Aufgabe 1: "WaitingQueue" [12 Punkte]

Schreiben Sie eine Klasse, welche einen Wartenummern-Automat (`WaitingQueue`) realisiert, um zum Beispiel das Anstellen der Kunden vor der Feinkost-Theke zu organisieren.

Man **"zieht"** sich eine Wartenummer beim Automaten und ist dann drann, wenn die eigene Wartenummer aufgerufen wird. Die Wartennummern, sollen dabei immer weiter hochgezählt werden und somit vergrößert sich die Anzahl der Wartenden.

Diese Warteschlange wird **"abgearbeitet"**, in dem die aktuell servisierte Nummer in der Warteschlange drann kommt - die Anzahl der Wartenden verringert sich dadurch . Dazu merkt sich der Wartenummern-Automat, die Nummer, die gerade drann ist.

Die Wartenummern sollen dabei aufsteigend, bis zu einem voreinstellbaren höchsten Wartenummer (bspw. 99), vergeben weren. Nach Vergabe der höchsten Wartenummer soll wieder bei 1 weiter gemacht werden und aufsteigend Wartenummern vergeben werden. Also im Fall von höchster Wartenummer 99, soll beispielsweise 97, dann 98, dann 99 und danach 1 als Wartenummern vergeben werden.

Die Anzahl der Wartenden ist natürlich dabei durch die maximal möglichen unterschiedlichen Anzahl von Wartenummern begrenzt. Bei einer höchsten Wartenummer von 99 sind das 99 Wartende.

Dieser Wartenummern-Automat soll Folgendes zur Verfügung stellen.

- Jeder Wartenummern-Automat hat eine **Bezeichnung** (e.g. "Feinkost", "Fleisch", "Käsetheke").
- Jeder Wartenummern-Automat kennt jene **Wartenummer**, die aktuelle gerade an der Reihe ist.
- Jeder Wartenummern-Automat hat eine voreinstellbare **höchste Wartenummer**.
- Beim **Erzeugen** einer WaitingQueue soll jedenfalls deren *Bezeichnung* mitgegeben werden. Alternativ soll es auch möglich sein, die *höchste Wartenummer* beim Erzeugen mitzugeben. Ist die höchste Wartenummer nicht gegeben, oder ungültig, soll ein Standardwert stattdessen gesetzt werden.

Dieser Wartenummern-Automat soll folgende Funktione zur Verfügung stellen.

- **`getWaitingNumber()`**: gibt neue (nächst höhere) Wartenummern aus. Ist die höchste Wartenummer erreicht soll wieder bei `1` fortgefahren werden. Ist die maximale Anzahl von Wartenden erreicht, soll `-1` zurück gegeben werden.
- **`call()`**: gibt die Wartenummer zurück, die als nächstes an der Reihe ist und dies darüberhinaus auf der Konsole ausgibt. Gibt es keine Wartenden, dann gibt die Methode `-1` zurück.
- **`service():`** geht zur nächsten Wartenummer weiter vor falls es Wartende gibt. Gibt es keine Wartenden, dann gibt die Methode `false` zurück, ansonsten `true`.
- **`isTurn(int n)`**: gibt zurück, ob die Wartenummer n gerade an der Reihe ist oder nicht.
- **`queueSize()`**: gibt die Anzahl der Wartenden zurück.

Ihre `WaitingQueue` soll wie folgt verwendet werden können:

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

            // Wartenummern-Automat fuer die Feinkost.
            // Hoechste zu vergebende Nummer: 3, dh. max. 3 Wartende moeglich
            WaitingQueue feinkost = new WaitingQueue("Feinkost", 3);

            Out.print("\nYour waiting number: " + feinkost.getWaitingNumber());
                  // Wartenummer 1, 1 Wartender
            Out.print("\nYour waiting number: " + feinkost.getWaitingNumber());
                  // Wartenummer 2, 2 Wartende
            Out.print("\nSize of queue: " + feinkost.queueSize());
                  // 2
            Out.print("\nYour waiting number: " + feinkost.getWaitingNumber());
                  // Wartenummern 3, 3 Wartende
            Out.print("\nSize of queue: " + feinkost.queueSize());
                  // 3

            int waitingNo = feinkost.getWaitingNumber();
                  // -1, weil kein Platz mehr in Warteschlange
            if (waitingNo == -1 ) Out.print("\nSorry no more slots at the moment");
                  // true
            Out.print("\nSize of queue: " + feinkost.queueSize());
                  // 3

            int nextToBeServiced = feinkost.call();
                  // liefert 1
            if (feinkost.isTurn(2)) Out.print("\nNot your turn: 2");
                  // liefert false, Wartende 3

            if (feinkost.isTurn(nextToBeServiced)) {
                  // liefert true
                  Out.print("\nNow servicing: " + nextToBeServiced);
                  feinkost.service();
                  // Wartende 2
            }
            Out.print("\nSize of queue: " + feinkost.queueSize());
                  // 2
            waitingNo = feinkost.getWaitingNumber();
                  // 1, weil wieder Platz in Warteschlange; auf 3 folgt 1
            Out.print("\nYour waiting number: " + waitingNo);
                  // 1

