Aufgabe 1: "ProcessController"

Gesucht ist die Realisierung eines **ProcessControllers**, welche es erlaubt die Abarbeitung von **Prozessen** über 2 verschiedene ProcessQueues zu steuern. Realisieren Sie dazu die folgenden Klassen:

Aufgabe 1a: "Process" [8 Punkte]

Jeder **Process** (Klasse `Process`) hat eine eindeutige **Prozess-Nummer**, eine **Prozess-Bezeichnung**, und einen **Prozess-Code** die dieser beim Anlegen erhält.

Die **Prozess-Nummer** (`int`) soll dabei über alle Prozesse eindeutig sein und beim Erstellen eines Processes automatisch (aufsteigend) vergeben werden, analog zum Beispiel `Account`. Diese Prozess-Nummer soll nach dem Erstellen nicht abänderbar sein.

Die **Prozess-Bezeichnung** (`String`) muss für jeden Prozess gegeben sein. Ist diese beim Erzeugen des Prozesses nicht angegeben, denn soll als Prozess-Bezeichnung "`Prozess #<Prozess-Nummer>`" verwendet werden - also bspw. "`Prozess #123`" für den Prozess mit der Prozess-Nummer `123`, bei dem kein Name beim Erzeugen mitgegeben wurde. Die Prozess-Bezeichnung soll durch eine Methode **`rename()`** siehe unten abänderbar sein.

Der **Prozess-Code** (`String []`) besteht aus einer Reihe von Programm-Zeilen (dh. ein `String[]`). Der Prozess-Code soll durch die Methoden, **`insertLine()`** und **`deleteLine()`** verändert werden können. Ist beim Erzeugen des Prozesses kein Prozess-Code gegeben, so hat der Prozess 0 Lines of Code. Die Nummerierung der Programm-Zeilen des Process-Code beginnt mit 0. Ev. beim Initialisieren nicht gegebene einzelne Code-Zeilen sollen durch die Code-Zeile `"NOP"` (= no operation) ersetzt werden.

`processNo()`	Gibt die Prozess-Nummer des Processes zurück.
`rename(String name)`	Ändert, falls ein gültiger neuer Name des Prozesses gegeben ist, den Namen des Prozesses. Ansonsten verbleibt der Name unverändert. Der Rückgabewert soll anzeigen, ob der Name des Prozesses geändert wurde.
`insertLine(int lineNo, String codeLine)`	Fügt eine Programm-Zeile (codeLine) in den Prozess-Code vor der angegebenen Zeilenangabe (`lineNo`) ein. Jede Zeilenangabe, größer als die Länge des Programm-Codes führt dazu, dass die Programm-Zeile als letzte Zeile zum Programm-Code hinzugefügt wird. Jede Zeilenangabe kleiner als 0 führt dazu, dass die Programm-Zeile vor der bisherigen ersten Zeile in den Programm-Code eingefügt wird. Ist keine gültige Code-Zeile gegeben, soll stattdessen die Code-Zeile "NOP" eingefügt werden.
`delLine(int lineNo)`	Löscht eine Programm-Zeile (`lineNo`) aus dem Programm-Code. Ist die Programm-Zeile ungültig, bleibt der Programm-Code unverändert. Der Rückgabewert soll anzeigen, ob die Zeile erfolgreich gelöscht werden konnte oder nicht.
`duration()`	Gibt die Dauer des Prozesses zurück. Die Dauer berechnet sich dabei auf Grund der Anzahl der Zeilen des Programm-Codes. Dh. ein Programm-Code mit 5 Zeilen liefert eine Dauer von 5.
`toString()`	Gibt eine textuelle Repräsentation des Prozesses zurück. Diese soll die eindeutige Prozess-Nummer, die Prozess-Bezeichnung, den Prozess-Code und die Dauer des Prozesses in geeigneter Form beinhalten.

Aufgabe 1b: "ProcessQueue" [8 Punkte]

Eine **ProcessQueue** (Klasse **`ProcessQueue`**) verwaltet eine **Reihe von Prozessen** (Klasse **`Process`**) nach dem First-in-last-out-Prinzip (vgl. Beispiel **`Stack`**).

