Projektbeschreibung:

Zum Marathonlauf melden sich die Teilnehmer mit Namen an und erhalten eine (fortlaufende) Startnummer. Der Veranstalter organisiert alle Starter in einer Startliste, in der sie nach Startnummern geordnet eingetragen sind. Kommt ein Läufer ins Ziel, wird er mit der erreichten Laufzeit in einer zweiten Rangliste hinzugefügt und aus der Startliste entnommen.

Aufgaben:

a) Entwerfen Sie ein Implementationsdiagramm zur Modellierung der Lauf-Organisation mit den Klassen Lauf, Starter und List und einer Konsolenklasse ConMarathon. Überlegen Sie für den gesamten Ablauf des Marathonlaufs gut, welche Attribute und Methoden in den Klassen einzusetzen sind. Falls Sie kein befriedigendes Modell entwickeln können, schauen Sie sich in den Hilfen eine mögliche Modellierungs-Variante an.

b) Implementieren Sie in Java-Code die Klasse Starter; der Konstruktor erhält den Starter-Namen und die Startnummer. Setzen Sie nur sinnvolle set-Methoden ein. Für die spätere Ausgabe hilfreich sind Methoden, die einen String mit den Daten "Nummer Name Zeit" und die Zeit (z. B. 3.25) in einen String im Format "hh:mm:ss" (z. B. 3:15:00) liefern.

c) Implementieren Sie die Klasse Lauf, die eine startListe und eine rangListe vom Typ List enthält. Fügen Sie dazu die NRW-Abiturklasse List im Projekt hinzu!
Modellieren Sie in Pseudo-Code die Lauf-Methode anmelden, die zu einem übernommenen Namen ein neues Starter-Objekt mit erzeugter, nächster Startnummer an die Startliste anhängt, und implementieren Sie die Methode dann auch in Java-Code.

d) Ergänzen Sie (auch im Implementationsdiagramm) zur Konsolen-Klasse ConMarathon als Attribut ein Lauf-Objekt (z. B. meinLauf) und die Methoden ready (meldet einzeln ca. acht Namen als Starter an; ruft dazu die zuvor erarbeitete Lauf-Methode anmelden auf) und steady (zeigt die Daten Nummer und Name zu allen Starter-Objekte der Startliste an).
Testen Sie ausführlich das Anmelden und die Anzeige der Starterdaten aus der Startliste.

e) Modellieren Sie abschließend die weitere Simulation des Marathonlaufes: Zufällig aus der Startliste entnommene Starter erhalten eine (zufällige, aber realistische) Zieleinlaufzeit (jeweils größer als die vorherige) und werden an die Rangliste angehängt. Abschließend werden die Daten (Platz, Nummer, Name und Zeit) alle Starter-Objekte der Rangliste angezeigt.

Nutzen Sie zur vorbereitenden Modellierung weiter das Entwurfs- oder Implementationsdiagramm, Algorithmus-Formulierungen in Pseudo-Code oder auch als Struktogramm und implementieren Sie erst abschließend in Java-Code.

Mögliche Erweiterungen und Hilfen für die erfolgreiche Projektbearbeitung

… finden Sie auf den folgenden Seiten. Sie sollten erst nach intensiver Bearbeitung gelesen werden, oder falls Sie auch nach intensivem Nachdenken und Nachfragen bei Mitschülern zu keiner zufriedenstellenden Teillösung gelangten.

Detaillierte Beschreibungen möglicher weiterer Methoden:

(Ihre eigene Modellierung sollte immer den Vorzug erhalten; nur bei Bedarf sollten Sie sich von den folgenden Detail-Beschreibungen 'inspirieren' lassen. Die hier angegebenen Methoden können, müssen aber nicht zwingend Teil Ihres Modells sein.)

f) Modellieren Sie als Struktogramm die ConMarathon-Methode *zeigeAlleStarter()*, die von *steady()* aufgerufen wird. Sie nutzt die Lauf-Hilfsmethode *holeNaechstenStarter (boolean pDenErsten)* und erhält damit das gewünschte erste / nächste Starter-Objekt, bildet einen String mit dessen Daten "Nummer Name" und zeigt die Daten an. Implementieren Sie diese Methoden in Java-Code.

g) Die ConMarathon-Methode go soll zufällige, jeweils längere Zeiten zwischen 2h30 (2.5) und 6h erzeugen und zufällig einem der Läufer aus der Starterliste auswählen. Dann soll durch Aufruf der Lauf-Methode *naechsterImZiel()* mit den Parametern Startnummer und Zeit dieser Läufer bearbeitet werden (siehe Folgendes). Entwickeln Sie auch diese Methode.

