Exercises

W03H03 - Arktischer Fischhandel

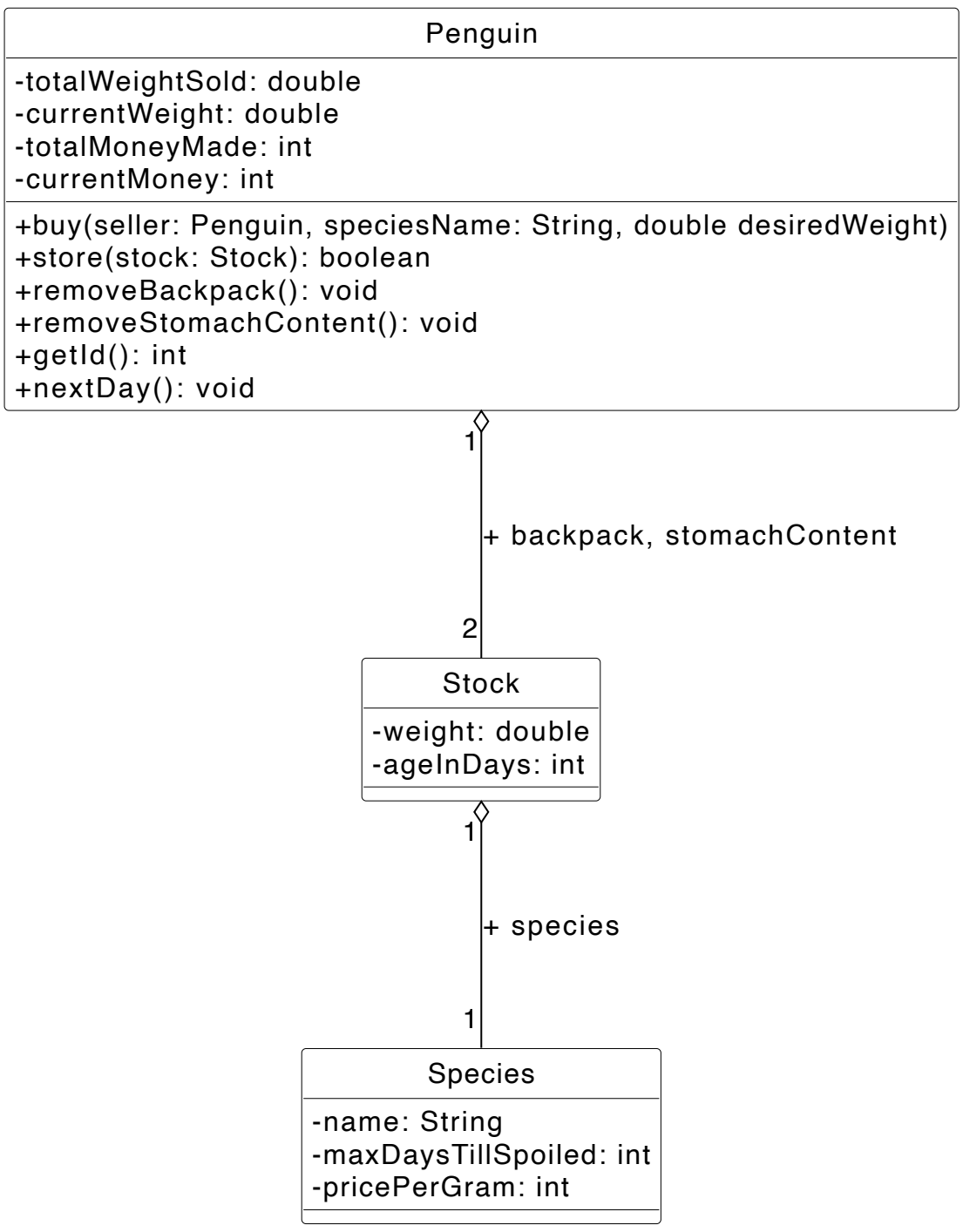
Points	Submission due date	Status	Difficulty	Categories
0 / 10	Nov 23, 2025 17:00	No graded result		Hausaufgabe

Tasks:

Arktischer Fischhandel

Ziel dieser Aufgabe ist es, zu lernen wie gute Testfälle ausgesucht werden, um Software umfangreich zu testen. Dafür schreibst du Tests für die Klasse `Penguin`.