Die Klasse **ProcessQueue** soll folgende Methoden sowie Konstruktoren zur Verfügung stellen:

`ProcessQueue(String name, int size)`	Initialisiert eine ProcessQueue. Eine ProcessQueue hat einen Namen und soll eine maximale Anzahl (`size`) von Prozessen aufnehmen können.Ist der Name nicht gültig, soll der Name auf "`unnamed`" gesetzt werden. Ist `size` ungültig soll eine ProcessQueue mit einer (selbst-gewählte) Default-Länge angelegt werden.
`nrProcesses()`	Gibt die Anzahl der in der ProcessQueue momentan enthaltenen Prozesse zurück.
`getProcess(int processNo)`	Gibt den Prozess mit der entsprechenden Prozess-Nummer (`processNo`) zurück. Falls sich dieser nicht in der ProcessQueue befindet soll `null` zurück gegeben werden.
`schedule(Process p)`	Fügt p in die ProcessQueue ein, falls noch Platz vorhanden ist und gibt `true` zurück. Ansonsten wird `false` zurückgegeben. Prozesse können mehrfach in der ProcessQueue aufscheinen.
`processNext()`	Gibt den nächsten (dh. den zuletzt eingefügten) Prozess aus der ProcessQueue zurück und entfernt diesen aus der ProcessQueue. Wenn sich kein Process in der ProcessQueue befindet, wird `null` zurück gegeben.
`duration()`	Gibt die Gesamtdauer aller in der ProcessQueue gespeicherten Prozesse zurück.
`toString()`	Gibt eine textuelle Repräsentation der ProzesseQueue zurück. Diese soll den Namen der ProcessQueue, die Anzahl von Processen, die Dauer aller Processes in der ProcessQueue und eine Auflistung jedes einzelnen Processes in der ProcessQueue beinhalten.

Aufgabe 1c: "ProcessController" [8 Punkte]

Eine **ProcessController** (Klasse **`ProcessController`**) verwaltet **zwei ProcessQueues**. Eine ProzessQueue beinhaltet **"hochpriorisierte" Prozesse** und eine zweite ProcessQueue, welche **"niedrigpriorisierte" Prozesse** beinhaltet.

Setzten Sie die Länge der ProcessQueue für die der "hochpriorisierte" Prozesse auf eine selbstdefinierte fixe Größe. Setzten Sie die Länge der ProcessQueue für die "niedrigpriorisierte" Prozesse auf die doppelte Länge der "hochpriorisierte" Prozesse.

Jeder **ProcessController** soll folgende Methoden zur Verfügung stellen:

`nrProcesses()`	Gibt die Anzahl der momentan enthaltenen Prozesse aller ProcessQueues zurück.
`contains(int processNo)`	Gibt `true` zurück, wenn sich in einer der ProcessQueues ein Process befindet, der die Prozess-Nummer `processNo` hat.
`schedule(Process p, boolean high)`	Fügt p in die "hochpriorisierte" ProcessQueue ein, falls `high` den Wert `true` hat, ansonsten in die "niedrigpriorisierte" ProcessQueue. Die Methode gibt `true` zurück falls der Process eingefügt werden konnte. Ansonsten wird `false` zurückgegeben.
`insertLow(Process p)`	Fügt p in die "niedrigpriorisierte" ProcessQueue ein. Die Methode gibt `true` zurück falls der Process eingefügt werden konnte. Ansonsten wird `false` zurückgegeben.

`processNext()`	Gibt den nächsten Prozess zurück und **entfernt diesen** aus der entsprechenden ProcessQueue. Dabei sollen Prozesse entsprechend der Priorität berücksichtigt werden - daher zuerst alle "hochpriorisierten" Prozesse abgearbeitet und danach die "niedrigpriorisierten" Prozesse. Wenn sich kein Process in keiner der ProcessQueues befindet, wird `null` zurück gegeben.
`insertLine(int processNo, int lineNo, String codeLine)`	Fügt für den Process mit der angegebenen Prozess-Nummer (`processNo`) eine Programm-Zeile (`codeLine`) in den Prozess-Code vor der angegebenen Programm-Zeile (`lineNo`) ein. Jede Zeilenangabe, größer als die Länge des Programm-Codes führt dazu, dass die Programm-Zeile als letzte Zeile zum Programm-Code hinzugefügt wird. Jede Zeilenangabe kleiner als 0 führt dazu, dass die Programm-Zeile vor der bisherigen ersten Zeile in den Programm-Code eingefügt wird. Ist keine gültige Code-Zeile gegeben, soll stattdessen die Code-Zeile "NOP" eingefügt werden.
`delLine(int processNo, int lineNo)`	Löscht eine Programm-Zeile (`lineNo`) aus dem Programm-Code. Ist die Programm-Zeile ungültig, bleibt der Programm-Code unverändert. Der Rückgabewert soll anzeigen, ob die Zeile erfolgreich gelöscht werden konnte oder nicht.
`duration()`	Gibt die Gesamtdauer aller in den ProcessQueues gespeicherten Prozesse zurück.
`toString()`	Gibt eine textuelle Repräsentation des ProcessController zurück. Diese soll die Anzahl und Dauer aller Processes des ProcessControllers und eine Auflistung der "hochprioritären" und "niedrigproritären" Prozesse beinhalten.

