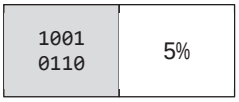
EXAMEN 42TIN1230 Java Advanced

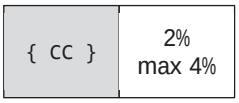
Puntenverdeling:

- Theoretisch deel (gesloten boek zonder laptop): 25%
- Praktijkvraag (open boek met laptop): 75%

Bij het verbeteren van de praktijkvraag wordt er rekening gehouden met de onderstaande criteria.



Niet compileerbare code of niet uitvoerbare code leidt tot een malus van 5% procent op het resultaat van de opgave.



Het niet volgen van de gangbare codeconventies leidt tot een malus van 2% per fout met een maximum van 4% op het resultaat van de opgave.

Het niet volgen van de specificatie (naamgeving van bestanden, klassen, variabelen, …) zoals gevraagd in de onderstaande implementatiedetails, leidt tot een malus van 2% per fout met een maximum van 4%.

De percentages worden eerst opgeteld vooraleer de malus berekend wordt.
Je behaalt 15/20, maar het compileert niet (5%), er zijn 4% codeconventiefouten en 4% specificatiefouten → Dit geeft een malus van 13% wat resulteert in 13.05/20.

Elk maluspercentage wordt PER VRAAG berekend.

Praktijkgedeelte

Lees eerst aandachtig de volledige opgave.

STAPPEN UIT TE VOEREN VOOR JE MET HET LAPTOPGEDEELTE BEGINT

Maak in Eclipse een **java project** met als naam <familienaam_voornaam_JA_zit1>.
Voorbeeld: een student met naam Jan Peeters maakt een project met naam *peeters_jan_JA_zit1*.

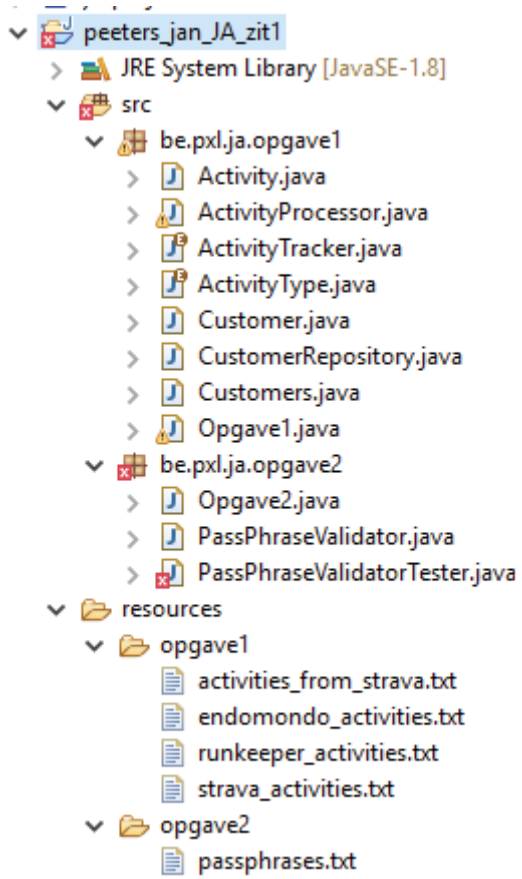
Voor iedere vraag maak je een aparte **package** aan met als naam <be.pxl.ja.opgaveX>.
Voorbeeld: voor opgave 1 maak je een package met naam *be.pxl.ja.opgave1*.

Voor je aan het examen begint haal je via FileZilla enkele **bestanden** binnen voor beide vragen:

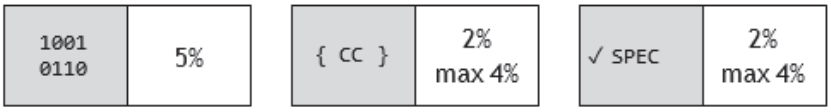
- Folder opgave1: bevat de klassen voor opgave1, deze klassen plaats je in de package be.pxl.ja.opgave1
- Folder opgave2: bevat de klassen voor opgave2, deze klassen plaats je in de package be.pxl.ja.opgave2
- Folder resources plaats je in de project directory.

In je oplossingen moeten deze bestanden gebruikt worden.
Plaats bovenaan in elke file je naam in **commentaar**.

Je startsituatie ziet er nu als volgt uit:



Opgave 1 (30 pt)



Je implementeert een systeem voor een sportwinkel waarbij klanten punten kunnen sparen voor hun sportactiviteiten. De gegevens van de sportactiviteiten worden geleverd door 2 activity tracking toepassingen: endomondo en strava.

