CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

Fachhochschul-Akademischer Lehrgang „Software Engineering“

Klausur aus der Lehrveranstaltung

„PROGRAMMIEREN 2“

Hofer Christian

Wild Karoline

Hauptantritt am 31.05.2023
1.Termin

1. SEMESTER

SS 2023

Name: _____

Beurteilung: _____

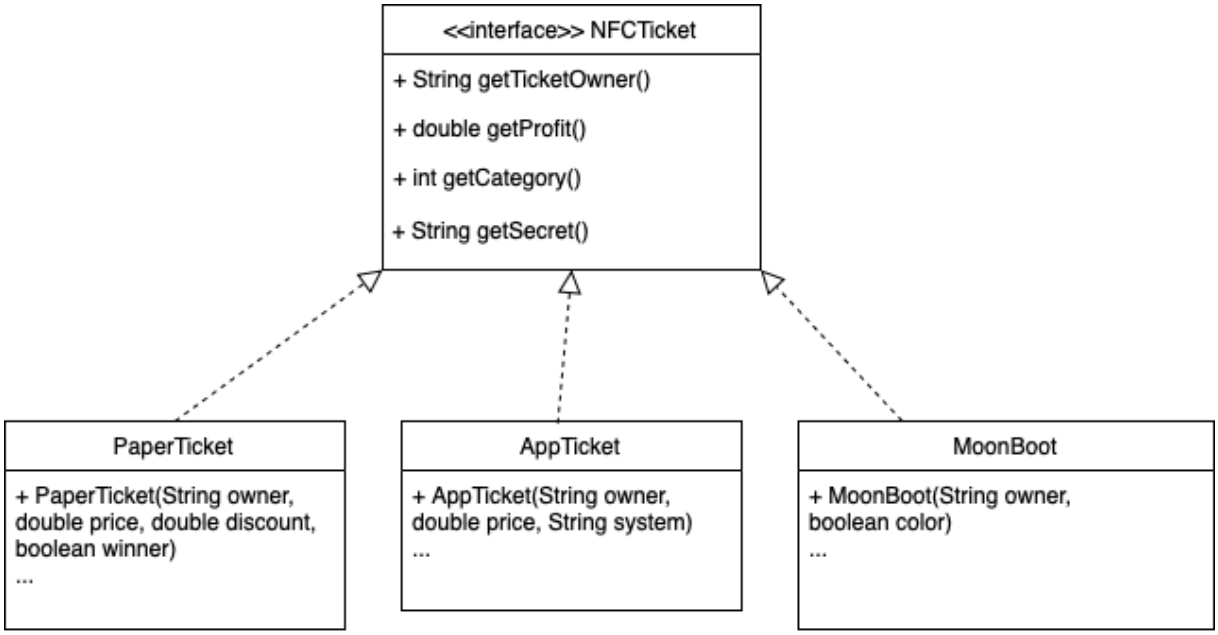
Klausurdauer: 180 Minuten

Sie dürfen alle Ihre Unterlagen, alte Programme und Internetquellen verwenden. Kommunikation, Austausch in jedweder Art (analog oder digital) oder andere Tools sind nicht erlaubt. Zu Beginn erhalten Sie Ihr Startparket. Bitte geben Sie am Ende der Zeit Ihr Projektverzeichnis als ZIP Datei über Moodle ab. Nur für compilierbaren und funktionsfähigen Code erhalten Sie Punkte!

Halten Sie sich an die Angaben: Keine Veränderung vorgegebener Methodennamen oder Struktur, Textausgabe statt return Wert, …

Hansi Hinterseer Konzert

Für die aktuelle Jubiläumskonzerttour von Hansi Hinterseer sollen Sie ein Verwaltungssystem für eines seiner Konzerte erstellen. Um den Handel mit gefälschten Tickets zu unterbinden benötigt jede Art von Ticket einen NFC Funkchip.