Im MOODLE finden Sie `ProcessControllerDemo.java`, das zeigt, wie Ihre Klassen verwendet werden können sollen.

Darüber können Sie zur Vereinfachung den jeweiligen `toString()`-Code im MOODLE herunterladen und diesen in Ihrer Realisierung direkt übernehmen.

Abgabe:

- ➔ **`ProcessQueueController.java` mit der Klasse `ProcessQueueController`.**
- ➔ **`ProcessQueue.java` mit der Klasse `ProcessQueue`.**
- ➔ **`Process.java` mit der Klasse `Process`.**

Aufgabe 2: "Map" [12 Punkte]

Gesucht ist eine Klasse **`Map`**, die es ermöglicht Schlüssel/Wert-Paar zu speichern. Als Schlüssel, sowie für die Werte sollen `Strings` vorgesehen werden. Unter einem eindeutigen Schlüssel soll der dazu gehörende Wert gespeichert werden. Schlüssel müssen in der Map eindeutig sein, Werte können mehrfach vorkommen.

Realisieren Sie die **`Map`** auf Basis von Arrays. Es soll dennoch möglich sein theoretisch beliebig viele Schlüssel/Wert-Paare zu speichern.

Folgende Funktionalität soll Map unterstützen:

- **`add(key, value)`: Hinzufügen** eines Schlüssel/Wert-Paares zur Map **bzw. Ändern** des Wert zu einem bereits gespeicherten Schlüssel. Zeigen Sie mit dem Rückgabewert an, ob der Wert erfolgreich eingefügt werden konnte.
- **`del(key)`**: Löschen eines Schlüssel/Wert-Paares aus dem Map. Das als Parameter übergebene Schlüssel soll gesucht und die Schlüssel/Wert-Zuordnung aus der Liste entfernt werden. Zeigen Sie mit dem Rückgabewert an, ob der Schlüssel gefunden und somit aus der Map entfernt werden konnte.
- **`get(key)`**: Gibt den Wert, welcher zu dem Wert hinterlegt ist zurück; ansonsten `null`.
- **`size()`**: Gibt die Anzahl der Schlüssel/Wert-Paare in der Map zurückgibt.
- **`keys()`**: Gibt die Schlüssel der Map als neues Array zurückgibt. Die Reihenfolge der Schlüssel in der Liste soll dabei egal sein.

Ihre `Map` sollen wie folgt verwendet werden können:

```
class MapDemo{
  public static void main(String [] args) {

    Out.print("\nConstructors----------------------------------");
    Map map = new Map();
    Out.print("\nmap: " + map.toString());          // empty Map

    map.add("1", "A");                              // 1,A is no contained
    Out.print("\nmap: " + map.toString());          // (1,A)

    map.add("2", "B");                              // 2,B is no contained
    Out.print("\nmap: " + map.toString());          // (1,A)(2,B)

    map.add("2", "X");                              // 2 is changes to X
    Out.print("\nmap: " + map.toString());          // (1,A)(2,X)

    map.del("1");
    Out.print("\nmap: " + map.toString());          // (2,X)

    map.add("3", "C");                              // 3,C is no contained
    Out.print("\nmap: " + map.toString());          // (2,X)(3,C)

    Out.print("\nmap size: " + map.size());         // 2


    Out.print("\nmap get: " + map.get("2"));   // X
    Out.print("\nmap get: " + map.get("3"));   // C
    Out.print("\nmap get: " + map.get("1"));   // null
    Out.print("\nmap get: " + map.get(null));  // null
    String [] keys = map.keys();
    Out.print("\nmap keys: ");
    for (int i = 0; i < keys.length; i++) {
     Out.print("\n   " + keys[i]);                 // 2 3
    }


  }
}
```

Sie können sich das `TimerDemo` über **MOODLE** herunterladen.

Abgabe:

➔ **`Map.java` mit der Klasse `Map`.**