h) Modellieren und implementieren Sie die Lauf-Methode *naechsterImZiel (int pNummer, double pZeit)*, die wie in der Projektbeschreibung dargestellt arbeitet und die erfolgreiche Ergänzung des Starter-Objektes zurückmeldet.
Beim realen Marathonlauf wäre eine geschützte Hilfsmethode *private boolean findeNummer (int pNummer)* hilfreich, die beim Zieleinlauf den Starter mit seiner Startnummer in der Startliste aufsucht und für die weitere Bearbeitung zum aktuellen Knoten der Liste macht (oder den Wert false zurückgibt, falls die Startnummer nicht in der Startliste ist).
Zufällig gebildete Startnummern mit *findeNaechsteNummer()* in der Startliste zu suchen ist wenig sinnvoll, da nach vielen Entnahmen von Startern, also bei fast leerer Startliste, sehr viele Startnummern dort nicht mehr existieren und vergeblich gesucht würden.
Praktischer für die Simulation ist es also, einen der noch in der Startliste verbliebenen zum Aktuellen der Liste zu machen (also zufällig oft startListe.next() aufzurufen), und diesen Aktuellen dann in *naechsterImZiel()* zu behandeln. Dazu könnte als Parameter pNummer der Wert 0 übergeben werden oder der Parameter entfallen.

i) Die Methode *finish()* sollte das gleiche wie die Methode *steady()* tuen: Es sollen damit die Daten aller Starter (Startnummer, Name, Zeit) in der Reihenfolge des Zieleinlaufs (also wie in der Rangliste eingefügt) abschließend angezeigt werden. Also wäre mit einem zusätzlichen Parameter der erneute Aufruf einer Methode (z. B. *zeigeAlleStarter(boolean pMitZeit)*) denkbar. Ergänzen und testen Sie diese Methode.

Mögliche Erweiterungen:

(1) Die prominenten Läufer bekommen die ersten z. B. 100 (anzPromis) Startnummern. Die ConMarathon-Methode *ready()* soll zufällig einen der acht Starter als PromiStarter anmelden. Ergänzen Sie dazu in der Lauf-Methode *anmelden()* den zusätzlichen Parameter pAlsPromi und die notwendige Fallbehandlung.

(2) Die Läufer starten jeweils in Gruppen von z. B. 100 (blockGroesse) Startern gleichzeitig. Ergänzen Sie die von der ConMarathon-Methode *steady()* aufgerufene Methode *zeigeAlleStarter()* um den boolean-Parameter pInBlocks und die blockweise Ausgabe der Starter. Da derzeit unter acht ein PromiLäufer startet, sollten also zwei Starter-Blocks (Block 1 mit Promis, Block 2 mit den übrigen Startern ab Startnummer 101) zu sehen sein. Testen Sie damit ausführlich die nach den Anmeldungen erzeugte Starterliste.
Tipps: Eine Methode *holeStartBlock (int pBlockNum, boolean pMitZeit)* liefert dazu einen Text zurück mit der Startnummer und dem Namen jedes Starters pro Zeile. Die Anweisung lText += "\n" hängt einen Zeilenumbruch an die Textzeile an, die folgenden Daten des nächsten Starters werden im String ergänzt und später in der Folgezeile angezeigt. Möglich ist auch eine Methode *zeigeStartBlock()*, die alle Starter des Blocks sofort anzeigt.

(3) Statt der acht Demo-Namen sollen nun realistisch viele Teilnehmer mit zufälligen Namen generiert werden, davon z. B. 2% PromiStarter. Nutzen Sie dazu die Klasse Zufallsnamen-Generator mit der Methode *getName (int pFormat, pGeschlecht)* im SLP. Ergänzen Sie die ConMarathon-Methode *ready2()*, die die gewünschte Anzahl Starter (und ca. 2% Promis) in die startListe einfügt.
Tipps: Ob ein Starter Promi wird, kann z. B. durch *Math.random()<0.02* ermittelt werden. Durch Einbindung der Klasse InOut (aus JavaEditor-Programmverzeichnis kopieren) wird so ermöglicht, in der Methode *steady()* bzw. *zeigeAlleStarter()* nach Anzeige eines Blocks auf den Benutzer-Tastendruck zu warten:
System.out.println("Anzeige des nächsten Blocks mit [Enter].");
while (InOut.readln()=="") {} // end of while