Es gibt drei unterschiedliche Ticketarten, die alle die Sicherheitsmethoden des **Interfaces NFCTicket** implementieren müssen um den unerlaubten Handel am Schwarzmarkt zu unterbinden. Erstellen Sie ein Interface mit den oben spezifizierten Methoden. Danach erstellen Sie die Konstuktoren: **(11 Punkte)**

PaperTicket: Die Klasse PaperTicket steht für ein klassisches Papierticket (mit integriertem NFC Sicherheitschip) und der Konstruktor wird mit folgenden Parametern initialisiert:

- **owner:** Name Ticketkäufer*in
- **price:** Ticketpreis (double)
- **discount:** Rabatt in Prozent (double 20.5 wird als 20.5 % interpretiert)
- **winner:** Papiertickets nehmen bei einem Gewinnspiel teil und Parameter gibt an ob Ticket gewonnen hat -> true, sonst false

Speichern Sie alle vier Eigenschaften in der Klasse auf protected Attributen (wählen Sie nachvollziehbare Namen). Zusätzlich müssen Sie im Konstruktor noch einen **Sicherheitscode** generieren (siehe Hinweis nächste Seite) und auf einem weiteren protected Attribut vom Typ **String** in der Klasse speichern.

Hinweis: Zum Generieren von Sicherheitscodes haben wir die Klasse **TicketFraudSystem** von Bunny Security Systems zugekauft. Sie bietet eine statische Methode *String createNewTicket(String type)* die uns diesen Sicherheitscode als Rückgabewert liefert. (Sie müssen den Code in der Methode nicht nachvollziehen.)

Für ein PaperTicket geben Sie in der zuvor genannten Methode beim Parameter type den String ***"paper"*** mit.

AppTicket: Die Klasse AppTicket steht für ein Ticket, welches mobil auf einem modernen Smartphone gekauft wurde und mit folgenden Eigenschaften initialisiert wird:

- **owner:** Name Ticketkäufer*in
- **price:** Ticketpreis
- **system:** Gibt den Namen des Smartphone Betriebssystems an.

Speichern Sie diese Eigenschaften in der Klasse auf selbst erstellten protected Attributen. Zusätzlich müssen Sie im Konstruktor noch einen **Sicherheitscode** generieren und auf einer eigenen Eigenschaft (**String**) in der Klasse speichern. Verwenden Sie erneut *createNewTicket* mit dem Parameter ***"app".***

MoonBoot: Markenzeichen von Hansi sind seine MoonBoots (=Winterschuhe). Für besonders treue Fans gibt es ein kleines MoonBoot Souvenir, dass einerseits Konzertkarte ist und andererseits auch als Stimmungslicht zu Hause bzw. während des Konzerts verwendet werden kann. Initialisieren Sie mit folgenden Eigenschaften:

- **owner:** Name Ticketkäufer*in
- **color:** Falls **true** dann ist es das Modell mit Beleuchtung in unterschiedlichen Farben, falls **false** kann es nur in einer Farbe leuchten.

Speichern Sie diese beiden Eigenschaften in der Klasse auf protected Attributen. Zusätzlich müssen Sie im Konstruktor noch einen **Sicherheitscode** generieren und auf einem protected Attribut (**String**) in der Klasse speichern. Verwenden Sie erneut *createNewTicket* mit dem Parameter ***"moonboot".***

Setzen Sie die abstrakten Methoden in den drei Klassen um:

getTicketOwner soll Besitzer*in als Rückgabewert liefern. **(1 Punkt)**

getSecret soll den Sicherheitscode des Tickets als Rückgabewert liefern. **(1 Punkt)**

getProfit soll den Gewinn (=Profit) der mit dem Verkauf des Tickets gemacht wurde als Rückgabewert liefern: **(7 Punkte)**

- **PaperTicket:** Vom Ticketpreis müssen zuerst 20 Euro Händlerkosten abgezogen werden und danach der Rabatt abgezogen werden um den Profit zu erhalten.
- **AppTicket:** Wenn das Betriebssystem ***"ios"*** ist dann ist der Profit 70 % des Ticketpreises. Bei „**android**“ 75 Prozent und bei allen anderen Betriebssystemen 80 % des Ticketpreises.
- **MoonBoot:** Die MoonBoot Modelle werden nur direkt verkauft – für ein Modell mit farbigem Licht gibt es 250 Euro Profit, für ein Modell mit einer Farbe 200 Euro.

getCategory soll die Ticketkategorie des Tickets zurückliefern: **(7 Punkte)**

- **PaperTicket:** Tickets mit einem Preis von mindestens 80 Euro erlauben Zugang zu Plätzen der Kategorie 2 – sonst Kategorie 3. Falls man beim Ticketkauf das Gewinnspiel gewonnen hat wird die Kategorie weiters um eine Kategorie gesenkt.
- **AppTicket:** Tickets mit Preisen von mindestens 100 Euro erlauben Zutritt zur Kategorie 1 bzw. bei Tickets ab 50 und unter 100 Euro Kategorie 2 bzw. sonst Kategorie 3.
- **MoonBoot:** MoonBoot mit farbiger Beleuchtung erlauben Zutritt zu Kategorie 1, mit einfarbigem zu Kategorie 2.

