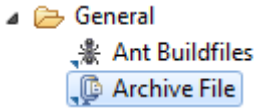
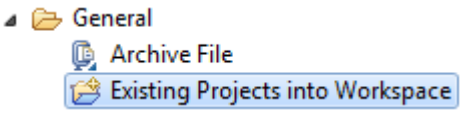
Aufgaben zur Abgabe als Prüfungsleistung für Prog I

- Allgemeine Regeln
  - Abgabetermin: siehe OLAT
    - Falls technische Probleme in OLAT auftreten, dann per E-Mail direkt an mich senden
  - Verwenden Sie keine Umlaute wie ä, ö, ü oder Sonderzeichen wie ß
  - Alle Klassen müssen in einem Paket namens „`pack`“ ausgeliefert werden, dann lässt es sich einheitlich testen: `package pack;`
  - Die Sprache soll für alle Klassen, Bezeichner, Kommentare usw. Englisch sein
  - Zielumgebung: mind. Java 20 mit Eclipse mind. 2023-06
  - Falls zusätzliche Dokumente gefordert sind, dann bitte als `pdf` abgeben
  - Falls Sie Hinweise oder Kommentare außerhalb des Quelltexts ergänzen möchten, dann muss diese Datei in das Projekt integriert sein und die Bezeichnung `readme.txt` haben
  - Falls Sie Quellen (Internet, etc.) verwenden, müssen diese angegeben werden.
    - Informationen recherchieren ist in Ordnung, aber eine fertige Lösung aus dem Internet oder einer anderen Person verwenden bzw. leicht abändern, wird als Plagiat gewertet
  - Plagiate (d.h. Kopien oder veränderte Kopien anderer Lösungen) werden mit 0 Punkten sowohl für das „Original“ als auch die „Kopie“ bewertet

Aufgaben zur Abgabe als Prüfungsleistung für Prog I

- Abgabeformat: Ihre Lösung bitte mittels OLAT einreichen
  - Ihr Projekt muss Ihren Namen haben, z.B. „`MuellerAufg1`“
  - Bitte Ihr Projekt so exportieren, dass es komplett alles enthält und problemlos wieder importiert werden kann und lauffähig ist
    - Führen Sie selbst vor Abgabe einen Probelauf durch
    - Export: Eclipse Projekt: Rechte Maustaste: Export: General: *Archive File* (dann: Save in zip Format, Finish)
    - Import: Im Package Explorer, rechte Maustaste: Import: General: *Existing Projects into Workspace*, Select Archive File (Browse `MuellerAufg1.zip`), Finish
      Hinweis: Dieser Import setzt voraus, dass das Projekt noch *nicht* existiert, deshalb das alte Projekt in Eclipse vorher *umbenennen*





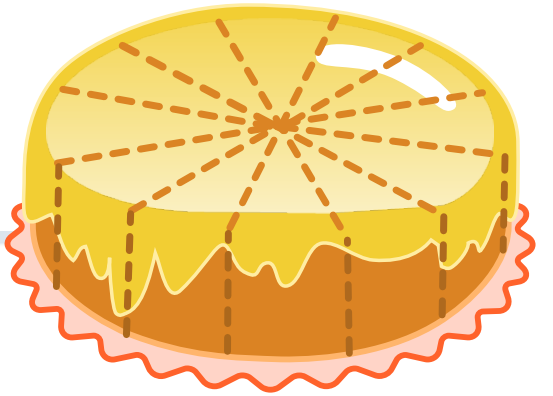
  - OLAT lässt die Veränderung der abgegebenen Dateien zu, solange die Abgabe noch *nicht* abgeschlossen wurde (Button „Endgültige Abgabe“) – bitte deshalb *keine* endgültige Abgabe vornehmen!
    - Falls Sie irrtümlich die „Endgültige Abgabe“ durchgeführt haben, dann zur Not eine neue Version direkt senden f.mehler@th-bingen.de
    - Obwohl in OLAT steht, dass nur eine endgültige Abgabe sichtbar ist, stimmt das nicht, die Abgaben sind für mich auch so verfügbar.

