PR1 VU Programmierung 1	Abschlussklausur	25.06.2019

Handyapps

Implementieren Sie die Klassen `App` und `Cellular`:

Ein `App`-Objekt hat die Instanzvariablen `id` (`string`, nicht leer), `memory` (`int`, positiv in MB) und `category` (ein Wert aus der vordefinierten Enumeration `Category` `Category::Social`, `Category::Gadget`, `Category::Tool` und `Category::Game`). Für die Klasse `App` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor(en) mit 2 oder 3 Parametern: Id, Memorybedarf und Kategorie in dieser Reihenfolge. Kategorie ist optional mit dem Defaultwert `Category::Game`. Sollte einer der Parameter nicht die Voraussetzungen erfüllen (z. B. Id ist leer oder Memorybedarf kleiner gleich 0), ist eine Exception vom Typ `runtime_error` zu werfen.
- `string get_id() const`: Retourniert den Wert der Instanzvariable `id`.
- `int get_memory() const`: Retourniert den Wert der Instanzvariable `memory`.
- `bool in_category(Category c) const`: Retourniert `true`, wenn das `this`-Objekt der Kategorie `c` angehört, `false` sonst.
- `operator<<`: Eine App muss in der Form [*id*: *category*, *memory* MB] ausgegeben werden, z. B. **[Angry Birds: Game, 12 MB]**. Der vordefinierte Vektor `category_names` kann für die Ausgabe der Enumerationswerte verwendet werden.

Ein `Cellular`-Objekt hat die Instanzvariablen `memory` (`int`, positiv in GB), `installed` (Liste der installierten Apps) und `running` (`vector<string>`, Liste der Ids der aktuell aktiven (laufenden) Apps). Für die Klasse `Cellular` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor mit einem Parameter Memory. Die Listen der installierten und laufenden Apps sind für ein neues `Cellular`-Objekt leer zu setzen. Sollte der Konstruktorparameter die Voraussetzungen nicht erfüllen (also kleiner oder gleich 0 sein), ist eine Exception vom Typ `runtime_error` zu werfen.
- `int used_memory() const`: Retourniert den Speicherplatz (in MB), der in Summe von allen installierten Apps belegt ist.
- `bool install(const App& app)`: Retourniert `false`, falls die App schon installiert ist (App-Id ist eindeutig) oder der am Handy verfügbare Speicher nicht mehr ausreicht (ein GB sind 1024 MB). Andernfalls wird die App am Ende der Liste der installierten Apps hinzugefügt und `true` retourniert.
- `bool run(const App& app)`: Retourniert `false`, falls die App nicht installiert ist oder bereits läuft. Andernfalls wird die Id der App in die Liste der IDs der laufenden Apps eingefügt und `true` retourniert.
- `operator<<`: Die Ausgabe eines Objekts vom Typ `Cellular` muss in der Form [{*Liste der Apps*} *belegt*/*insgesamt* GB] erfolgen, wobei aktuell laufende Apps in der Liste der Apps durch einen nachgestellten Stern ('*') gekennzeichnet werden und *belegt* und *insgesamt* den aktuell belegten und insgesamt verfügbaren (entspricht dem Wert der Instanzvariable `memory` des `this`-Objekts) Speicher in GB angeben (ein GB sind 1024 MB), z. B.:
  [{[WhatsApp: Social, 244 MB]*, [Angry Birds: Game, 12 MB]} 0.25/64 GB].
- Zusatz für 10 Punkte: Erweitern Sie die Klasse `Cellular` um die Methode `vector<App> deinstall(Category c)`:
  Alle Apps, die der Kategorie `c` angehören, sollen deinstalliert (also aus der Liste der installierten Apps entfernt) werden. Die relative Reihenfolge der verbleibenden Listeneinträge ist beizubehalten. Es ist zu beachten, dass Apps, die aktuell laufen, nicht deinstalliert werden können. Zu retournieren ist eine Liste aller Apps, die zwar der Kategorie `c` angehören, aber nicht deinstalliert werden konnten (da sie aktuell laufen) in derselben relativen Reihenfolge, in der sie in der Liste der installierten Apps auftreten.
- Zusatz für 15 Punkte: Erweitern Sie die Klasse `Cellular` um die Methode `int optimize()`:
  Die Einträge in der Liste der installierten Apps sind so umzuordnen, dass sie nach Kategorie gruppiert sind. Dabei sind die Gruppen in der Reihenfolge der Definition der zugehörigen Enumerationswerte zu bilden. Also zuerst alle Apps der Kategorie `Category::Social`, dann alle der Kategorie `Category::Gadget` und so weiter. Innerhalb der Kategorien ist die relative Reihenfolge der ursprünglichen Liste beizubehalten. Die Methode retourniert die Anzahl der Kategorien, für die jeweils keine App installiert ist.

Implementieren Sie die Klassen `App` und `Cellular` 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.