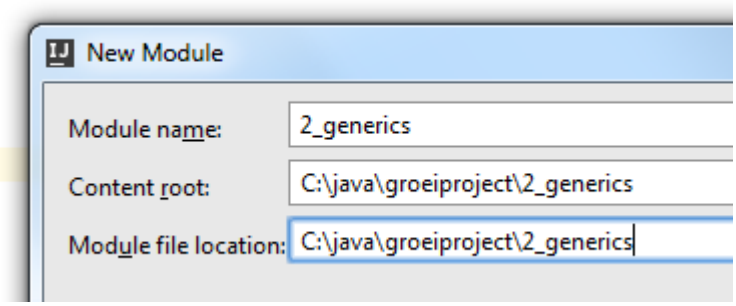
GROEIPROJECT
MODULE 2: GENERICS EN DOCUMENTEREN

1. Voorbereiding

1.1. Open het groeiproject en maak daarin een tweede module: `2_generics`
Let op dat de locatie van de module net onder de project map is (hier: "groeiproject"):



1.2. Kopieer vanuit de vorige module de hele package `be.kdg...model` (alle klassen behalve de multiklasse) en de package `be.kdg...data` naar de src map van de nieuwe module.
De multiklasse `Dicatators` zullen we deze week niet nodig hebben!

2. Generics

We gaan zelf onze eigen generieke datastructuur bouwen met de naam "**PriorityQueue**". Deze klasse bestaat eigenlijk al in de Java JDK (`java.util.PriorityQueue`), maar ze werkt anders dan we zouden willen, dus we maken deze klasse opnieuw zelf *from scratch*!
Werking:

- **enqueue**: Elk element dat aan de queue wordt toegevoegd krijgt een bepaalde prioriteit mee (1..5) en wordt volgens die prioriteit (hoogste eerst) in de queue gestockeerd. Elementen met dezelfde prioriteit worden in hun reeks volgens het FIFO-principe toegevoegd.
- **dequeue**: In de reeks met de hoogste prioriteit wordt het oudste element verwijderd (FIFO).

2.1. Maak onder een nieuwe package `generics` de interface `FIFOQueue`.
Neem de volgende code gewoon over:

```
public interface FIFOQueue<T> {

    boolean enqueue(T element, int priority);

    T dequeue();

    int search(T element);

    int getSize();

}
```

2.2. Maak in dezelfde package `generics` de klasse `PriorityQueue` die bovenstaande interface implementeert. Opgelet! Deze klasse moet generiek zijn voor elk Object-type; gebruik dus <T>

2.3. We zullen als onderliggende implementatie werken met een `TreeMap`: de **key** is het prioriteitsgetal (1..5) en de **value** is een `LinkedList` van elementen met dezelfde prioriteit.
Declareer deze `TreeMap` als private attribuut en zorg ervoor dat de keys in dalende volgorde staan gesorteerd (TIP: `Comparator.reverseOrder`)

2.4. Implementeer nu alle methoden van de interface `FIFOQueue`:

- **enqueue:** Het element en de prioriteit komen als parameters binnen. Onderzoek eerst of het element nog niet voorkomt in de map (ongeacht de prioriteit). Indien niet, dan voeg je het achteraan toe in de juiste prioriteitsreeks.
- **dequeue**: Retourneer het element dat zich vooraan bevindt in de hoogste prioriteitsreeks.
- **search**: Zoek het element op in de map. Indien gevonden, dan retourneer je de positie in de queue. Indien niet gevonden: `-1`
- **getSize**: Geef het totaal aantal elementen in de queue.
- **toString**: Genereer een overzicht van de `PriorityQueue` in de vorm van een string met op elke afzonderlijke regel: het prioriteitsgetal, gevolgd door een : en daarachter het element zelf.
  Voorbeeld voor een PriorityQueue van strings:

```
5: beta
3: delta
2: alfa
2: gamma
```

2.5. Maak nu een aparte klasse `Demo_2` met daarin een `main`.
Test eerst met een `PriorityQueue` van strings. Neem deze testcode over:

```
PriorityQueue<String> myQueue = new PriorityQueue<>();
myQueue.enqueue("alfa", 2);
myQueue.enqueue("beta", 5);
myQueue.enqueue("gamma", 2);
myQueue.enqueue("delta", 3);

System.out.println("Overzicht van de PriorityQueue:");
System.out.println(myQueue.toString());
System.out.println("aantal: " + myQueue.getSize());
System.out.println("positie van gamma: " + myQueue.search("gamma"));

for(int i = 0; i < 4; i++) {
    System.out.println("Dequeue: " + myQueue.dequeue());
}
System.out.println("Size na dequeue: " + myQueue.getSize());
```

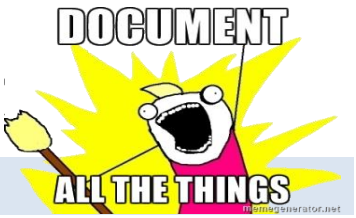
2.6. Verwachte output (gaat voort op volgende pagina):

```
Overzicht van de PriorityQueue:
5: beta
3: delta
2: alfa
2: gamma

aantal: 4
positie van gamma: 4
Dequeue: beta
Dequeue: delta
Dequeue: alfa
Dequeue: gamma
```

```
Size na dequeue: 0
```

2.7. Maak nu een `PriorityQueue` met elementen van de basisklasse (In ons geval dus een `PriorityQueue` van `Dictator`-objecten). Gebruik de klasse `Data` om elementen aan de `PriorityQueue` toe te voegen en geef telkens als prioriteit een randomwaarde mee (1..5). Probeer alle functionaliteiten uit!