Implementatiedetails

De volgende enums en klassen zijn reeds voorzien in het project:

- In package be.pxl.ja.opgave1
  - Een enum **ActivityTracker** met de twee ondersteunde activity trackers: ENDOMONDO en STRAVA. Het is toegelaten om voor enum-waarden extra eigenschappen te voorzien.
  - Een enum **ActivityType** met de waarden RUNNING, RIDING en SWIMMING. Bij iedere enum-waarde staat het aantal punten dat een sporter krijgt per afgelegde kilometer. Zo verdien je bijvoorbeeld 3 punten per gelopen kilometer.
  - De klasse **Customer** om de gegevens van een klant van de sportwinkel bij te houden.
  - De klasse **Activity.** De klasse bevat de volgende eigenschappen voor een sportactiviteit:
    - customerNumber: de klantnummer van de sporter die de activiteit heeft uitgevoerd.
    - activityType: het type activiteit dat de sporter heeft uitgevoerd.
    - activityDate: de datum (LocalDate) dat de activiteit werd uitgevoerd.
    - distance: de afstand in kilometer die de sporter heeft afgelegd.
    - tracker: de activityTracker die werd gebruikt om de activiteit te registreren.
  - De klasse **Customers** bevat een static lijst met alle klantgegevens.

    - In de constructor van de klasse **CustomerRepository** worden alle klanten uit de klasse Customers in een HashMap opgeslaan, zodat klanten gemakkelijk adhv hun klantnummer opgezocht kunnen worden.
- In de folder "resources/opgave1" vind je bestanden terug met de gegevens van sportactiviteiten.

Let op dat je oplossing leesbaar en onderhoudbaar is. Indien gewenst mag je zelf extra klassen en interfaces voorzien.

De klasse 'CustomerRepository'

Vervolledig de klasse **CustomerRepository**. Voorzie een implementatie voor de methoden *getByCustomerNumber(String customerNumber)* en *findAll()*. De eerste methode geeft de klant met het gegeven klantnummer terug als resultaat. Indien er geen klant bestaat met het gegeven klantnummer geef je *null* als resultaat. De tweede methode voorziet een lijst met alle klanten.

De klasse 'ActivityProcessor'

In de klasse **ActivityProcessor** gaan we de functionaliteit voorzien om een bestand met gegevens van sportactiviteiten te verwerken. Je hebt de customerRepository in de klasse ter beschikking om tijdens het verwerken van de sportactiviteiten punten toe te kennen aan de klanten. In de methode *processActivities* zijn 2 parameters voorzien: activityFile bevat het Path voor een bestand met activiteiten, errorFile bevat het Path waar je alle foutmeldingen naar wegschrijft. De methode *processActivities* retourneert een lijst met alle correct verwerkte activiteiten.

Wanneer je een foutmelding wegschrijft start je met datum en tijdstip van de foutmelding, vervolgens het bestand dat je verwerkt en tenslotte een omschrijving van het probleem. Zorg ervoor dat het bestand voor foutmeldingen wordt aangemaakt indien dit nog niet bestaat. Indien het bestand wel al bestaat ga je de informatie toevoegen.

Je mag een activityFile enkel verwerken indien de bestandsnaam “strava” of “endomondo” bevat. Bij een ongeldige bestandsnaam verschijnt er bijvoorbeeld onderstaande foutboodschap in de errorFile en wordt het bestand niet verwerkt.

```
2018-01-03T09:32:53.225 - runkeeper_activities.txt - INVALID FILENAME
```

Let er wel op dat strava- en endomondo-bestanden een ander formaat gebruiken voor de sportactiviteiten.
Strava voorziet de gegevens in het volgende formaat:

- Eerst de voornaam, naam en klantennummer gevolgd door een ;
- Daarna de datum in formaat dd/MM/yyyy opnieuw gevolgd door een ;
- Vervolgens het type activiteit gevolgd door ;
- En tenslotte de afgelegde afstand. Voor zwemmen (swimming) is dit uitgedrukt in meter, voor fietsen (running) en lopen (riding) in kilometer.

```
Marlene Rhodes 209064159;06/01/2018;swimming;1468
Jessica Ross 209064561;19/01/2018;running;18.0
Kyle Cannon 209067589;06/01/2018;running;25.4
Ronald Reed 209066419;10/01/2018;riding;97.5
```

Endomondo levert de gegevens aan in het volgende formaat:

- Eerst de datum van de activiteit in formaat yyyyMMdd gevolgd door ;
- Daarna het klantennummer gevolgd door ;
- Vervolgens het type activiteit en de afgelegde afstand (zelfde eenheid als bij strava) gescheiden door ;

```
20180116;209066371;running;9.8
20180116;209067026;swimming;282
20180107;209066729;riding;31.6
```

