- Zorg ervoor dat je it-nummer, naam en handtekening hebt genoteerd op het blad met aanwezigheden.
- Werk in een workspace op de `E:` of `Z:` drive. Sla regelmatig op.
- Je mag enkel de parallel-website, slides uit de theorie en je boek gebruiken. Dus: **geen** internet, opgeloste oefeningen of losse papieren.
- Geen enkele vorm van communicatie is toegestaan!
- Verzorg je indentatie en gebruik duidelijke namen voor variabelen, functies, etc. Gebruik van commentaar wordt aangemoedigd. Zorg ervoor da je geen code herhaalt; de kwaliteit van je code draagt bij tot je eindtotaal.

Instructies voor importeren

- Download het bestand op `http://parallel.vub.ac.be/examen/staging-resident/exjuni2019.zip`.
- Importeer het in Eclipse:
  1. Ga naar *File* → *Import…*
  2. Selecteer *General* → *Existing projects into Workspace*
  3. Kies *Select archive file*, selecteer het gedownloadde bestand en klik op Finish.
  4. **Hernoem** je project: selecteer het project en druk F2, en noem het `exjuni2020_NaamVoornaam`.

Instructies voor indienen

- Je uploadt je oplossing op volgende URL: `http://parallel.vub.ac.be/indienen`. Exporteer je *project* als zip-bestand `exjuni2020_NaamVoornaam.zip`:
  1. Rechts-klik op je project;
  2. kies *Export*;
  3. kies *General* →*Archive File*;
  4. Geef je bestand de juiste naam in het veld *To archive file*.

  Let op: je kunt de indienprocedure slechts één maal doorlopen! Kijk goed na of je de finale versie van je code doorstuurt!
- Dit voorblad geldt als bewijs voor je aanwezigheid. Vul je *naam en rolnummer* in en nadat je indient *handteken* je deze pagina en geef je die af.

Veel succes!

