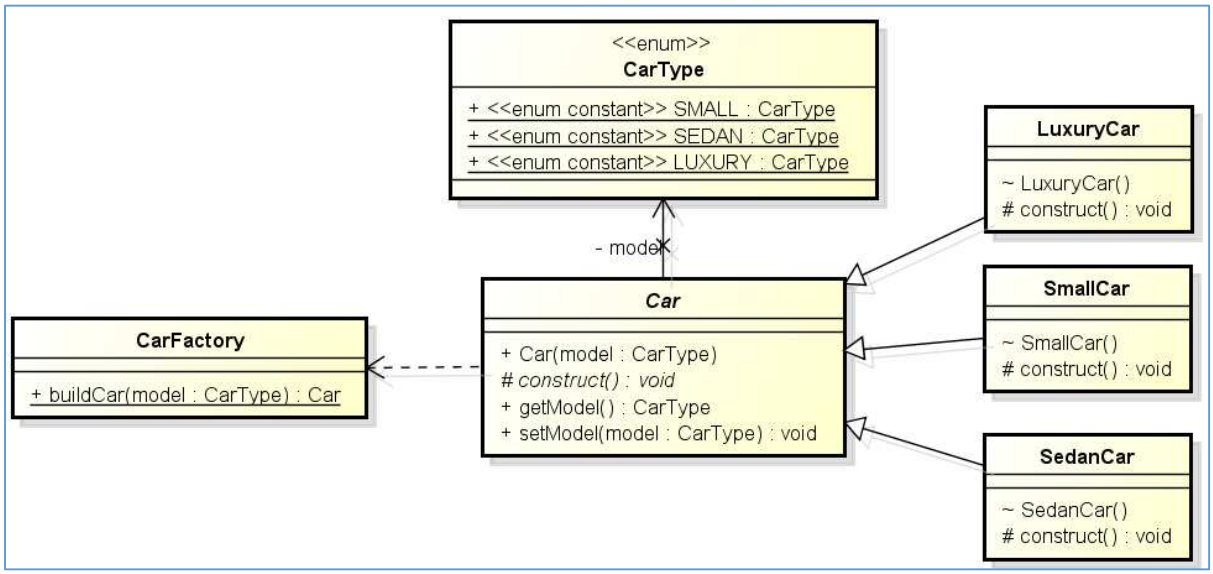
Einleitungsstory:

„So vermehrt sich das gemeine Minion: Sie werden als beschworene Wesen immer dann geschaffen wenn sich ein Minion über eine faulige Banane ärgert und sich jemanden wünscht, an dem es das auslassen kann. Dann teilt sich das wütende Original und erschafft so eine Kopie von sich selbst. Das erklärt auch, warum sie kein Problem damit haben, sich gegenseitig zu hauen – sie tun ja niemand anderem weh.“ (*Prof. J.R.R. Pfitzelhuber, 1985 vor Kirk, „Der Boogeyman und andere Ungeheuer“)*

Als Experte für unkonventionelle Lösungen haben Sie sich bereits einen soliden Ruf erarbeitet, als eines Tages Edith, Adoptivtochter des Superschurken Gru an Ihre Tür klingelt. Da Gru und sein Wissenschaftler Dr. Nefario es nicht so mit Ordnung haben, quillt die Villa von Gru inzwischen über vor kleinen gelben Minions. Edith möchte Sie engagieren, um die Minions zu zählen und zu ermitteln, wie viele frische Bananen pro Tag gebraucht werden, damit sie sich nicht weiter vermehren. Da die Minions sich bei spektakulären Industrieunfällen immer mal selbst reduzieren, sollte sich so das Platzproblem auf Dauer lösen lassen.

Aufgabe 1 - MinionFactory

Dr. Nefario hat schon mal eine Minion-Klasse angelegt, die die wichtigsten Attribute enthält. Das reicht Ihnen aber nicht aus: Ihnen hat vor kurzem erst ein Kumpel beim Biertrinken das Factory-Pattern auf eine Serviette gemalt und jetzt wollen sie es unbedingt anwenden. Seine Zeichnung bezieht sich leider auf Autos. Fertigen Sie eine neue Zeichnung des Patterns für die Minions an und setzen Sie sie um. Für den MinionType orientieren Sie sich dabei an der Farbe, und für die Umsetzung von „construct“ reicht eine einfache Textausgabe. Finden Sie eine Lösung dafür, die evilNumber an die Sub-Klassen zu übergeben. Beachten Sie auch Dr. Nefarios Kommentare zu den Attributen. Erklären Sie, welche Vorteile das Factory-Pattern hat.



[Antwort: in der CarFactory wird in der Methode buildCar überprüft, ob nur Objekte instanziiert werden, die auch den Vorgaben zu CarType entsprechen. Car hat den Vorteil der Vererbung und ermöglicht die Implementierung gleicher Methodenvorgänge für alle Unterklassen. Mittels #construct() können aber in den Unterklassen für gleiche Arbeitsaufträge unterschiedliche Implementierungen gemacht werden. Damit sind die Klassen felxibel und garantieren eine gewisse Stabilität in der Programmierung.]

