Übungsblatt 5

Aufgabe 1

1. D: Wenn C wahr, dann ist A und ¬B wahr (logisches und)­­­­­­ E: A und ¬B sind wahr oder ¬C ist wahr (logisches oder)

(b)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | ¬A | ¬B | (¬A) ∨ B | ¬((¬A) ∨ B) | (A ∧ (¬B)) |
| W | W | F | F | W | F | F |
| W | F | F | W | F | W | W |
| F | W | W | F | W | F | F |
| F | F | W | W | W | F | F |
|  |  |  |  |  |  |  |

(c)

D: C ⇒ (A ∧ (¬B)) kann man umformen zu D: ¬C ∨ (A ∧ (¬B))

Boolean D = !paule.canViewSunsrise() || (paule.getDirection() = Direction.EAST && paule.frontIsClear();

(d)

Aufgabe 2

1. Strikte Operatoren sind kommutativ. Das heißt, dass es zum Beispiel keine feste Reihenfolge bei den Operatoren „und“ und „oder“ gibt, was dazu führt, das Programme abstürzen können. Die Lösung bilden die Semistrikten Operatoren, durch die eine Reihenfolge entsteht (erst A und dann B oder A oder sonst B).

Beispiel:

(b) (i)

E1: kann ausgewertet werden, da es nur eine Aussage/Bedingung ist E2: E3:

(ii)

E1: kann ausgewertet werden, da es nur eine Aussage/Bedingung ist E2: E3:

(c) Der Stilfehler befindet sich in „(ham.grainAvailable() && ham.frontIsClear()) == false“ da man Boolsche Ausdrücke nicht so negiert, sondern indem man ein „!“ davorsetzt. Verbesserung: „!(ham.grainAvailable() && ham.frontIsClear())“.

Aufgabe 3

1. Konstruktor:

public BankAccount (String name, Hamster hamster, Integer grainNumber, final Integer maximumOverdraft){}

Klasseninvariante: Hamster soll initialisiert sein, es soll einen nicht-leeren Namen geben, grainNumber darf nicht negativ sein, Überziehungslimit muss kleiner gleich 0 sein und darf sich nicht verändern.

(b) Es könnte sein, dass eine NullPointerException geworfen wird, wenn nicht geprüft wird ob, secondaryHamster null ist, bevor darauf zugegriffen wird. Um dieses Problem zu umgehen könnte man „Optional“ benutzen, um einen optionalen zweiten Besitzer hinzuzufügen und damit eine NullPointerException zu verhindern

(c) Vorbedingung: BankAccount ist initialisiert und es werden nicht mehr grains eingezahlt als der Hamster gerade im Mund hat

Nachbedingung: Der Betrag auf dem Konto wurde um die Einzahlung erhöht und der Hamster hat die eingezahlte Anzahl an Körnern weniger im Mund

(d) Vorbedingung: BankAccount ist initialisiert und der abhebende Betrag ist kleiner gleich der Kontostand des Hamsters

Nachbedingung: Der Kontostand verringert sich um den abgehobenen Betrag und der Hamster hat die Anzahl des abgehobenen Betrags als Körner im Mund

(e) (i)

this: Wird verwendet, um auf das aktuelle Objekt zu verweisen, auf das eine Methode aufgerufen wird. null: Stellt dar, dass eine Variable keine gültige Referenz auf ein Objekt oder keinen gültigen Wert enthält.

(ii) Die Abfrage wird den Wert „40“ zurückgeben, da secondAccount und firstAccount durch Zeile 3 auf dasselbe Objekt zeigen (das, auf dass zuerst nur firstAccount gezeigt hat).

(iii) Damit ein Ausruf eine NullPointerException auswirft, müsste toBankAccount nicht initialisiert sein.

Bsp:

BankAccount fromBankAccount = new BankAccount (String name, Hamster hamster, Integer grainNumber, final Integer maximumOverdraft);

BankAccount toBankAccount = new BankAccount ();

Da toBankAccount nicht richtig initialisiert wurde, können keine Grains mit der Operation toBankAccount.depositGrains(grains) übertragen werden und es würde die NullPointerException geworfen werden, da die Entität toBankAccount zur Laufzeit auf kein Objekt zeigt bzw. auf kein Objekt referenziert wird.

(iv) Ja, da man im Code erkennen kann, ob null als Wert genutzt wird oder nicht. Wenn null genutzt wird kann man davon ausgehen, dass eine NullPointerException geworfen wird, sobald das Objekt zur Laufzeit aufgerufen wird.