STUDENT NAME		EDUCATION	
STUDENT NUMBER			

Test name	Object-Oriented Programming 2
Subject code	OOP2
Test date	
Test time	
Examiners	Martijn Pomp, Jan Doornbos
Test duration	180 minutes
Number of exercies/questions	6
☐ Answer form ☐ Answer sheet ☐ Writing paper ☐ On the test itself ☒ Git repository	

Maximum attainable points		100	
Number of points with which the test was passed		55	
Permitted aids			
☐ none	☐ textbook	☐ calculator	☒ printed coding conventions
☒ draft paper	☐ law book	☐ graphics calculator	☐
☐ graph paper	☐ dictionary	☒ IntelliJ (without plugins)	☐

General test instructions:

- Write your details clearly and correctly in your repository (README.md).
- Warn the invigilator if something is unclear about the test.
- Hand in all the material when leaving the test room.

If, during the test, you have a complaint about the contents of the test or about how the test is held, you must submit your complaint in writing within 2 working days to the relevant Examination Committee.

My school can trust the fact that I took this test independently without the help of others and that I have only used the tools and aids that the examiner has allowed me to do.

CASE SMARTBBQ

With the current corona crisis, a well-known BBQ manufacturer sees a gap in the market. To prevent many people from gathering around a BBQ, they have an idea to create a SmartBBQ™. The following paragraphs list the main features of the program.

SMARTBBQ™

The SmartBBQ™ can be set to an exact temperature in whole degrees Celsius, with a maximum of 250 degrees Celsius.
By default, the BBQ is off. A maximum of six food items can be placed on the SmartBBQ™. This is predetermined.
When one turns on the SmartBBQ™ with the `turnOn` function, the temperature should be given along with it.

MEAT

Different types of meat can be put on the barbecue. For the barbecue, it is important to know the following characteristics:

- Current cooking percentage **(0 - ∞)**
- Current tanning percentage **(0 - ∞)**
- Meat origin: Chicken/Cow/Pig

Also, the meat contains a `toString()` method that returns the name of the meat.

VEGETABLES

Not only meat can go on the barbecue, but also other foods. In this case, vegetables.
For the barbecue, it is important to know the following characteristics:

- Current tanning percentage **(0 - ∞)**
- Moisture percentage **(100% - 0 %)**

Also, the vegetable contains a `toString()` method that returns the name of the vegetable.

FOOD

The food items have the following parameters. Where T_{BBQ} the current temperature of the SmartBBQ™ is. The food items get a `grill` method that grills the food item for one second at a certain temperature. The temperature of the food item increases by 0.5 degree per second. Other properties are in the table below.

Table 1 Food parameters

FOOD	COOKED INCREASE /SECOND	TAN INCREASE /SECOND	MOISTER DECREASE/ SECOND	MEAT ORIGIN
HAMBURGER	$\frac{1}{100 * \pi} * T_{BBQ}$	$0.06 * \frac{T_{BBQ}}{100}$	n.v.t.	Koe
SAUSAGE	$\frac{1}{80 * \pi} * T_{BBQ}$	$0.05 * \frac{T_{BBQ}}{100}$	n.v.t.	Varken
BELL PEPER	n.v.t.	$0.03 * T_{BBQ}$	$0.1 * T_{BBQ}$	n.v.t.
CORN	n.v.t.	$0.08 * T_{BBQ}$	$0.05 * T_{BBQ}$	n.v.t.

FRIDGE

There is one fridge (singleton) in which food is placed and taken out. The fridge can hold an infinite amount of food. The user can only choose whether he wants meat or vegetables. Not which specific kind. The user then gets the first available food item of the type he chose.

If no more food is available of the selected type, an exception is raised: `NoMoreFoodException`.

The temperature of the fridge can be set, but defaults to 8 degrees Celsius.

Your colleague has already created a foundation with the class diagram. You can find these in Appendix 2. Unfortunately, he spilled a little coffee.

ASSIGNMENT 1

Build the application based on the given class diagram and the text above.