Angespornt durch den Erfolg der Saleuine Clara und Carl-Heinz boomt der arktische Fischhandel. Nicht nur große Unternehmen beteiligen sich, auch privat wird viel gefischt und in Teilen verkauft. Damit alles mit rechten Dingen zugeht, haben Gesundheits- und Finanzamt strenge Vorschriften erlassen, wie die Handelsaktivität dokumentiert werden muss. Einige Pinguine sind sich noch unsicher, ob sie alles korrekt umsetzen und haben Dich deshalb gebeten, ihre Buchführung zu testen.



Jeder Pinguin kann seine Fischvorräte (`Stock`) auf zwei verschiedene Arten transportieren: entweder in seinem Magen, wie sie es schon seit Jahrtausenden gemacht haben, (`getStomachContent()`) oder in seinem Rucksack (`getBackpack()`), eine Innovation, die sich schnell durchgesetzt hat. Wenn ein Pinguin keinen Vorrat in Rucksack oder Magen hat, soll das entsprechende Attribut auf `null` gesetzt sein.

Damit nichts durcheinanderkommt, bewahren die Pinguine immer nur Fische einer einzigen Spezies (`Species`) an einem Ort auf und mischen sie nie. Pro Fischvorrat wird das Gewicht gespeichert (`Stock.getWeight()`) und um den Überblick zu behalten, speichert jeder Pinguin noch das Gesamtgewicht seiner Fischvorräte (`Penguin.getCurrentWeight()`). Alle Gewichte in dieser Aufgabe sind in Gramm gemessen. Außerdem speichert jeder Pinguin wie viel Geld in PinguDollar er zur Verfügung hat (`Penguin.getCurrentMoney()`).

Um seine Geschäfte besser zu überblicken, muss jeder Pinguin außerdem Buch darüber führen, wie viel Geld er bereits durch Verkäufe eingenommen hat (`Penguin.getTotalMoneyMade()`) und wie viel Fisch er dafür verkauft hat (`Penguin.getTotalWeightSold()`). Eigene Einkäufe ändern an diesen Werten nichts.

Um sicherzustellen, dass die Fische frisch sind, müssen die Pinguine auch Buch über das Alter ihrer Waren führen (`Stock.getAgeInDays()`). Die Ware muss aussortiert werden, wenn das Alter die Haltbarkeit der Spezies überschreitet (`Species.getMaxDaysTillSpoiled()`), spätestens aber nach hundert Tagen. Vorräte verschiedenen Alters dürfen zusammengelegt werden, dabei dürfen Fische aber nie als jünger deklariert werden, als sie tatsächlich sind (d. h. wenn z. B. 5 Tage alter und 7 Tage alter Lachs zusammengelegt werden soll, muss das Resultat als 7 Tage alt deklariert werden).

Selbstverständlich dürfen Pinguine den Namen oder die Haltbarkeit einer Spezies nicht ändern. Sie können sich allerdings frei entscheiden zu welchem Preis (`Species.getPricePerGram()`) sie die Fische anbieten wollen. Das heißt insbesondere auch, dass Pinguine zwei Fischvorräte mit unterschiedlichem Preis, aber ansonsten gleicher Spezies immer zusammenlegen können, in dem sie einen der Preise ändern.

Die Pingu-Behörden wissen, dass es beim Hantieren mit Fließkommazahlen zu Rundungsfehlern kommen kann, und, dass auch Messwerte nicht beliebig genau sein können. Deshalb gibt es gewisse Toleranzen: Gewichte müssen nur bis auf ein Milligram genau sein. Wenn auf Basis des Gewichts der Preis der Ware berechnet werden soll, wird immer zu Gunsten des Kunden gerundet.

Die Klassen `Species` und `Stock` speichern nur Daten, Du musst nicht testen, ob sie richtig funktionieren. Du musst nur testen, ob die Implementierung von `Penguin` den Anforderungen entspricht. Die Methoden in `Penguin` sind im folgenden relativ knapp beschrieben. Überlege Dir selbst, welche der oben beschriebenen Anforderungen die einzelnen Methoden verletzen könnten

`Penguin.store(stock)`

Diese Methode dient zum Einlagern von selbst gefangenem Fisch. Der Pinguin soll die gefangenen Fische, wenn möglich in seinem Rucksack, sonst in seinem Magen aufbewahren. Wenn der Pinguin bereits Fische der gleichen Spezies transportiert, soll er die neuen Fische mit den bereits vorhandenen zusammen aufbewahren. Wenn kein Platz mehr für die neuen Fische ist, soll nichts passieren und die Methode soll `false` zurückgeben, um das Problem anzuzeigen. Versucht man abgelaufene Fische oder einen leeren Vorrat einzulagern, soll nichts passieren und die Methode ebenfalls `false` zurückgeben.