Aufgabe2 – Von Daten zur Liste

Dr. Nefario hat die Minions schon gezählt. Da die aber nicht stillhalten wollen, sind ihm Duplikate untergekommen und ungeordnet sind sie auch. Sie finden die Zählung in seinem „EvilPlan“. Passen Sie seine Zählung auf Ihre MinionFactory an. Da Sie gute Programmierung gelernt haben, ist Ihnen auch klar, dass Sie die Minion-Klasse weiter überarbeiten müssen. Überschreiben Sie hashCode() und equals() und machen Sie die Klasse immutable.

Übertragen Sie Dr. Nefarios Datensätze in eine Liste und nutzen Sie dabei das Comparable-Interface, um sie nach ihrer evilNumber zu sortieren. Was könnte man tun, um die Duplikate aus der Liste zu entfernen?

[Antwort: Man müsste für jedes Objekt in der Liste prüfen, ob es in der Liste noch weitere identische gibt. Dazu müsste man beispielsweise ein Objekt suchen und über alle weiteren iterieren und Duplikate dabei löschen.]

Aufgabe3 – Was sind die Unterschiede?

Irgendwie ist Ihnen das Entfernen der Duplikate mit der Liste zu aufwendig. Sie wissen, dass das einfacher geht: Erklären Sie den Unterschied zwischen TreeSet und HashSet.
Warum kann man nicht einfach *Set<Minion>= new Set<>();* verwenden?

[Antwort1: HashSet implementiert nur das Interface „AbstractSet", während TreeSet noch „SortedSet" implementiert. HashSet stellt sich dabei intern als Hash-Tabelle dar, wodurch die grundlegenden Operationen sehr schnell werden. Die Datensätze sind aber nicht geordnet. HashSet verwendet für jegliche Vergleiche equals(). TreeSet ist als Binärbaum strukturiert, wodurch die grundlegenden Operationen verhältnismäßig langsam werden. Es hat aber den Vorteil, dass die Daten von vorn herein geordnet sind. TreeSet verwendet compareTo()]

[Antwort2: Weil new Set<>() nicht funktioniert. Set ist ein Interface, das unter Anderem von den Klassen TreeSet und HashSet implementiert wird und es ist nicht möglich, ein Objekt aus einem Interface zu instanziieren.]

Aufgabe 4 – Von Liste zu Ordnung

Sie entscheiden sich für ein TreeSet. Edith erklärt Ihnen, dass die violetten Minions keine Bananen brauchen, da sie sich von Möbelstücken ernähren. Übertragen Sie nur die Daten der gelben Minions von der Liste in ein TreeSet und erklären Sie, was mit den Duplikaten passiert. Schreiben Sie die endgültige Liste in eine .txt-Datei, die Edith als Checkliste für die Bananen-Ausgabe verwenden kann, nachdem Sie toString() angepasst haben.

[Antwort: Wenn ein Datensatz in TreeSet eingefügt wird, prüft die Implementierung von TreeSet, ob es schon ein gleichartiges Objekt gibt. Dazu verwendet es compareTo(). Wenn schon so ein Objekt existiert, wird das aktuelle Objekt nicht hinzugefügt.]

Die Ausgabe könnte so aussehen: `Minion [ Nr. 1 – gelb – braucht Banane ]`
Geben Sie die Anzahl an benötigten Bananen pro Tag an. *[Antwort: 44 Bananen]*

Unit-Tests

Überprüfen Sie die Funktionalität Ihrer Implementierung mit entsprechenden JUnit-Tests und weisen Sie mit diesen Tests nach, dass die implementierten Operationen richtig funktionieren.

Achtung

Bitte beachten Sie folgendes, damit es nicht zu unnötigen Punktabzügen in der Bewertung kommt:

- Benennen Sie Klassen und Methoden konsistent und verständlich. Klassen sollten Nomen als Namen, Methoden Verben haben.
- Dokumentieren Sie alle Klassen, Interfaces, Konstruktoren und Methoden mit JavaDoc. Dokumentieren Sie auch alle Parameter, Rückgabewerte und Ausnahmen.
- Formatieren Sie Ihren Code konsistent. Ein guter Standard sind 4 Spaces Einrückung pro Ebene ohne Verwendung von Tabulatoren.
- Halten Sie sich an die Sun Java-Code-Convention (http://www.oracle.com/technetwork/java/codeconventions-150003.pdf)
- Programmieren Sie defensiv und testen Sie Eingabewerte auf deren Gültigkeit. Ihr Programm darf
- auch mit Daten nicht abstürzen, die außerhalb der Aufgabenstellung liegen.
- Machen Sie keine Konsoleneingaben oder -Ausgaben, es sei denn die Aufgabe fordert dies explizit.
- Halten Sie Daten und Methoden zusammen. Trennen Sie diese nicht unnötig auf.
- Kopieren Sie keinen Code sondern versuchen Sie mit den bekannten Mitteln der Objektorientierung Code-Duplikate zu vermeiden.