ASSIGNMENT 2

The `turnOn` feature also starts a timer that triggers the SmartBBQ™ to cook the food. This is done through a Timer and a TimerTask (see Appendix 1: Timer and TimerTask class information). The timer simulates one second of the BBQ process each time. The implementation of this must be done **in** the SmartBBQ™. Make this implementation.

ASSIGNMENT 3

After testing the SmartBBQ™ and the fridge for some time, it happens quite often that the fridge goes empty unnoticed. The manufacturer would like a notification towards the user through an exception: the `FridgeIsEmptyException` this is thrown as soon as you want to take something out of an empty fridge. Implement this.

ASSIGNMENT 4

Included with the SmartBBQ™ is a thermometer. You can use this thermometer to query the temperature of different entities. So, you can use this thermometer to retrieve the temperature of the fridge, the BBQ and the food items.
Create a thermometer class with one method. To this method you can pass one of the mentioned elements. Then this method returns the following: a string containing the text: "`I measured a temperature of xx degrees Celsius`", *and* a classification of temperature. Of these, there are three possibilities: cold, medium, hot. Anything below 10 degrees is cold. Between 10 and 70 degrees is medium. Anything above that is hot.

ASSIGNMENT 5

a) Due to an incorrect implementation in the past, by a student not from NHL Stenden, the good meat was not always caught from the fridge. The company is now asking for a meaningful Unit Test that tests the `getNextMeat` method.
b) There are also times when vegetables are too dry to eat. Again, the manufacturer of the SmartBBQ™ wants a meaningful Unit Test for testing the moisture percentage. Make this.

ASSIGNMENT 6

Create a Main class with a static main method that:

- Makes a SmartBBQ™;
- Puts three corn cobs and two bell peppers in the fridge;
- Put two hamburgers and three sausages in the fridge;
- Put three pieces of meat on the SmartBBQ™;
- Then turn on the SmartBBQ™ with a temperature of 180 degrees Celsius;
- Measure the temperature of piece of meat;
- Increase the temperature of the SmartBBQ™ to 200 degrees Celsius;
- Measure the temperature of piece of meat.

APPENDIX 1: TIMER AND TIMERTASK CLASS INFORMATION

java.util.Timer class

Introduction

The **java.util.Timer** class provides facility for threads to schedule tasks for future execution in a background thread.

- This class is thread-safe i.e multiple threads can share a single Timer object without the need for external synchronization.
- This class schedules tasks for one-time execution, or for repeated execution at regular intervals.
- All constructors start a timer thread.

Constructor & Description
Timer() This constructor creates a new timer.

Method & Description
void schedule(TimerTask task, long delay, long period) This method schedules the specified task for repeated fixed-delay execution, beginning after the specified delay.