  - Sie können mehr als eine Datei in OLAT hochladen, z.B. zwei Entwicklungsprojekte als Java-Archiv sollten als getrennte Dateien hochgeladen werden

Vorgaben für Abgabeaufgaben

- Allgemeine Vorgaben
  - Aufgaben 1 und 2 sind mit statischen Funktionen zu programmieren; falls Sie es objektorientiert lösen möchten, dann alle Code Conventions einhalten
    - In den Aufgaben 1 und 2 dürfen keine Arrays, Listen oder ähnliches verwendet werden
  - Ausgaben exakt wie auf den Folien (Einrückungen, führende Leerzeichen etc.)
  - Trennung von Berechnung und Ausgabe: Es soll keine `print`-Anweisung innerhalb einer Berechnungs- oder ablaufsteuernden Funktion geben.
    - Sie dürfen aber jederzeit eine Ausgabefunktion aufrufen
  - Vermeiden Sie globale Variablen (d.h. Variablen, die außerhalb von Funktionen deklariert sind), nur globale Konstanten sind erlaubt
  - Lagern Sie wiederverwendbare oder zur Strukturierung sinnvolle Funktionalitäten in Funktionen oder Methoden aus
  - Achten Sie auf Qualitätskriterien (siehe Organisationsfolien und Java Code Conventions), z.B. Vermeidung von Redundanzen, hartcodierten Zahlen, Robustheit bei Berechnungen (nicht erlaubte Werte abfangen), Vermeidung von Typecasts (falls überhaupt, dann nur mit Erläuterung) bzw. besser: Typkonvertierungen

1. Aufgabe



https://de.serlo.org/mathe/49984/was-ist-ein-bruch

- Hochzusammengesetzte Zahlen (highly composite numbers, kurz: HCN) sind natürliche Zahlen, die **mehr** Teiler besitzen als jede kleinere natürliche Zahl
  - Beispiel: Die Zahl 12 ist eine hochzusammengesetzte Zahl, die 6 Teiler besitzt: 1, 2, 3, 4, 6, 12
  - Die Zahlen 1 bis 11 haben weniger als 6 Teiler
  - Gegenbeispiel: 10 ist keine HCN, da die kleinere Zahl 6 ebenfalls 4 Teiler besitzt
  - Liste der ersten HCN: https://de.wikipedia.org/wiki/Hochzusammengesetzte_Zahl

- Anwendungsbeispiele
  - Ein Dutzend Kuchenstücke lässt sich fair auf viele verschiedene Varianten aufteilen, z.B. auf 2 Personen, auf 3 Personen, auf 4 oder 6 Personen usw.
  - Die Stunden eines Tages (24), die Minuten und Sekunden (je 60) sind ebenfalls HCN, auch das Winkelgradsystem 360 ° basiert auf einer HCN
  - HCN sind quasi das Gegenstück zu Primzahlen, die nur zwei Teiler besitzen

Anforderungen

1. Ermitteln Sie mit Hilfe einer statischen Funktion in einer Klasse namens `HighlyComposite.java`, ob die Zahlen 840 bzw. 830 HCN sind oder nicht
2. Geben Sie die Liste der ersten "N" HCN mit der Anzahl der Faktoren aus, wobei N der Funktion als Parameter übergeben wird, z.B. $N = 13$
3. Analysieren Sie für eine beliebige natürliche Zahl *n*, welche Teiler ≥ 2 diese Zahl besitzt und geben alle Teiler der Größe nach aus
   - Idee des Algorithmus: Beginne mit dem Teiler 2 und dividiere *n* solange durch 2, wie die Division mod Teiler glatt aufgeht. Falls es keine Teiler mit 2 mehr gibt, nimm den nächsten Teiler 3 usw.