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
```

IT-Nummer:
IT-Password:

Naam:

Rolnummer:

Handtekening:

Wacht met omkeren van het examen
tot het startsein wordt gegeven!

Vraag 1 (10 minuten):

In deze vraag laat je zien dat je met objecten kan werken en de basics van Java kan toepassen.

(a) Genereer willekeurige `double`s in de `public static int generate()` methode, tot hun som groter wordt dan 1000 en geeft terug hoeveel termen er nodig waren. Gebruik de standaard `Random.nextDouble()` methode zonder bijkomende argumenten.

(b) Roep bovenstaande methode 10 maal op in `public static void main(...)`; print de resultaten op tien verschillende lijnen.

Vraag 2 (20 minuten):

Voor deze vraag programmeer je een lineaire congruentiële toevalsgenerator (Eng.: linear congruential generator, LCG). Zo'n LCG onthoudt één waarde X_n en genereert de volgende waarde met

$$X_{n+1} = (aX_n + c) \mod m. \tag{1}$$

Hierin zijn a, c en m vaste waarden:

Constante	Waarde
a	1140671485
c	12820163
m	2^{24}

Vervolledig nu de `MyRandom` klasse:

(a) Voeg een `private int state;` veld toe aan de state X_n bijhoudt.

(b) Voeg twee constructors toe:

i. Eén constructor neemt als argument een initiële waarde voor X_n.

ii. De andere constructor gebruikt `System.currentTimeMillis()` als initiële waarde.

(c) Implementeer de `private void advance();` Deze methode updatet X_n volgens verg. (1).

(d) Implementeer de methode `public int nextInt();`. Deze methode roept `advance()` op en geeft de nieuwe state terug.

(e) Voeg een methode `public boolean nextBoolean()` toe, analoog aan deelvraag 2.(e), die (ongeveer) $^{50}/_{50}$ kans heeft om `true` of `false` terug te geven. Schrijf in commentaar waarom je denkt dat je methode "eerlijk" is. **Hint:** De hoogste waarde die X_n kan aannemen bedraagt m. M.a.w., $0 \leq X_n < m$.

Vraag 3 (90 minuten):

Een n-body simulatie stelt ons in staat beter te begrijpen hoe het heelal werkt, bijvoorbeeld hoe sterren en planeten zich vormen. We gaan een simpele n-body simulatie uitvoeren voor deze opgave, om de vorming van een sterrenstelsel uit een willekeurige "gaswolk" te simuleren.

Hiervoor moeten planeten kunnen samenvoegen en bewegen. Hierdoor ontstaan grotere planeten (of zelfs sterren) en bewegen de planeten na een bepaalde tijd in banen (cfr. fig. 1).

(a) Vervolledig de volgende methoden in `class Vector`:

i. `public static Vector add(Vector a, Vector b);` return $\vec{a} + \vec{b}$, de som van de argumenten.

ii. `public static Vector subtract(Vector a, Vector b);` return $\vec{a} - \vec{b}$, het verschil van de argumenten.

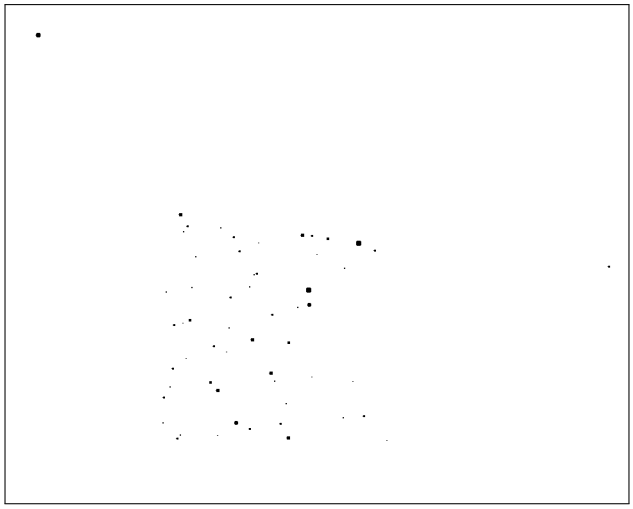
iii. `public double length();` return $|\overrightarrow{\texttt{this}}|$, de lengte van de vector.

iv. `public static double distance(Vector a, Vector b);` return $d(\vec{a}, \vec{b})$, de afstand tussen $\vec{a}$ en $\vec{b}$.

v. `public Vector multiply(double f);` return $f \cdot \overrightarrow{\texttt{this}}$, scalaire vermenigvuldiging.

Hint: Een aantal methoden van `class Vector` kunnen in functie van elkaar geschreven worden. Hergebruik methoden waar mogelijk

(b) Vul nu de constructor van `class NBodySimulation` aan: genereer 100 planeten. Elke planeet i heeft een snelheid $\vec{v}_i$, een versnelling $\vec{a}_i$ en een willekeurige positie $\vec{p}_i$ tussen `-WINDOW/2` en `WINDOW/2` voor beide coördinaatgetallen. Gebruik de daarvoor bedoelde constructor van `class Vector`. Elke planeet heeft ook een willekeurige beginmassa m_i, tussen 0.01 en 1.0. **Hint:** Maak zelf een `class Planet`.



Figuur 1 – Voorbeeld van een random planetenstelsel na enkele seconden simulatie.

(c) Teken elke planeet als een schijf met straal $r_i = \sqrt[3]{10 \cdot m_i}$. Gebruik `Graphics.fillOval(...)`

(d) In elke update stap gebeurt in drie fases:

i. Bereken de versnellingsvector $\vec{a}_i$ voor elke planeet:

$$\vec{a}_i = \sum_{j=0, j\neq i}^{N} G \frac{m_j}{d(\vec{p}_i, \vec{p}_j)^2} \vec{1}_r = \sum_{j=0, j\neq i}^{N} G \frac{m_j}{d(\vec{p}_i, \vec{p}_j)^3} (\vec{p}_j - \vec{p}_i),$$

met $G = 4.301 \times 10^{-3}\,\mathrm{km} \cdot \mathrm{M}_\odot^{-1} \cdot (\mathrm{km/s})^2$, en $d(\vec{a}, \vec{b})$ de afstand tussen twee planeten, en $\vec{1}_r$ de normaalvector die wijst van planeet i naar planeet j.

ii. Update de snelheidsvector $\vec{v}_i$ met de versnelling $\vec{a}_i$, en analoog positievector $\vec{p}_i$ met de snelheidsvector $\vec{v}_i$:

$$\vec{v}_i := \vec{v}_i + \tau \cdot \vec{a}_i$$
$$\vec{p}_i := \vec{p}_i + \tau \cdot \vec{v}_i.$$

De periode τ staat al in de code: $\tau = 10\,\mathrm{ms}$. Zorg dat de simulatie 1000 maal versneld wordt: je kan dus de periode τ als in seconden beschouwen; je hoeft nergens van eenheden te veranderen.

iii. Wanneer twee planeten dichter bij elkaar komen dan hun groottes ($d(\vec{p}_i, \vec{p}_j) < r_i + r_j$, cfr. deelvraag 3.(d)), voegen ze samen. Zorg voor behoud van impuls: de nieuwe snelheid kan je berekenen met

$$\vec{v}' = \frac{m_i \vec{v}_i + m_j \vec{v}_j}{m_i + m_j}$$

.