java.util.TimerTask class

Introduction

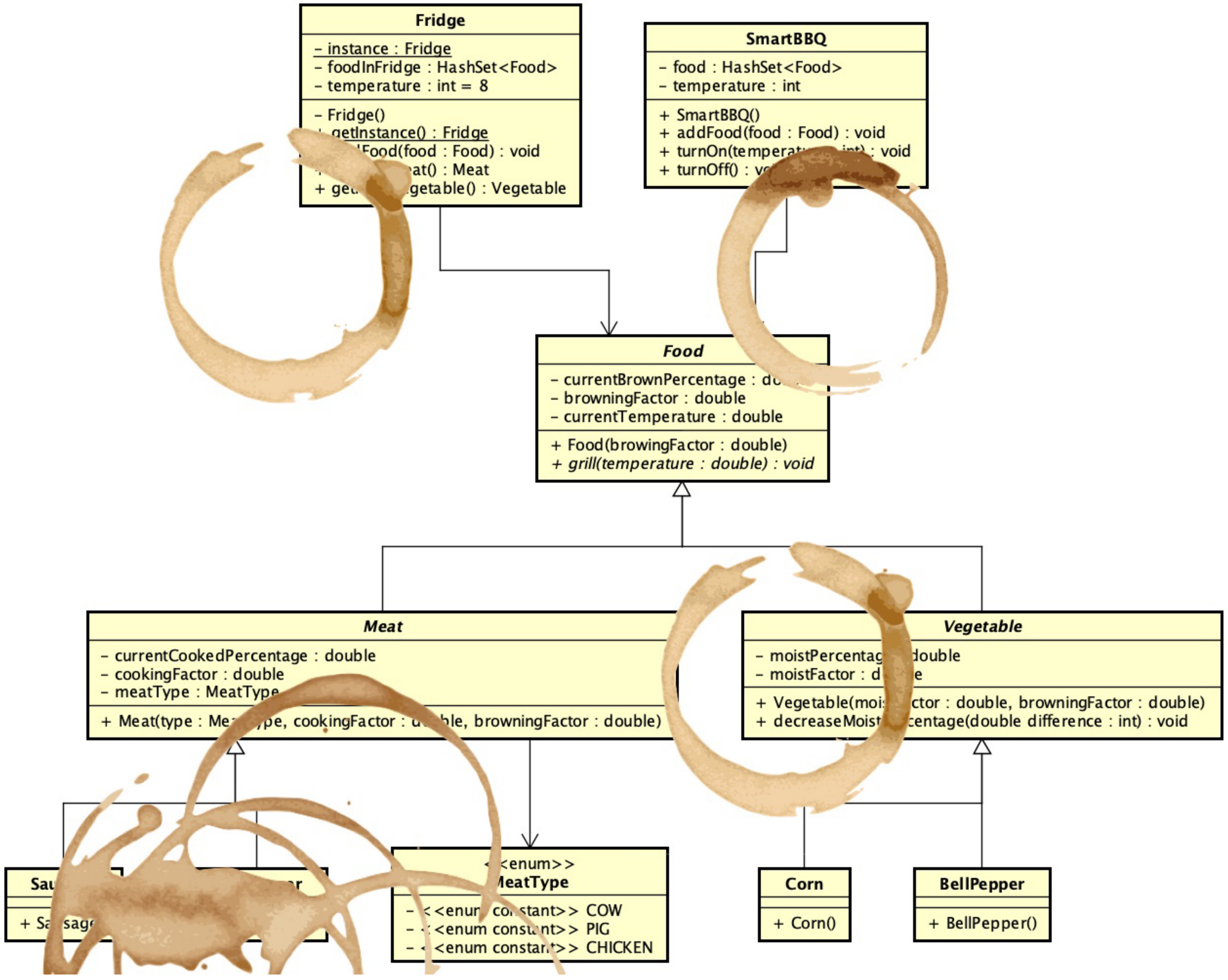
The **java.util.TimerTask** class represents a task that can be scheduled for one-time or repeated execution by a Timer. *Every class can be extended from TimerTask*.

Constructor & Description
TimerTask() This constructor creates a new timer task.

Method & Description
abstract void run() This method represents the action to be performed by this timer task.

NHL STENDEN hogeschool

APPENDIX 2: CLASS DIAGRAM


Fridge
– instance : Fridge
– foodInFridge : HashSet<Food>
– temperature : int = 8
– Fridge()
+ getInstance() : Fridge
lFood(food : Food) : void
at() : Meat
+ get getable() : Vegetable
SmartBBQ
– food : HashSet<Food>
– temperature : int
+ SmartBBQ()
+ addFood(food : Food) : void
+ turnOn(tempera int) : void
+ turnOff() : v
Food
– currentBrownPercentage : do
– browningFactor : double
– currentTemperature : double
+ Food(browingFactor : double)
+ grill(temperature : double) : void
Meat
– currentCookedPercentage : double
– cookingFactor : double
– meatType : MeatType
+ Meat(type : Me ype, cookingFactor : ble, browningFactor : double)
Vegetable
– moistPercentag double
– moistFactor : d e
+ Vegetable(mois ctor : double, browningFactor : double)
+ decreaseMoist centage(double difference : int) : void
Sau
+ Sa sag
r
<<enum>>
MeatType
– <<enum constant>> COW
– <<enum constant>> PIG
– <<enum constant>> CHICKEN
Corn
+ Corn()
BellPepper
+ BellPepper()