- Ausgaben wie folgt:
  (jede der drei Anforderungen in einer separaten Funktion, in der bereits vorhandene Funktionen aufgerufen werden dürfen)

```
Is the number 840 HCN? true
Is the number 830 HCN? false
-----------------------------------------------
HighlyComposite: 1 with number of factors: 1
HighlyComposite: 2 with number of factors: 2
HighlyComposite: 4 with number of factors: 3
…
HighlyComposite: 360 with number of factors: 24
-----------------------------------------------
Analyze factors for 840
Factor: 2, current value after div: 420
Factor: 2, current value after div: 210
Factor: 2, current value after div: 105
Factor: 3, current value after div: 35
Factor: 5, current value after div: 7
Factor: 7, current value after div: 1
```

2. Aufgabe: Simulation

- Zwei Personen Anna und Ben werfen einen Basketball mittels Freiwurf nacheinander auf einen Korb.
  - Anna hat eine Trefferwahrscheinlichkeit von 70%, Ben 40%
  - Das Spiel wird von der Person gewonnen, die als erste trifft; beide werfen abwechselnd nacheinander solange, bis das Spiel entschieden ist
  - Da Ben eine schlechtere Trefferquote hat, darf er zuerst werfen
  - Wie hoch ist die Wahrscheinlichkeit, dass Ben das Spiel gewinnt?

https://fadeawayworld.net/nba/top-10-nba-players-with-the-best-free-throw-percentage-of-all-time-stephen-curry-is-the-greatest-ft-shooter-of-all-time

Anforderungen

- Jede der folgenden Anforderungen soll in einer separaten Funktion bereitgestellt werden, in der vorhandene Funktionen aufgerufen werden dürfen
  1. Zur Simulation eines Wurfs wird ein Zufallszahlengenerator verwendet, der eine Zufallszahl zwischen 0 und 1 erzeugt
     - Falls Spieler A eine Trefferquote von 70% hat, wird z.B. getestet, ob die erzeugte Zufallszahl ≤ 0,7 ist, dann war der Wurf erfolgreich
  2. Ein Spiel wird solange durchgeführt, bis einer der beiden getroffen hat, d.h. es wird abwechselnd für jeden Spieler ein Wurf simuliert, bis einer der beiden getroffen hat
  3. Berechnen Sie mittels Simulation von 100.000 Spielen, wie hoch die Erfolgsquote für A bzw. B ist (Anzahl Gewinne von A / Anzahl Spiele)
  4. Berechnen Sie mittels Simulation von 100.000 Spielen, wie lange die Spiele im Durchschnitt dauern (d.h. die Anzahl der Würfe, die im Durchschnitt durchgeführt werden)
     - Hinweis: Da eine Funktion in Java nur einen einzigen Rückgabewert hat, müsste man für eine gemeinsame Simulation entweder Konzepte aus späteren Kapiteln zu strukturierten Datentypen verwenden oder zwei Funktionen für zwei Simulationen bereitstellen, die gewisse Redundanzen besitzen; versuchen Sie aber die Redundanzen zu minimieren z.B. durch Bereitstellung von gemeinsam genutzten Hilfsfunktionen

Anforderungen

- Die für die Simulation verwendeten Parameter (Wahrscheinlichkeiten für A, für B und Anzahl Spiele sollen zentral in der `main`-Methode der Klasse `Simulation.java` änderbar sein
- Für die Steuerung des abwechselnden Spielablaufs ist ein Aufzählungstyp `enum` sinnvoll, den Sie verwenden können, falls Sie das selbständig erarbeiten/bereits kennen, aber auch die Verwendung eines `boolean` Werts ist möglich (kein `String`, kein `int`)
- Die Ergebnisse der Simulation (Anzahl Treffer A bzw. B, Erfolgsquoten für A bzw. B, durchschnittliche Länge der Spiele) sollen auf der Konsole wie folgt ausgegeben warden
  - 2 Nachkommastellen für Erfolgsquote, 4 Nachkommas für Spiellänge
  - Typische Hilfsklassen aus den Bibliotheken (selbst recherchieren): `DecimalFormat, NumberFormat, printf`

```
Success rate of each player
XX.XX% success rate for player A
XX.XX% success rate for player B
------------------------------------------------
Average length of a game
Average number of shots per game is X.XXXX
```

Zufallszahlen erzeugen

- Um für eine Simulation geeignete Pseudozufallszahlen zu erzeugen, gehen Sie wie folgt vor:

```
// Unter das Paket einfügen:
import java.util.Random;

// seed = 1 erzeugt wiederholbare Zufallssequenzen
private static Random randomGenerator = new Random(1);

// in einer Methode:
double randomNumber = randomGenerator.nextDouble();
```

- `randomNumber` wird dann eine zufällige Zahl zwischen 0 und 1 sein