Hilfen für die erfolgreiche Projektbearbeitung

(1) Hilfreiche Daten, die als Attribute (Konstanten) gespeichert werden:
(1a) in der Klasse ConMarathon die Simulationsparameter:

```
private static String[] demoNamen = {"Alian","Bert","Cisco","Ernie","Fake","Gui","Honk","Ian"};
private final double minZeit = 2.5;  // Mindestzeit bis zum Zieleinlauf für die Zeitsimulation
private final double maxZeit = 6.0;  // Höchstzeit bis zum Zieleinlauf
```

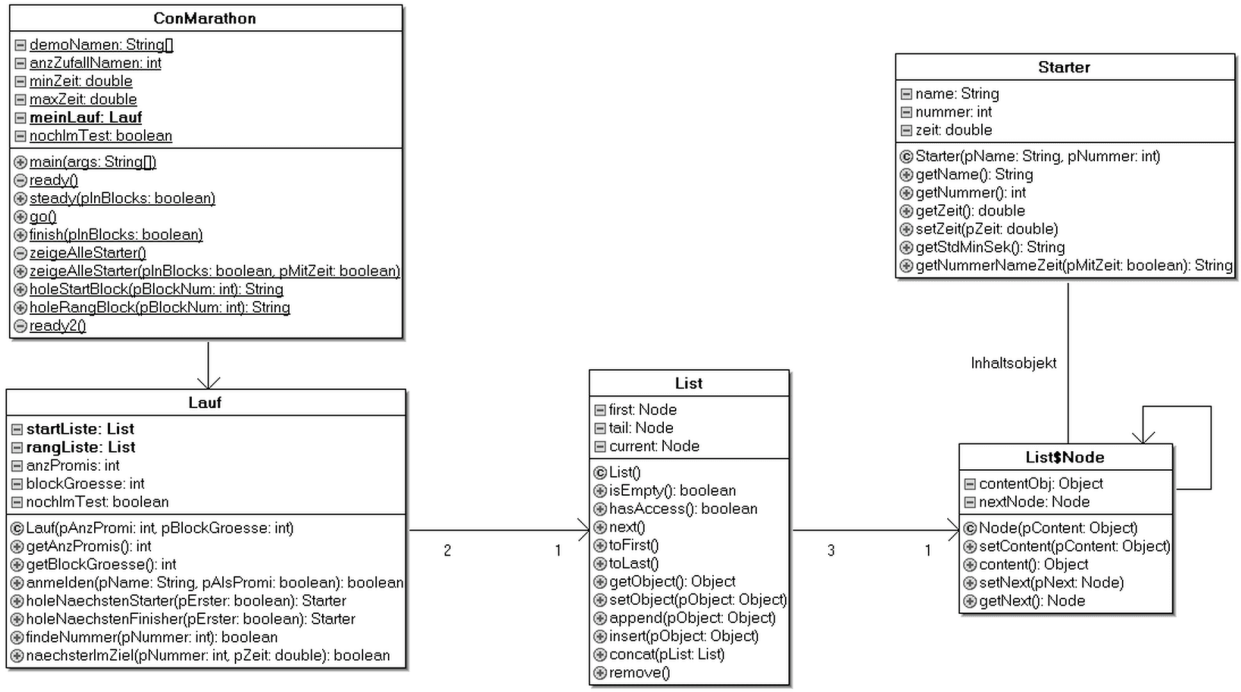
(1b) in der Klasse Lauf die Marathon-spezifischen Daten:

```
private int anzPromis = 20;  // mögliche Anzahl der PromiStarter und höchste Promi-Startnummer
private int blockGroesse = 20;  // Anzahl der Läufer in einem Startblock / einer Zielgruppe
```

Hilfen für die erfolgreiche Projektbearbeitung

(2) Eine mögliche Modellierungs-Variante (andere Varianten sind möglich!):

Im Rahmen des MVC-Konzeptes stellt eine Konsolen-Klasse ConMarathon die Schnittstelle zum Benutzer (als View für die Anzeige) dar. Sie enthält als Attribute ein Lauf-Objekt *meinLauf* und alle Daten, die für die Simulation notwendig sind, z. B. ein Feld mit acht Demonamen, die Mindest- und Höchstzeit für den simulierten Zieleinlauf. Der gesamte Simulationsprozess wird durch z. B. diese Hauptmethoden modularisiert: *ready* (realisiert die Anmeldungen), *steady* (zeigt die Startaufstellung an), *go* (ermittelt Zieleinlaufzeiten) und *finish* (zeigt die Starter nach dem Zieleinlauf komplett einschließlich Zeit an).
Als Controller und Model dient die Klasse Lauf mit ihren Attributen *startListe*, *rangListe* (List-Objekte) sowie später in der Erweiterung *anzPromis* (Anzahl prominente Starter mit kleiner Startnummer) und *blockGroesse* (Anzahl Starter pro Block; vgl. Erweiterungen). Die Klasse Lauf bietet Methoden für die Anmeldung eines Starters (z. B. *anmelden (String pName, boolean pAlsPromi)*), zum Holen eines Starter-Objekts aus der *startListe* oder *rangListe* (z. B. *holeNaechstenStarter (boolean pErster)* und *holeNaechstenFinisher (boolean pErster)*).
Als Model dient die Klasse Starter, deren Objekte Knoteninhalte der beiden Lauf-Listen sind.
Daraus ergibt sich dieses Implementationsdiagramm:



