrueFalseFalse() {  
 Product product = **new** Product(**true**, **false**, **false**);  
 assertFalse(product.canBePrinted());  
 }  
  
 @Test  
 **public void** testFalseTrueTrue() {  
 Product product = **new** Product(**false**, **true**, **true**);  
 assertFalse(product.canBePrinted());  
 }  
  
 @Test  
 **public void** testFalseTrueFalse() {  
 Product product = **new** Product(**false**, **true**, **false**);  
 assertFalse(product.canBePrinted());  
 }  
  
 @Test  
 **public void** testFalseFalseTrue() {  
 Product product = **new** Product(**false**, **false**, **true**);  
 assertFalse(product.canBePrinted());  
 }  
  
 @Test  
 **public void** testFalseFalseFalse() {  
 Product product = **new** Product(**false**, **false**, **false**);  
 assertFalse(product.canBePrinted());  
 }

}

Hiermee test ik de volgende methode (van het type boolean, waarin de waarde van D = <Decision> wordt bepaald):

**public class** ProductDecisionTest {  
  
 **public class** Product {  
 **private boolean** isAvailable;  
 **private boolean** hasValidName;  
 **private boolean** hasValidId;  
  
 **public** Product(**boolean** isAvailable, **boolean** hasValidName, **boolean** hasValidId) {  
 **this**.isAvailable = isAvailable;  
 **this**.hasValidName = hasValidName;  
 **this**.hasValidId = hasValidId;  
 }  
  
 **public boolean** canBePrinted() {  
 **return** isAvailable && (hasValidName || hasValidId);  
 }  
 }

**EQUIVALENTIEKLASSEN EN RANDWAARDEN**

Voor het nemen van een besluit in de software heb ik een domein dat bestaat uit de volgende equivalentieklassen (maximaal 3 te testen):

|  |  |  |
| --- | --- | --- |
|  | Logische weergave | Besluit |
| Equivalentieklasse 1 | Product is beschikbaar en heeft een geldige naam of geldige ID | Product kan worden afgedrukt |
| Equivalentieklasse 2 | Product is beschikbaar, maar heeft geen geldige naam en geen geldige ID | Product kan niet worden afgedrukt |
| Equivalentieklasse 3 | Product is niet beschikbaar | Product kan niet worden afgedrukt |

Geef nu alle relevante fysieke testgevallen (incl. de meest intensieve test van randwaarden) en geef voor elk testgeval aan voor welke equivalentieklasse dit een test is (als je de laatste regels niet meer nodig hebt, kun je deze velden vullen met een spatie):

|  |  |  |
| --- | --- | --- |
| Equivalentieklasse | Fysiek testgeval | Besluit |
| 1 | isAvailable = true, hasValidName = true, hasValidId = true | true |
| 1 | isAvailable = true, hasValidName = true, hasValidId = false | true |
| 1 | isAvailable = true, hasValidName = false, hasValidId = true | true |
| 2 | isAvailable = true, hasValidName = false, hasValidId = false | false |
| 3 | isAvailable = false, hasValidName = true, hasValidId = true | false |
| 3 | isAvailable = false, hasValidName = true, hasValidId = false | false |
| 3 | isAvailable = false, hasValidName = false, hasValidId = true | false |
| 3 | isAvailable = false, hasValidName = false, hasValidId = false | false |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**EQUIVALENTIEKLASEN EN RANDWAARDEN (JUNIT)**

Ik heb de testcases voor Equivalentieklassen en rand waarden omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

**package** org.freezer;  
  
**import** org.junit.jupiter.api.Test;  
**import static** org.junit.jupiter.api.Assertions.\*;  
  
**public class** ProductDecisionTest {  
  
 **public class** Product {  
 **private boolean** isAvailable;  
 **private boolean** hasValidName;  
 **private boolean** hasValidId;  
  
 **public** Product(**boolean** isAvailable, **boolean** hasValidName, **boolean** hasValidId) {  
 **this**.isAvailable = isAvailable;  
 **this**.hasValidName = hasValidName;  
 **this**.hasValidId = hasValidId;  
 }  
  
 **public boolean** canBePrinted() {  
 **return** isAvailable && (hasValidName || hasValidId);  
 }  
 }

// Test cases voor Equivalentieklassen en Randwaarden  
@Test  
**public void** testAvailableWithValidNameAndValidId() {  
 Product product = **new** Product(**true**, **true**, **true**);  
 assertTrue(product.canBePrinted());  
}  
  