Die Klasse **ConcertSystem** soll eine ArrayList enthalten die die Tickets aller Besucher*innen verwaltet. Erstellen Sie das Attribut und initialisieren Sie diese im Konstruktor. **(2 Punkte)**

Implementieren Sie die Methode ***addTicket***: Fügen Sie ein Ticket nur hinzu wenn es noch nicht in ArrayList hinzugefügt wurde und zusätzlich auch ein gültiges Ticket ist. (Siehe Anmerkung) Die Methode soll **true** zurück liefern falls erfolgreich, sonst **false**. **(2 Punkte)**

Hinweis: Die Gültigkeit des Tickets kann mit der Klasse **TicketFraudSystem** überprüft werden, indem die statische Methode *boolean checkTicket(NFCTicket ticket)* das mitgegebene Ticket überprüft. Die Methode liefert bei Gültigkeit **true** zurück und sonst **false**. (Sie müssen den Code der Klasse nicht nachvollziehen)

Die restlichen Methoden in **ConcertSystem** können Sie in beliebiger Reihenfolge implementieren:

double profitPerCategory(int category): Basierend auf der im Parameter spezifizierten Kategorie berechnen Sie den durchschnittlichen Profit aller Tickets mit dieser Kategorie und liefern Sie das Ergebnis als Rückgabewert zurück. Beispiel: Falls der Parameter 1 wäre dann den durchschnittlichen Profit aller Tickets der Kategorie 1, bei Parameter 2 ….**(5 Punkte)**

ArrayList<String> invitedToStage(): Soll eine Liste von Namen in einer ArrayList zurückliefern die gemeinsam auf der Bühne mit Hansi dürfen. Folgende Personen sollen ausgewählt werden: **(6 Punkte)**

- Alle Personen mit einem MoonBoot Ticket – diese lassen sich daran erkennen, dass der erste Buchstabe Ihres Sicherheitscodes mit einem **"M"** beginnt. (Den Sicherheitscode erhalten Sie mit getSecret())
- Alle Personen mit AppTicket (erkennbar an einem **"A"** am Anfang Ihres Sicherheitscodes) die ein Ticket für die Kategorie 1 besitzen.

Hinweis: Ein Stringobjekt besitzt die Methode *startsWith(„…")* mit der überprüft werden kann ob ein String genau mit einer gewissen Zeichenkette beginnt.

(knifflig) HashMap<Integer, Double> donationPerCategory(): Hansi hat ein Herz für Tiere und deshalb wird ein Teil des Profits für den Schutz gefährdeter Häschen gespendet. Es wird

- 30 % des Profits von Tickets der Kategorie 1,
- 20 % des Profits von Tickets der Kategorie 2 und
- 10 % des Profits von Tickets der Kategorie 3 gespendet.

Die Methode soll eine HashMap zurückliefern die für jede Ticketkategorie die Summe der Spenden speichert. Das heißt der Schlüssel soll die Kategorie sein und der Wert die Summe der Spenden der entsprechenden Kategorie. Beispiel: {1: 100, 2:200, 3:1000} In diesem

Beispiel wären die Spenden von Kategorie 1 Tickets in Summe 100, von Kategorie 2 Tickets 200 und von Kategorie 3 Tickets 1000. **(6 Punkte)**

public void sortAndPrintForAccounting(): Sortieren Sie die ArrayList mit dem Comparator *AccountingComparator*. Für die Buchhaltung müssen die Tickets anhand der beiden Kriterien Kategorie und Sicherheitscode sortiert werden: zuerst nach der Kategorie des Tickets (niedrigere Kategorie zuerst) und falls Kategorien gleich sind dann nach Ihrem Sicherheitscode (alphabetische Reihenfolge)