Nutzen Sie diese Modellierungs-Variante nur, falls Ihnen keine eigene Modellierung gelingt. Die Namen aller Methoden und Attribute sind frei wählbar, sollten aber selbsterklärend sein.

Hilfen für die erfolgreiche Projektbearbeitung

(3) Pseudo-Code der Lauf-Methode anmelden (mit Name, als PromiStarter):
(hier ist die Erweiterung mit den PromiStartern bereits integriert: PromiStarter erhalten kleine Startnummern und starten dadurch vorne im Feld – später: im ersten Startblock)

```
falls der Starter als PromiStarter anzumelden ist
dann suche in der Startliste den, dessen Startnummer größer ist als die Anzahl der PromiLäufer
        merke dazu jeweils die Startnummer des Vorgängers
        falls diese Vorgänger-Startnummer kleiner als die PromiAnzahl ist
        dann erzeuge einen neuen Starter mit dem Namen und mit nächster Startnummer
                füge den Starter hinter dem Vorgänger (also vor dem Aktuellen) in der Startliste ein
        sonst gebe Fehler zurück
sonst ermittele die bislang größte vergebene Startnummer in der Startliste
        erzeuge einen neuen Starter mit dem Namen und mit nächster Startnummer
        füge den Starter hinten an die Startliste an
gebe Rückmeldung zum Gelingen dieser Anmeldung
```

(4) Pseudo-Code der Hilfsmethode zeigeAlleStarter (aufgerufen von der Methode steady):

Variante ohne Block-Aufteilung:

```
gehe zum Ersten in der Startliste
solange Starter anzuzeigen sind wiederhole
        zeige die Daten des Starters an
        gehe zum Nächsten in der Startliste
ende wiederhole
```

Variante mit Block-Aufteilung:

```
(ermittele die Anzahl der Starter pro Startblock)
lege anhand übergebener Blocknummer fest:
  größte Startnummer dieses Startblocks
lege anhand bekannter Blockgröße fest:
  kleinste Startnummer dieses Startblocks
gehe zum Ersten in der Startliste
solange Starter vorhanden sind wiederhole
        falls Startnummer zu groß ist
        dann beende wiederholen
        sonst falls Startnummer nicht zu klein ist
                dann merke die Daten des Starters
        gehe zum Nächsten in der Startliste
ende wiederhole
```

Diese Methode *zeigeAlleStarter()* sollte in der Klasse ConMarathon (dem View) implementiert werden, da sie eine Ausgabe auf der Konsole liefert. Die Klasse hat aber keinen direkten Zugriff auf die Startliste der Klasse Lauf. Daher ist eine zusätzliche Lauf-Hilfsmethode *holeNaechstenStarter (boolean pErster)* hilfreich, die das erste oder nächste Starter-Objekt liefert (oder den Wert null).

Hilfen für die erfolgreiche Projektbearbeitung

(5) Java-Code der ConMarathon-Methode zeigeAlleStarter:
(wird auch zur ersten Kontrolle genutzt, ob nach dem Anmelden Starter-Objekte in der Liste sind; in der Projekt-Erweiterung soll die Ausgabe auch blockweise erfolgen: erst StartNr 1-100, dann 101-200, ...)

```
/**
* zeigt auf der Konsole zu allen Startern in der Startliste die Werte Nummer, Name;
* nutzt ConMarathon-Methode holeNaechstenStarter (boolean pDenErsten);
* liefert einen String zurück der Form "101  Ernie"
* oder einen leeren String "", falls kein Listenelement mehr vorhanden ist
*/
public static void zeigeAlleStarter()
{
  System.out.println("\nHilfsausgabe der kompletten Starterliste");
  int i=1;
  Starter lTemp;
  String lText = "";
  do {
    lTemp = meinLauf.holeNaechstenStarter (i==1);  // Parameter pDenErsten
    if (lTemp != null) {
      lText = lTemp.getNummerNameZeit(false);  // Parameter pMitZeit
    } // end of if
    System.out.println (lText);
    i++;
  } while (lText != ""); // end of do-while
}
```