`Penguin.removeBackpack()`, `Penguin.removeStomachContent()`

Diese Methoden dienen dazu einen der Vorratsorte zu leeren, um Platz für neue Fische zu machen.

`Penguin.buy(seller, speciesName, desiredWeight)`

Mit dieser Methode wird das Handeln zwischen den Pinguinen umgesetzt. Dabei soll der gekaufte Fisch vom Verkäufer an den Käufer und das nötige Geld vom Käufer and den Verkäufer übergehen. Außerdem müssen alle oben beschriebenen Bilanzen richtig angepasst werden. Wenn möglich sollen dabei Fischbestände gleicher Spezies beim Käufer zusammengelegt werden. Wenn der Käufer zu wenig Geld oder der Verkäufer zu wenig Ware hat, um den Handel durchzuführen, soll entsprechend weniger verkauft werden. Falls überhaupt kein Handel zustandekommen kann, dann soll diese Methode `false` zurückgeben, um das Problem anzuzeigen.

`Penguin.getId()`

Jeder Pinguin soll zur Identifikation eine eindeutige und unveränderliche ID haben, die mit dieser Methode erfragt werden kann.

`Penguin.nextDay()`

Diese Methode informiert den Pinguin darüber, dass ein weiterer Tag vergangen ist. Dann müssen die Pinguine das Alter ihrer Ware updaten, und prüfen, ob sie aus dem Sortiment genommen werden muss.

Für diese Aufgabe sollst du die Klasse `Penguin` testen, indem du die vorgegebene Klasse `PenguinTest` ergänzt, also neue Testmethoden hinzufügst. In `PenguTest` ist bereits die Methode `newPenguin()` vorgegeben. Erstelle Pinguine nur mit dieser Methode, *niemals* mit einem Konstruktoraufruf `new Penguin()`, ansonsten kann Artemis deine Lösung nicht auswerten.

Außerdem ist *eine* Implementierung von `Penguin` vorgegeben, damit Du Deine Tests ausprobieren kannst. Die Implementierung ist im Großen und Ganzen richtig, enthält aber einige kleine Fehler. Beachte, dass andere korrekte Implementierungen denkbar sind, auch wenn die gegebene Implementierung einen bestimmten Punkt korrekt umsetzt.

Wichtige Hinweise zu den Tests:

Du musst alle Deine Tests mit der Annotation `@Test` markieren, andere Annotationen wie `@ParameterizedTest` werden **nicht bewertet!**

Du darfst eine oder mehrere Methoden mit der Annotation `@BeforeEach` markieren. Diese Methoden werden dann vor der Ausführung jedes Testfalls ausgeführt. Das kann hilfreich sein, wenn Du immer dasselbe Setup für verschiedene Testfälle benötigst. Andere Annotationen wie `@BeforeAll` o.ä. werden bei dieser Aufgabe **nicht ausgeführt!**

- **Tipp:** Jede Testmethode sollte nur einen einzelnen Testfall abdecken. Das hilft dir, Fehler leichter zu finden. Außerdem separiert das die Testfälle, sodass Fehler in einem Testfall die anderen nicht beeinflusst.
- **Tipp:** Du kannst dir Hilfsmethoden in `PenguinTest` definieren, die die Tests übersichtlicher machen. Hilfreich könnten z. B. Methoden wie `assertStockEquals`, `assertSpeciesMergable` sein, um auf einen Schlag `Stock`- und `Species`-Objekte testen zu können.

1. ⦿ **Richtige Implementierung** No results
   Fehlerhafte Tests, die z.B. eine korrekte Implementierung als falsch einstufen, werden aussortiert und zählen nicht in die Bewertung.
2. ⦿ **Falsche Implementierungen** No results
   Hier wird getestet, ob Deine Tests bei allen falschen Implementierungen erkennen, dass sie falsch sind.

Exercise details

Release date	Nov 11, 2025 07:00
Start date	Nov 11, 2025 07:05
Submission due date	Nov 23, 2025 17:00
Complaint possible	No