@Test  
**public void** testAvailableWithValidNameOnly() {  
 Product product = **new** Product(**true**, **true**, **false**);  
 assertTrue(product.canBePrinted());  
}  
  
@Test  
**public void** testAvailableWithValidIdOnly() {  
 Product product = **new** Product(**true**, **false**, **true**);  
 assertTrue(product.canBePrinted());  
}  
  
@Test  
**public void** testAvailableWithInvalidNameAndInvalidId() {  
 Product product = **new** Product(**true**, **false**, **false**);  
 assertFalse(product.canBePrinted());  
}  
  
@Test  
**public void** testUnavailableWithValidNameAndValidId() {  
 Product product = **new** Product(**false**, **true**, **true**);  
 assertFalse(product.canBePrinted());  
}  
  
@Test  
**public void** testUnavailableWithValidNameOnly() {  
 Product product = **new** Product(**false**, **true**, **false**);  
 assertFalse(product.canBePrinted());  
}  
  
@Test  
**public void** testUnavailableWithValidIdOnly() {  
 Product product = **new** Product(**false**, **false**, **true**);  
 assertFalse(product.canBePrinted());  
}  
  
@Test  
**public void** testUnavailableWithInvalidNameAndInvalidId() {  
 Product product = **new** Product(**false**, **false**, **false**);  
 assertFalse(product.canBePrinted());  
}

Hiermee test ik de volgende methode (waarin de besluiten op basis van de equivalentieklassen worden bepaald):

**public boolean** canBePrinted() {  
 **return** isAvailable && (hasValidName || hasValidId);  
}

**PAIRWISE TESTING**

Voor Pairwise Testing maak ik gebruik van de volgende parameters (op basis waarvan een besluit wordt genomen; vereenvoudig tot een set van 4 parameters die ieder 3, 2, 2 en 2 mogelijkheden hebben (anders wordt handmatig testen met Pairwise Testing veel te ingewikkeld; vul een spatie in, als je een optie niet wilt gebruiken):

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Optie 1** | **Optie 2** | **Optie 3** |
| isAvailable | true | false | true |
| hasValidName | true | false |  |
| hasValidId | true | false |  |
|  |  |  |  |

Na de tussenstappen in het proces heb ik de volgende testtabel met fysieke testgevallen opgesteld (vul spaties in, als er teveel testgevallen in de tabel staan voor jouw casus):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **isAvailable** | **hasValidName** | **hasValidId** |  | **Decision** |
| true | true | true |  | true |
| true | true | false |  | true |
| true | false | true |  | true |
| true | false | false |  | false |
| false | true | true |  | false |
| false | true | false |  | false |
| false | false | true |  | false |
| false | false | false |  | false |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**PAIRWISE TESTING (JUNIT)**

Ik heb de testcases voor Pairwise Testing omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

@Test  
**public void** testPairwise() {  
 Product product1 = **new** Product(**true**, **true**, **true**);  
 Product product2 = **new** Product(**true**, **true**, **false**);  
 Product product3 = **new** Product(**true**, **false**, **true**);  
 Product product4 = **new** Product(**true**, **false**, **false**);  
 Product product5 = **new** Product(**false**, **true**, **true**);  
 Product product6 = **new** Product(**false**, **true**, **false**);  
 Product product7 = **new** Product(**false**, **false**, **true**);  
 Product product8 = **new** Product(**false**, **false**, **false**);  
  
  
 assertTrue(product1.canBePrinted()); // Verwacht: true  
 assertTrue(product2.canBePrinted()); // Verwacht: true  
 assertTrue(product3.canBePrinted()); // Verwacht: true  
 assertFalse(product4.canBePrinted()); // Verwacht: false  
 assertFalse(product5.canBePrinted()); // Verwacht: false  
 assertFalse(product6.canBePrinted()); // Verwacht: false  
 assertFalse(product7.canBePrinted()); // Verwacht: false  
 assertFalse(product8.canBePrinted()); // Verwacht: false  
}

Hiermee test ik de volgende methode (waarin het besluit op basis van de meegegeven parameters wordt bepaald):

**public boolean** canBePrinted() {  
 **return** isAvailable && (hasValidName || hasValidId);  
}