Voor iedere afgelegde kilometer ken je punten toe aan de betreffende klant. Het aantal punten vind je terug in de enum **ActivityType**. Enkele voorbeelden:

```
129.5km fietsen -> 129 punten
2549m zwemmen -> 10 punten
9.3km lopen -> 27 punten
```

Indien een klantnummer ongeldig is, geef je een gepaste foutboodschap in de errorFile:

```
2018-01-03T10:06:34.304 - activities_from_strava.txt - UNKOWN CUSTOMER:
Curtis Farley 20067666;11/01/2018;running;24.1
```

Indien andere gegevens niet correct verwerkt kunnen worden, schrijf je de boodschap (getMessage()) van de exception ook weg naar de errorFile:

```
2018-01-03T09:54:37.144 - activities_from_strava.txt - Text '24122017' could not
be parsed at index 2
2018-01-03T09:54:37.163 - endomondo_activities.txt - No enum constant
be.pxl.ja.opgave1.ActivityType.TREADMILL
2018-01-03T09:54:37.171 - endomondo_activities.txt - Text '20171316' could not be
parsed: Invalid value for MonthOfYear (valid values 1 - 12): 13
2018-01-03T09:54:37.172 - endomondo_activities.txt - Text '2017129' could not be
parsed at index 6
```

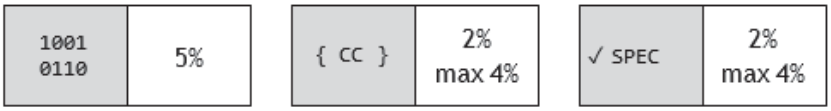
De klasse 'Opgave1'

De klasse **Opgave1** bevat het hoofdprogramma.

Voeg nu volgende functionaliteit toe in het hoofdprogramma. Voor deelvragen 1,2,3,5 en 6 maak je, indien mogelijk, gebruik van streams en lambda expressies.

1. Hoeveel klanten wonen in Louisville?
2. Toon de naam, voornaam en geboortedatum van de klanten die vandaag jarig zijn.
3. Toon de naam, voornaam en geboortedatum van de 10 jongste klanten.
4. Verwerk nu alle bestanden met activiteiten uit de folder "resources/opgave1" (geen subdirectories verwerken!) mbv de methode *processActivities* uit de klasse **ActivityProcessor**. Zorg dat je alle verwerkte activiteiten verzamelt in een lijst. Zorg dat je programma ook werkt als er bestanden met activiteiten worden toegevoegd en verwijderd. Als errorFile gebruik je het bestand "resources/opgave1/log/errors.log".
5. Maak een top 10 van klanten met het meeste punten.
6. Van de meest actieve klant (klant met het hoogste aantal punten) toon je de datum, type en punten van zijn activiteiten. De activiteiten sorteer je op datum (meest recente activiteit eerst).

Opgave 2 (15pt)



In deze opgave implementeren we een thread die een wachtwoordzin (passphrase) kan valideren. De wachtwoordzin kan een lijst met integer waarden of een lijst met string waarden zijn (of nog een ander datatype indien gewenst). Een wachtwoordzin is geldig als er GEEN dubbels in staan.

In de klasse **PassPhraseValidatorTester** wordt de thread 4 keer opgestart.
De wachtwoordzin *(12, 18, 15, 32)* is geldig. De wachtwoordzin *(12, 18, 15, 18, 32)* is niet geldig omdat 18 dubbel is. De wachtwoordzin *("kat", "olifant", "wombat", "ooievaar")* is dan weer geldig en tenslotte is de wachtwoordzin *("kat", "wombat", "kat", "olifant", "wombat", "ooievaar", "kat")* niet geldig omdat zowel "kat" als "wombat" meermaals voorkomen.

Implementeer nu de generieke thread **PassPhraseValidator** en pas de **PassPhraseValidatorTester** verder aan, zodat deze volgende output geeft:

```
[12, 18, 15, 32] valid: true
[12, 18, 15, 18, 32] valid: false
[kat, olifant, wombat, ooievaar] valid: true
[kat, wombat, kat, olifant, wombat, ooievaar, kat] valid: false
```

De klasse **PassPhraseValidator** is afgeleid van de klasse Thread en maakt gebruik van generieke datatypes.

In het hoofdprogramma **Opgave2** lees je het bestand "resources/opgave2/passphrases.txt" in. Iedere lijn in het bestand bevat een wachtwoordzin. Maak van een ingelezen wachtwoordzin een lijst van String-waarden en laat deze lijst valideren door een **PassPhraseValidator**. Het hoofdprogramma geeft als resultaat hoeveel GELDIGE wachtwoordzinnen het gegeven bestand bevat?
Gewenste output van Opgave2:

```
Aantal geldige wachtwoordzinnen: 466
```