Beoordeling praktijkopdracht Java i

Competentie 2.2 (1, 2, 4, 5, 6, 7, 8, 11)		**Beoordeling door**	
Onderdeel	Max Score	Student	Docent
1. GUI functionaliteit			
a. Juiste opbouw GUI (statische elementen aanwezig (1 pt) juiste opbouw (1 pt))	2		
b. Juiste weergave van gevraagde GUI elementen (generieke visualisatie)	4		
2. Exception Handling			
a. Afvangen standaard Exceptions	2		
b. Custom Exception Handling	2		
3. Applicatie logica			
a. Efficiënte en correcte verwerking van bestand (lezen en parsen)	2		
b. Juiste berekening (logica)	2		
c. Code is generiek geschreven (meerdere classes (2 pt), parameters (1 pt) en return types (1 pt) korte compacte functies (1 pt))	5		
d. Code voldoet aan de Java conventies (naamgeving van classes, variabelen et cetera)	2		
4. Documentatie en commentaar			
a. Goed gebruik van commentaar inclusief Javadoc documentatie	2		
Totaal Het cijfer is het aantal behaalde punten delen door het maximaal aantal te behalen punten maal 10.	**Max: 23**		

Afbeelding beoordelingsformulier: eigen werk auteur

HAN

1.	GUI functionaliteit	
a.	Juiste opbouw GUI (statische elementen aanwezig (1 pt) juiste opbouw (1 pt))	2
b.	Juiste weergave van gevraagde GUI elementen (generieke visualisatie)	4

Bron afbeelding:
eigen werk auteur

1a: Puur statisch (zoals je programma opent).
1b: Zoal de GUI zich gedraagt. (in alle situaties).

Tip: Gebruik geen GUI builder.
Tip: Vorm, layout en volgorde heeft geen invloed op je cijfer.
Uitzondering: GUI onderdelen moeten wel zichtbaar zijn.

2. Exception Handling	
a. Afvangen standaard Exceptions	2
b. Custom Exception Handling	2

Bron afbeelding:
eigen werk auteur

2a: Afhandelen van excepties die mogelijk optreden door methoden die je worden opgelegd (zoals bij het inlezen van files).

2b: Creëer een Exception object. Bepaal wanneer jouw Exception mogelijk opgeworpen wordt. Vang je Exception Object af bij het optreden van de Exception.

3.	Applicatie logica	
a.	Efficiënte en correcte verwerking van bestand (lezen en parsen)	2
b.	Juiste berekening (logica)	2
c.	Code is generiek geschreven (meerdere classes (2 pt), parameters (1 pt) en return types (1 pt) korte compacte functies (1 pt))	5
d.	Code voldoet aan de Java conventies (naamgeving van classes, variabelen et cetera)	2

Bron afbeelding:
eigen werk auteur

Java Conventies – Opbouw Java File

- **Opbouw van een Java File van boven naar onder:**
  - Package
  - Commentaar
  - Imports
  - Classses
    - Class variables
      - Gegroepeerd per access modifier
    - Instance variables
      - Gegroepeerd per access modifier
    - Constructors
    - Methods
      - Logisch gegroepeerd (geen willekeur in getters/setters en verwante methoden)

Java Conventies – Declaratie en initialisatie

- **Declaratie en initialisatie liever apart**
  - Fiets f1; //dit geeft namelijk ruimte voor mogelijk commentaar
  - f1 = new Fiets(); //kijk hier weer commentaar
  - Fiets f1 = new Fiets(); //Dit is niet perse fout
  - Fiets f1, f2, f3, f4, f4412 = new Fiets(); //Dit is ongewenst
- **Declaraties zet je bovenin Classes en Methods**
  - Tellers in loops als uitzondering.
  - Initialisatie mag wél later.
  - Groepeer op acces modifier waar mogelijk!

Java Conventies – Naamgevingen

- Algemene conventies:
  - Zelfbeschrijvende namen
    - Voorbeeld: int studentCounter = 0; //in plaats van: int sc = 0
- Classes:
  - Hele woorden, zelfstandig naamwoorden, begint met hoofdletter
    - Voorbeeld: Class Translator //in plaats van translator
- Methods:
  - Werkwoord + zelfstandig naamwoord. Camelcase, start lowercase.
    - Voorbeeld: String getNaam(); //en berekenSnelheid() ipv bereken() of snelheid()
- Variabelen:
  - Woorden in plaats van letters, kleine letters, Camelcase, start lowercase.
    - Voorbeeld: String naam; //of String naamKlas;
  - Uitzondering: constanten (gebruik final), dan alles Uppercase
    - Voorbeeld: final int MAX_WiDTH = 100; //niet: final float pie = 3.14;

Java Conventies – inspringen/Haakjes

- **Er zijn specifieke conventies, maar..**
  - Enkel controle op consistentie.
  - Eenmaal een keuze maken en je eraan houden
- **Dus:**

Niet:

```
if (1 == 1) {
     doeBereken();
}

if (2==2)
{
  doeBereken();
}
```

Wel:

```
if (1 == 1) {
     doeBereken();
}

if (2==2) {
     doeBereken();
}
```

Ook goed:

```
if (1 == 1)
{
  doeBereken();
}

if (2==2)
{
  doeBereken();
}
```

4.	**Documentatie en commentaar**	
	a. Goed gebruik van commentaar inclusief Javadoc documentatie	2

Bron afbeelding:
eigen werk auteur

4a. Classes en Methods voorzien van javadoc.

Classes:

- Naam
- Toepassing/functie
- Auteur, Datum, Versie

Methods:

- Toepassing/functie
- Parameters
- Returnstatement

Belangrijk:

- Verwijder oud commentaar(code).
  (Tip: genereer javadoc en **vul aan**, het is niet af na genereren javadoc)
- Verwijder zinloze printstatements.
  (Tip: gebruik breakpoints en debugmodus)

Tot slot..

- **Slechts twee punten te verdienen door te “hacken” tot een juiste oplossing.**

  (Er worden onvoldoendes uitgedeeld aan werkende applicaties)
- **Leer de concepten van object oriëntatie!**
- **Begrijp je eigen (meegenomen) uitwerkingen.**

- **Er wordt op meerdere manieren gecontroleerd op plagiaat!**