Übungsblatt 9

Aufgabe 1

1. Der statische Typ von Paule in Zeile 6 ist Hamster, da Paule mit dem Typ Hamster initialisiert wurde und sich der statische Typ zur Laufzeit nicht ändert. Die möglichen dynamischen Typen von Paule sind Hamster (da dies die erste Zuweisung für Paule ist) oder RotatingHamster (da das der dynamische Typ von hamster ist und Paule durch Zeile 6 möglicherweise auf dasselbe Objekt referenziert. Der dynamische Typ von Paule wird zur Laufzeit entschieden, sobald die Operation Math.random() (Zeile 5) ausgewertet wird.
2. Der Code kann so nicht kompilieren, da die Operation paule.move(5) (Zeile11) nur ausgeführt werden kann, wenn Paules dynamischer Typ wirklich RotatingHamster ist, da diese move-Funktion nur in RotatingHamster implementiert ist. Es handelt sich um einen statisch semantischen Fehler, da alle Bezeichner den Regeln entsprechen (lexikalisch), die Syntax-Regeln eingehalten sind (syntaktisch), der code allerdings bereits zur Kompilierzeit schon „keinen Sinn“ ergibt (statisch semantisch) und nicht erst zur Laufzeit (dynamisch semantisch). Damit der Code kompiliert und trotzdem den gewünschten Effekt umsetzt, könnte man die in RotatingHamster implementierte move-Funktion in die Hamster Klasse schreiben und somit die move-Funktion überladen. Dies geht, da sich beide Funktionen über ihre Parameter unterscheiden und eine solche Funktion ebenfalls für alle Hamster sinnvoll wäre. // oder moveNSteps implementieren lol. In Zeile 10 muss nichts verändert werden, da sich paule unabhängig vom dynamischen Typen am Ende nach links drehen wird und somit die Funktionalität erhalten bleibt.
3. Dynamisches Binden in der Vererbung garantiert, dass für jeden Operationsaufruf x.f( ) die spezialisierteste Version von f verwendet wird. Ein Beispiel dafür findet sich in Zeile 10. Sollte paule zur Laufzeit auf dasselbe Objekt wie hamster zeigen, also auch vom dynamischen Typ RotatingHamster sein, so wird in Zeile 10 die turnLeft-Funktion von RotatingHamster aufgerufen.

Bei polymorphen Zuweisungen kann eine Entität oder ein Ausdruck eines bestimmten Typs auch auf Objekte der Unterklassen referenziert werden. Ein Beispiel dafür findet sich in Zeile 3. Hier zeigt die Objektvariable hamster auf eine Instanz von RotatingHamster.

1. Das möglicherweise Auftreten Problem ist das Diamantenproblem. Haben wir zum Beispiel die Klassen A, B und C, wobei B und C jeweils von A erben. Wenn wir nun eine Klasse D hinzufügen, die jeweils von B und C erben, kann es sein, dass es funktionen f( ) gibt, die bereits in B und C unterschiedlich implementiert wurden und somit nicht eindeutig klar ist, welche Implementierung D nutzen soll.

Aufgabe 2

1. Die Operation move ist so implementiert, zunächst eine zufällige Zahl zwischen 0 und 1 generiert wird. Ist diese Zahl kleiner als 0.7 führt ein LazyHamster die move Operation eines normalen Hamsters aus. Ist die Zahl größer gleich 0.7 so sagt der LazyHamster, dass er zu faul ist, um sich zu bewegen. Zusammengefasst führt ein LazyHamster die move Operation eines Hamsters mit einer Wahrscheinlichkeit von 70% aus. Mit einer 30%igen Wahrscheinlichkeit sagt er, dass er zu faul sei.

Vorbedingung: identical to move in Hamster.

Nachbedingung: LazyHamster either moved or said that he was too lazy.

1. Die Operation move ist so implementiert, dass ein CheatingHamster bei jedem Aufruf von move zweimal die move Operation von Hamster ausführt.

Vorbedingung: identical to move in Hamster but the frontIsClear( ) refers to 2 tiles in front of the hamster.

Nachbedingung: CheatingHamster moved 2 tiles.

1. Im Sinne der Vererbung müssen die Vorbedingungen der Überschriebenen Operationen beibehalten oder abgeschwächt worden sein. Im Fall vom LazyHamster ist dies erfüllt. Beim CheatingHamster allerdings nicht, hier fordert unsere Vorbedingung, dass 2 tiles vor dem Hamster frei sein müssen, was eine Verschärfung der Vorbedingung darstellt, wodurch die Vererbung nicht korrekt ist.

Die Nachbedingungen müssen entweder beibehalten werden oder verstärkt werden. Dies führt beim LazyHamster zu einem Problem, da sich der Hamster nicht zu 100% bewegt, wodurch die Nachbedingung von move aus Hamster nicht beibehalten wird.

1. final Hamster paula = new CheatingHamster();  
   paula.move();

Hier könnte die move Operation von CheatingHamster Probleme verursachen. Paula ist vom statischen Typ Hamster. Wenn nun jemand auf den Hamster Paula move aufruft, stellt der Nutzer sicher, dass die Vorbedingungen von move() aus der Klasse Hamster erfüllt sind. Da Paula allerdings vom dynamischen Typ ein CheatingHamster ist, der eine stärkere Vorbedingung als Hamster fordert, kann es sein, dass ein Aufruf vom move auf Paula zwar korrekt ist, Paula allerdings nicht 2 Felder laufen kann, da sich auf dem 2. Feld eine Wand befindet, wodurch die Nachbedingung nicht erfüllt werden kann.