Geben Sie danach in der Methode für jedes Ticket Kategorie und Sicherheitscode aus. Ihnen steht frei wie genau, solange beide Werte separat nebeneinander oder untereinander ausgegeben werden (und damit die Sortierung nachvollziehbar ist). **(6 Punkte)**

Testen Sie Ihre Implementation in der Klasse Main: **(6 Punkte)**

- Erstellen Sie 2 Paper Tickets, 2 App Tickets und 2 MoonBoot Tickets
- Testen Sie für diese 6 Tickets getProfit bzw. getCategory Funktionalität (zumindest aufrufen und ausgeben)
- Erstellen Sie eine Instanz von ConcertSystem und fügen Sie alle 6 Tickets hinzu
- Testen Sie die vier ConcertSystem Methoden

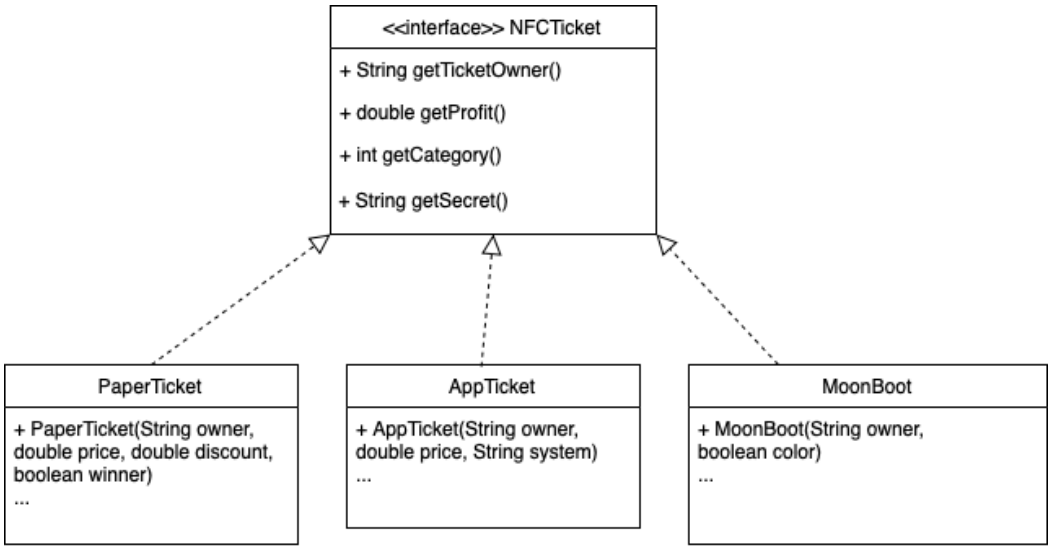
********* ENDE DEUTSCHE VERSION - START ENGLISH VERSION *********

The grading is based on the german version – the english translation is just for your assistance. You are allowed to use all your documents, old programs and internet sources. Communication, exchange in any way (analog or digital) or other tools are not allowed. At the beginning you will receive your starting package. At the end of the time, please hand in your project directory as a ZIP file via Moodle. You will only receive points for compilable and functional code!

Stick to the specifications: No change of given method names or structure, text output instead of return value, ...

Hansi Hinterseer concert

For the current anniversary concert tour of Hansi Hinterseer you are supposed to create a management system for one of his concerts. To prevent the trade with fake tickets every kind of ticket needs a NFC radio chip.



There are three different ticket types, all of which must implement the security methods of the **NFCTicket interface to** prevent unauthorized trading on the black market. Create an interface with the methods specified above in the diagram. Then create the constructors: **(11 points)**

PaperTicket: The class PaperTicket represents a classic paper ticket (with integrated NFC security chip) and the constructor is initialized with the following parameters:

- **owner:** Name ticket buyer
- **price:** ticket price (double)
- **discount:** Discount in percent (double 20.5 is interpreted as 20.5 %)
- **winner:** paper tickets take part in a lottery and parameter indicates whether ticket has won -> true, otherwise false

Store all four properties in the class on protected attributes (choose comprehensible names). In addition, you must generate a **security code** in the constructor (see note) and store it on another protected attribute of type **string in** the class.

Note: To generate security codes, we have purchased the **TicketFraudSystem** class from Bunny Security Systems. It provides a static method *String createNewTicket(String type)* that gives us this security code as a return value. (You don't need to understand the code in the method).
For a PaperTicket, pass the string ***"paper"*** in the parameter in the previously mentioned method.

