Monsieur Corintecakère von AixCuisine hat sich beschwert, dass nicht konstruierbare Dreiecke durch fehlerhafte Nutzereingaben angelegt werden und danach die Software unsinnige Ergebnisse liefert. Aix-Car-Group Ltd. hat gemeldet, dass die Software zwar Fehler beim Anlegen von Fahrzeugen meldet, aber trotzdem unvollständige Fahrzeugobjekte erzeugt werden, die weitere Fehler produzieren.

Es ist Zeit, die Software ein bisschen zu verbessern, ohne viel neue Funktionalität hinzuzufügen. Ihre Mitarbeiterin Emma verrät Ihnen, dass dieser Prozess "Refactoring" genannt wird.

Ergebnis des letzten Design-Meetings

Refactoring:

* Es soll ein Fehlerabfang für nicht konstruierbare Dreiecke eingebaut werden.
* Statt Rückgaben von boolean-Werten sollen Exceptions genutzt werden, um Fehler zu behandeln.
* Es soll, entsprechend der Vorgaben der Aix-Car-Group Ltd., über geeignete Container für den Fuhrpark und die Mitarbeiter entschieden werden.

Eine neue Funktionalität soll hinzugefügt werden: Sortierung der Mitarbeiter und Fahrzeuge nach unterschiedlichen Kriterien.

Anwesend: Emma Kläwer, Klara Schwaygsam und Mattis Grünohr

STATIC

**Aufgabe 1**

1. Wann sind Dreiecke nicht konstruierbar? Diskutieren Sie in Partnerarbeit darüber, warum normale set-Methoden, die normalerweise für den Fehlerabfang zuständig sind, in der Klasse Dreieck nicht verhindern können, dass ungültige Dreiecke erzeugt werden.
2. Wenn Dreiecke verändert werden, benötigt man diese Funktionalität auch außerhalb des Konstruktors. Emma Kläwer spricht immer vom DRY-Prinzip (don't repeat yourself) bei der Programmierung. Wie kann man diese Funktionalität am besten wiederverwendbar machen?
3. Braucht man für die in B) konzipierte Methode ein Objekt? Informieren Sie sich über das Schlüsselwort static. Nutzen Sie dafür die **beigelegten Kontrollfragen.**
4. Ändere die Klasse Dreieck, so dass eine objektunabhängige Funktionalität zur Überprüfung eingefügt wird.

COLLECTIONS UND GENERICS

**Aufgabe 2**

Geeignete Container sollen in die Software der Aix-Car-Group Ltd. eingebaut werden. Am BWV gibt es eine Schulung zu verschiedenen Containern in Java. Informieren Sie sich und überprüfen Sie ihr Wissen anhand der Kontrollfragen.

**Aufgabe 3**

Entscheiden Sie sich anhand der Vorgaben der Aix-Car-Group Ltd. für geeignete Container. Bereiten Sie sich darauf vor, Ihre Entscheidung zu verteidigen.

Auszug aus dem Lastenheft:

* Abteilungen verwalten die Mitarbeiter immer sortiert nach Namen. Eine doppelte Zuordnung ist verboten. Sortiertes Einfügen soll automatisch erfolgen. Die Mitarbeiter müssen aber nicht nachträglich anders sortiert werden können.
* Die Mitarbeiterverwaltung soll Mitarbeiter nach Namen und Einkommen sortieren können. Durch die ständige Umsortierung sind viele Zugriffe notwendig.
* Fuhrpark soll nach Auslastung sortieren können. Auslastung ändert sich laufend.
* Die Anzahl der eingesetzten Schichtarbeiter unterliegt starken Schwankungen. Der Container für Schichtarbeiter sollte also bezüglich der Anzahl der verwalteten Elemente effizient vergrößert und verkleinert werden können. Außerdem soll nach gearbeiteten Stunden sortiert werden können
* Die Lagerverwaltung (zukünftige Erweiterung) soll sehr schnell beim Zugriff sein und auf jeden Fall vermeiden, dass Güter doppelt eingefügt werden. Eine Sortierung ist nicht erforderlich.

**Aufgabe 4**

Führen Sie die beschlossenen Änderungen für die Container in Ihrer Software durch. Ausgenommen ist die zukünftige Lagerverwaltung.

INTERFACES

**Aufgabe 5**

Die Mitarbeiter sollen sortiert werden. Dafür muss man die Objekte vergleichen können. Klara gibt Ihnen den Tipp, dass Sie dafür das Java-Interface Comparable nutzen und damit Ihre ArrayList vom Typ Mitarbeiter sortieren können. Aber was hat es mit den Interfaces auf sich? Informieren Sie sich mit Hilfe einer Schulung über Interfaces.

**Aufgabe 6**

Lassen Sie von der Mitarbeiter-Klasse das Interface Comparable implementieren. Es soll nach den Namen der Mitarbeiter verglichen werden. Nutzen Sie anschließend die Methode Collections.sort(), um Ihre Mitarbeiter zu sortieren.

**Aufgabe 7**

Damit nach schlecht ausgelasteten Fahrzeugen gesucht werden kann, sollen Sie Fahrzeuge nach Auslastung vergleichen können. Lösen Sie das Problem.

Testen Sie Ihre Lösung durch eine wiederholte zufällige Beladung und Entladung der LKWs bevor sie sortieren.

**Aufgabe 8**

Des Weiteren sollen Mitarbeiter auch nach Einkommen sortiert werden können. Sie merken, dass man ein Interface nicht zweimal implementieren kann. Klara gibt Ihnen wieder einen Tipp: Es gibt ein weiteres Interface Comparator. Allerdings kann das nicht die Klasse Mitarbeiter selber machen. Implementieren Sie das gewünschte Interface als eigenständige Klasse. Nutzen Sie **die überladene sort-Methode** der Klasse Collections, welche eine List und ein Comparator Objekt erwartet.

T

|  |
| --- |
| <<interface>> Comparator |
| +compare( T obj1, T obj2 ) |

Da die neue Klasse MitarbeiterComparator nur für die Mitarbeiter da ist, kann man sie auch als **innere static Klasse** in die Klasse Mitarbeiter schreiben. Die Instanziierung erfolge dann mit:

Mitarbeiter.MitarbeiterComparator mc = new Mitarbeiter.MitarbeiterComparator();