Zug

Implementieren Sie die Klassen **`Wagon`** und **`Train`**.

Für ein **`Wagon`**-Objekt werden die Masse (in Tonnen; ganze Zahl >=5 und <=100) und die Einrichtung (ein Wert aus der vordefinierten Enumeration **`Feature`**: **`Feature::Bar`**, **`Feature::Restaurant`**, **`Feature::Couchette`**, **`Feature::Standard`**, **`Feature::Toilet`**) erfasst. Folgende Methoden und Operatoren sind für die Klasse **`Wagon`** zu implementieren:

- Konstruktor(en) mit einem und zwei Parametern. Masse und Einrichtung in dieser Reihenfolge. Die Einrichtung ist optional mit Defaultwert **`Feature::Standard.`** Entspricht einer der Parameter nicht den Vorgaben (z.B. Masse nicht im erlaubten Bereich) so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`int get_mass() const`**: Retourniert die Masse des Waggons.
- **`bool has_feature(Feature) const`**: Liefert **`true`**, wenn der Waggon die im Parameterwert spezifizierte Einrichtung hat, **`false`** sonst.
- **`operator<<`**: Die Ausgabe eines Objekts des Typs **`Wagon`** muss in der Form [mass: *Masse* tons *Feature*-wagon] erfolgen, z.B.: [mass: 27 tons bar-wagon]. Der vordefinierte Vektor **`feature_names`** kann für die Ausgabe der Enumerationswerte verwendet werden.

Ein **`Train`**-Objekt hat eine Lok-Masse (in Tonnen; ganze Zahl >=50 und <=200), eine Maximallast (in Tonnen; ganze Zahl >=200 und <=10.000) sowie eine Liste von Waggons. Folgende Methoden und Operatoren sind für die Klasse **`Train`** zu implementieren:

- Konstruktor(en) mit zwei und drei Parametern. Lok-Masse, Maximallast und Waggonliste in dieser Reihenfolge. Die Waggonliste ist optional mit dem Defaultwert einer leeren Liste. Entspricht einer der Parameter nicht den Vorgaben (z.B. Lok-Masse oder Maximallast nicht im erlauben Bereich), so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`int total_load() const`**: Retourniert die Gesamtlast des Zuges. Das ist die Summe der Massen aller Waggons plus die Lok-Masse.
- **`bool ready() const`**: Retourniert **`true`**, falls der Zug fahrbereit ist, das heißt, die Gesamtlast des Zuges höchstens so groß (<=) ist wie die Maximallast, zumindest je ein Waggon mit jeder der möglichen Einrichtungen vorhanden ist und niemals mehr als drei Waggons hintereinander hängen, die nicht die Einrichtung **`Feature::Toilet`** anbieten.
- **`void couple(vector<Wagon>):`** Die im Parameter übergebene Liste von Waggons soll an den Zug angekoppelt (an das Ende der Waggonliste des Zugs angefügt) werden. Die Reihenfolge der Waggons in den Teillisten (ursprüngliche Waggonliste des Zuges und anzukoppelnde Liste) soll dabei unverändert bleiben.
- **`vector<Wagon> uncouple(size_t from):`** Ist **`from`** eine gültige Waggonnummer (wenn die Waggons von 0 beginnend in der Reihenfolge, in der sie in der Waggonliste des Zuges auftreten, durchnummeriert werden), so ist der Waggon mit dieser Nummer zusammen mit allen nachfolgenden Waggons vom Zug abzukoppeln (aus der Waggonliste des Zuges zu entfernen). Andernfalls ist eine Exception vom Typ **`runtime_error`** zu werfen. Die Liste der abgekoppelten Waggons ist (in der Reihenfolge, in der sie sich ursprünglich im Zug befunden haben) zu retournieren.
- **`operator<<:`** Die Ausgabe eines Objekts des Typs **`Train`** soll in der Form [*Gesamtlast*/*Maximallast* tons, *fahrbereit* {*Liste der Waggons*}] erfolgen. Dabei ist für *fahrbereit* der String "ready" bzw. "not ready" einzusetzen, je nachdem, ob der Zug fahrbereit ist oder nicht, z.B.:
  [240/210 tons, not ready {[mass: 27 tons bar-wagon], [mass: 15 tons toilet-wagon]}].
- Zusatz für 10 Punkte: **`void sort()`**: Die Waggons sind so umzusortieren, dass Sie nach Einrichtung (in der Reihenfolge, wie die Werte im enum definiert sind) sortiert sind. Das heißt, es sollen zuerst die Waggons mit **`Feature::Bar`**, dann die mit **`Einrichtung::Restaurant`** und so weiter kommen. Die relative Reihenfolge der Waggons in den einzelnen Einrichtungskategorien ist beizubehalten.
- Zusatz für 15 Punkte: **`void switches(size_t from, size_t count, Train& to, size_t pos)`**: Der Waggon an Indexposition **`from`** und seine **`count-1`** direkten Nachfolger sind aus dem Zug **`this`** zu entfernen. Die entfernten Waggons werden im Zug **`to`** so eingefügt, dass der erste eingefügte Waggon die Indexposition **`pos`** erhält. Die relativen Reihenfolgen der Waggons in den einzelnen Teillisten ist beizubehalten. Ist das Umkoppeln wegen illegaler Werte für **`von`**, **`anzahl`** oder **`pos`** nicht möglich, so ist eine Exception vom Typ **`runtime_error`** zu werfen.

Implementieren Sie die Klassen **`Wagon`** und **`Train`** mit den notwendigen Konstruktoren, Methoden und Operatoren, sodass jedenfalls das Rahmenprogramm kompiliert und ausgeführt werden kann und die gewünschten Ergebnisse liefert. Achten Sie in Ihren Konstruktoren darauf, dass nur gültige Objekte erstellt werden können. Werfen Sie gegebenenfalls eine Exception vom Typ **`runtime_error`**.

Für Ihr Programm dürfen Sie **nur** die im vorgegebenen Rahmenprogramm angeführten include-Dateien verwenden!
Instanzvariablen sind **`private`** zu definieren und die Verwendung globaler Variablen ist (abgesehen von im Rahmenprogramm eventuell bereits definierten) nicht erlaubt! Die Datenkapselung darf nicht durchbrochen werden. Es ist daher unter anderem nicht erlaubt, Referenzen oder Pointer auf private Instanzvariablen einer Klasse nach außen zu vermitteln, **`friend`**-Deklarationen (mit Ausnahme bei Operatorfunktionen) zu verwenden, oder setter-Methoden zu implementieren, die die Integrität der Daten nicht gewährleisten. Interpretationsspielraum in der Angabe können Sie zu Ihren Gunsten nutzen.
Die Teilaufgaben, bei denen keine Punkteanzahl angegeben ist, gelten als Basisfunktionalität. Für eine positive Beurteilung ist zumindest die Basisfunktionalität zu implementieren. Diese wird mit 30 Punkten bewertet.
Die übrigen Teilaufgaben müssen nicht unbedingt implementiert werden, führen aber im Falle einer korrekten Implementierung zu einer entsprechenden Erhöhung der Punkteanzahl.