(6) Java-Code der Lauf-Hilfs-Methode holeNaechstenStarter (pErster):

```
/**
 * liefert aus der startListe das erste oder nächste Starter-Objekt
 * @param boolean pErster: falls true: Erster gewünscht, sonst Nächster
 * @return Starter: ein Starter-Objekt aus der startListe
 */
public Starter holeNaechstenStarter (boolean pErster) {
  Starter lRet = null;
  if (pErster)
  {
    this.startListe.toFirst();
  } else {
    this.startListe.next();
  } // end of if-else
  if (this.startListe.hasAccess())
  {
    lRet = (Starter)this.startListe.getObject();
  }
  return lRet;
}
```

Dokumentation der Klasse ConMarathon:

Modifier and Type	Method and Description
`static void`	`finish(boolean pInBlocks)` zeigt auf der Konsole die Daten aller Finisher (Rang, Nummer, Name, Zeit); ruft dazu die Methode zeigeAlleStarter auf;
`static void`	`go()` ermittelt zufällige Ziel-Zeiten zu jedem Starter, trägt diese ein, hängt zielerreichende Starter an rangListe des Laufs an und entnimmt den Starter aus der startListe des Laufs
`static java.lang.String`	`holeRangBlock(int pBlockNum)` liefert als String in Zeilen untereinander "Nummer Name Zeit" aller Finisher eines Blocks; die Blockgrösse ist als Attributwert von Lauf bekannt.
`static java.lang.String`	`holeStartBlock(int pBlockNum)` liefert als String in Zeilen untereinander "Nummer Name" aller Starter eines Blocks; die Blockgrösse ist als Attributwert von Lauf bekannt.
`static void`	`main(java.lang.String[] args)`
`static void`	`ready()` holt zufällig Gewählten aus den acht Demonamen, vergibt Start-nummer und hängt ein weiteres Starter-Objekt an Startliste an.
`static void`	`ready2()` erzeugt mittels ZufallsnamenGenerator Namen zu Starter-Objekten, vergibt Startnummern; berücksichtigt auch ca. 2% PromiStarter (mit Startnummern 1 bis anzPromis); ersetzt Methode ready, in der nur acht Demonamen genutzt werden
`static void`	`steady(boolean pInBlocks)` zeigt alle Starter an (erweitert: unterteilt in Starterblocks)
`static void`	`zeigeAlleStarter()` zeigt auf der Konsole zu allen Startern in der Startliste die Werte Nummer, Name in der Form "101 Ernie"
`static void`	`zeigeAlleStarter(boolean pInBlocks, boolean pMitZeit)` zeigt auf der Konsole zu allen Startern in der Startliste bzw. Finishern in der Rangliste die Werte (Platz), Nummer, Name, (Zeit) in der Form "101 Ernie" oder "#3 105 Bert 3:15:00"

Dokumentation der Klasse Lauf:

Modifier and Type	Method and Description
boolean	`anmelden(java.lang.String pName, boolean pAlsPromi)` erzeugt mit übernommenem Namen ein neues Starter-Objekt und hängt es mit der nächstgrößeren Startnummer an die Startliste an
boolean	`findeNummer(int pNummer)` sucht die übernommene Startnummer in der Startliste und macht Gefundenen zum Aktuellen
int	`getAnzPromis()`
int	`getBlockGroesse()`
Starter	`holeNaechstenFinisher(boolean pErster)` liefert aus der rangListe das erste oder nächste Starter-Objekt
Starter	`holeNaechstenStarter(boolean pErster)` liefert aus der startListe das erste oder nächste Starter-Objekt
boolean	`naechsterImZiel(int pNummer, double pZeit)` holt den aktuellen oder den mit übernommener Startnummer aus Startliste entfernt diesen Starter aus der StartListe trägt die übernommene Zeit ein und fügt den Starter in Rangliste ein

Dokumentation der Klasse Starter:

Modifier and Type	Method and Description
java.lang.String	`getName()`
int	`getNummer()`
java.lang.String	`getNummerNameZeit(boolean pMitZeit)` liefert einen String mit den Daten "Nummer Name" bzw. (falls pMit-Zeit vom Wert true ist) "Nummer Name Zeit"
java.lang.String	`getStdMinSek()` wandelt den Attribut-Wert zeit um von double in einen String hh:mm:ss und liefert diesen zurück
double	`getZeit()`
void	`setZeit(double pZeit)`