            // Wartenummern-Automat fuer die Feinkost.
            // Hoechste zu vergebende Nummer: 10, dh. max. 10 Wartende moeglich
            WaitingQueue kaese = new WaitingQueue("Kaese-Theke", 10);
            if (kaese.service()==-1) Out.print("\nNo one to service");
            // true
      } // end main()
} // end class WaitingQueueDemo
```

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

Abgabe:

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

Aufgabe 2: "Timer" [12 Punkte]

Entwickeln Sie eine Klasse **`Timer`**, die es ermöglicht Zeiten zu repräsentieren. Dabei soll intern die Zeit in Sekunden (**`long`**) gespeichert werden.

Sehen Sie zwei Konstruktoren vor. Einen, um einen `Timer` zu erzeugen, dessen Ausgangszeit auf 00:00:00 gestellt ist. Ein weiterer Konstruktor soll ein `Timer`-Objekt mit entsprechend voreingestellter Zeit erzeugen können und dazu die Zeit über drei separate Parameter (Stunden, Minuten und Sekunden) erhalten. Ist die Parametrisierung beim Erstellen ungültig soll ebenfalls ein "auf den Zeitpunkt 00:00:00"-gestelltes `Timer`-Objekt erzeugt werden.

Stellen Sie folgende Funktionalitäten zur Verfügung:

- **`inc()`**-Methoden, die es ermöglicht die Zeit um:
  - **a)** eine Anzahl von Sekunden hinauf zu zählen,
  - **b)** eine Anzahl von Minuten und Sekunden hinauf zu zählen,
  - **c)** eine gewisse Anzahl von Stunden, Minuten und Sekunden hinauf zu zählen und
  - **d)** den Wert, der durch ein `Timer`-Objekt gegeben ist hinauf zu zählen.

  Zeigen Sie mit Hilfe eines Rückgabewertes an, ob die Parametrisierung bzgl. der Stunden, Minuten, und Sekunden bzw. des `Timer`-Objekts korrekt war.
- **`diff(Timer t)`**-Methode, die es ermöglicht die Zeitdifferenz (Betrag) zwischen der Zeit des `Timers` und der Zeit des `Timers` ***t*** zu berechnen. Es soll ein neues `Timer`-Objekt zurückgeben, das die Differenz (Betrag) der beiden Zeiten ausdrückt.
- **`reset()`-**Methode, die es ermöglicht, die Zeit des `Timers` wieder auf seinen ursprünglichen Ausgangszeitpunkt bei der Initialisierung des Objekts zurück zu setzen.
- **`compareTo(Timer t)`**, die es ermöglicht, die Zeit des `Timers` gegenüber dem übergebenen `Timer`-Objekt ***t*** zu vergleichen. Die Methode liefert einen positiven `int`-Wert, falls die Zeit des `Timers` grösser ist als die Zeit des `Timers` ***t***, einen negativen Wert, wenn die Zeit des `Timers` kleiner ist als die Zeit des `Timers` ***t*** oder `0` falls beide Zeiten gleich sind. Ist ***t*** nicht gegeben wird ein positiver Wert zurück gegeben.
- **`asText()`-**Methode, die es ermöglicht, die Zeit des `Timers` als `String` im Format "`[h:m:s:]`" zurück zu geben.

Ihre `Timer` sollen wie folgt verwendet werden können:

```
class TimerDemo {

      public static void main(String[] args) {

            Out.print("\nCONSTRUCTORS ----------------------");
            Timer t1 = new Timer();
            Out.print("\nt1: " + t1.asText());

            Timer t2 = new Timer(24,0,0);
            Out.print("\nt2: " + t2.asText());

            Timer t3 = new Timer(-1,0,0);
            Out.print("\nt3: " + t3.asText());

            Timer t4 = new Timer(2,15,9);
            Out.print("\nt4: " + t4.asText());

            Out.print("\nDECREMENT sec ----------------------");
            boolean b = t4.decrement(15*60+10);
            Out.print("\nt4: " + t4.asText()+ " Rueckgabewert:" +b);

            Out.print("\nDECREMENT min ----------------------");
            t4.reset();
            Out.print("\nt4: " + t4.asText());
            b = t4.decrement(16,10);
            Out.print("\nt4: " + t4.asText()+ " Rueckgabewert:" +b);

            Timer t5 = new Timer(1,0,0);
            Out.print("\nt5: " + t5.asText());
            b = t5.decrement(60*60);
            Out.print("\nt5: " + t5.asText()+ " Rueckgabewert:" +b);


            Out.print("\nDECREMENT h ----------------------");
            t4.reset();
            Out.print("\nt4: " + t4.asText());
            b = t4.decrement(1,16,10);
            Out.print("\nt4: " + t4.asText()+ " Rueckgabewert:" +b);

            t4.reset();
            Out.print("\nt4: " + t4.asText());
            b = t4.decrement(1*60*60+16*60+10);
            Out.print("\nt4: " + t4.asText()+ " Rueckgabewert:" +b);

            Out.print("\nDIFF timer ----------------------");
            t4.reset();
            Out.print("\nt4: " + t4.asText());
            Timer t6= t2.diff(t4);
            Out.print("\nt6: " + t6.asText());


            Out.print("\nDIFF static ----------------------");
            t4.reset();
            Out.print("\nt4: " + t4.asText());
            long time = Timer.diff(t2, t4);
            Out.print("\ntime: " + time);

      } // end main()
} // end class TimerTester
```

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

Abgabe:

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