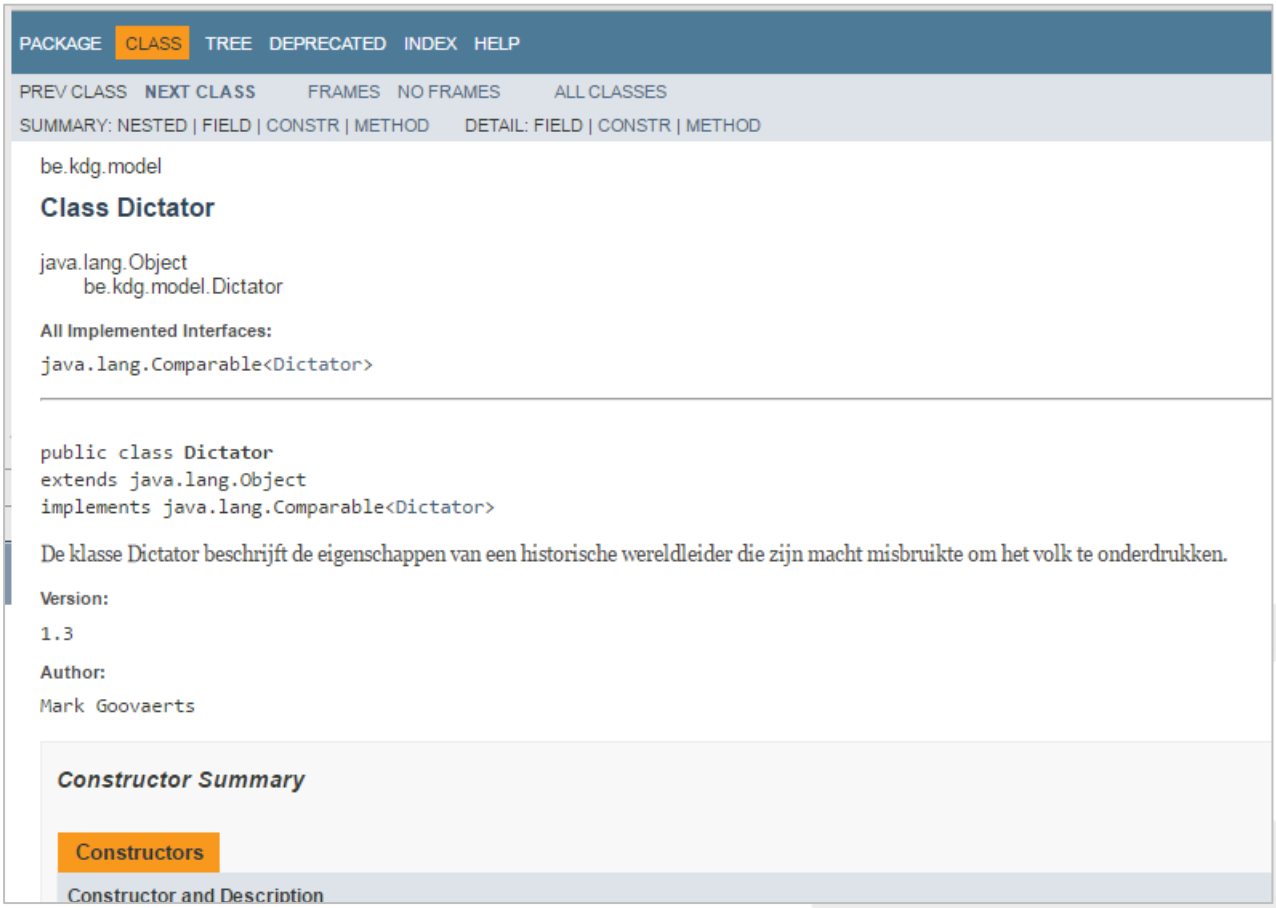


3. Javadoc

3.1. Schrijf **Javadoc**-commentaar voor de basisklasse (in ons geval: `Dictator`) en de klasse `PriorityQueue`:

- Schrijf een class-header met een korte beschrijving en een `@author` en een `@version`-tag
- Schrijf boven elke contructor en elke methode gepaste commentaar. Voor een property (attribuut met een simpele getter/setter) mag je in de plaats ook het attribuut documenteren. Merk op dat IntelliJ zelf de gepaste Javadoc-tags voorziet.

3.2. Ga naar de menukeuze "*Tools* > *Generate JavaDoc*" en maak enkel documentatie voor de module `2_generics` (Dus kiezen voor "*Custom scope*"). Plaats de HTML-documenten in een nieuwe submap **`doc`** onder je project. Controleer het resultaat in een webbrowser:


PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

be.kdg.model

Class Dictator

java.lang.Object
be.kdg.model.Dictator

All Implemented Interfaces:

`java.lang.Comparable<Dictator>`

```
public class Dictator
extends java.lang.Object
implements java.lang.Comparable<Dictator>
```

De klasse Dictator beschrijft de eigenschappen van een historische wereldleider die zijn macht misbruikte om het volk te onderdrukken.

Version:

`1.3`

Author:

`Mark Goovaerts`

Constructor Summary

Constructors

Constructor and Description