AppTicket: The class AppTicket stands for a ticket that was purchased mobile on a modern smartphone and is initialized with the following properties:

- **owner:** Name ticket buyer
- **price:** ticket price
- **system:** Specifies the name of the smartphone operating system.

Store these properties in the class on self-created protected attributes. Additionally, you still need to generate a **security code** in the constructor and store it on a custom property (**string**) in the class. Again use *createNewTicket* with the parameter ***"app".***

MoonBoot: Hansi's trademark are his MoonBoots (=winter shoes). For especially loyal fans, there is a small MoonBoot souvenir that on the one hand is a concert ticket and on the other hand can also be used as a mood light at home or during the concert. Initialize with the following features:

- **owner:** Name ticket buyer
- **color:** If **true** then it is the model with lighting in different colors, if **false** it can light only in one color

Store these two properties in the class on protected attributes. In addition, you still need to generate a **security code in** the constructor and store it on a protected attribute (**string**) in the class. Again use *createNewTicket* with the parameter ***"moonboot".***

Implement the abstract methods in the three classes:
getTicketOwner should return owner*in as return value. **(1 point)**

getSecret is to return the security code of the ticket. **(1 point)**

getProfit should return the profit made with the sale of the ticket: **(7 points)**

- **PaperTicket:** From the ticket price, 20 euros of merchant costs must be deducted first and then the discount must be deducted to get the profit.
- **AppTicket:** If the operating system is ***"ios"*** then the profit is 70% of the ticket price. For "**android**" 75% and for all other operating systems 80% of the ticket price.
- **MoonBoot:** The MoonBoot models are sold only directly - for a model with colored light there is 250 euros profit, for a model with one color 200 euros.

getCategory shall return the ticket category of the ticket: **(7 points)**

- **PaperTicket:** Tickets with a price of at least 80 euros allow access to seats of category 2 - otherwise category 3. If you have won the lottery when buying the ticket, the category is further reduced by one category.
- **AppTicket:** Tickets with prices of at least 100 euros allow access to category 1 or, for tickets from 50 and under 100 euros, category 2 or otherwise category 3.
- **MoonBoot:** MoonBoot with colored lighting allow access to category 1, with one color to category 2.

The ConcertSystem class should contain an ArrayList that manages the tickets of all visitors. Create the attribute and initialize it in the constructor. **(2 points)**

Implement the ***addTicket*** method: add a ticket only if it has not yet been added to ArrayList and is also a valid ticket. (See note) The method should return **true** if successful, **false** otherwise. **(2 points)**

Note: The validity of the ticket can be checked with the **TicketFraudSystem** class by using the static method *boolean checkTicket(NFCTicket ticket) to* check the ticket provided. The method returns **true** if valid and **false** otherwise. (You do not need to understand the code of the class)

You can implement the remaining methods in **ConcertSystem in** any order:

double profitPerCategory(int category): Based on the category specified in the parameter, calculate the average profit of all tickets with this category and return the result as a return

value. Example: If the parameter would be 1 then the average profit of all tickets of category 1, for parameter 2 … **(5 points)**

ArrayList<String> invitedToStage(): Shall return a list of names in an ArrayList that are allowed on stage together with Hansi. The following people shall be selected: **(6 points)**

- All persons with a MoonBoot ticket - these can be recognized by the fact that the first letter of the security code starts with an **"M"**. (You can get the security code with getSecret())
- All persons with AppTicket (recognizable by an **"A"** at the beginning of your security code) who have a ticket for category 1.

Note: A String object has the method *startsWith("...")* which can be used to check if a string starts exactly with a certain string.

(tricky) HashMap<Integer, Double> donationPerCategory(): Hansi has a heart for animals and therefore a part of the profit is donated for the protection of endangered bunnies. He will donate:

- 30% of the profit of category 1 tickets,
- 20% of the profit from category 2 tickets and
- 10% of the profit from category 3 tickets.

The method should return a HashMap that stores the sum of donations for each ticket category. That is, the key should be the category and the value the sum of the donations of the corresponding category. Example: {1: 100, 2:200, 3:1000} In this example the donations of category 1 tickets would be in sum 100, of category 2 tickets 200 and of category 3 tickets 1000. **(6 points)**

public void sortAndPrintForAccounting(): Sort the ArrayList with the comparator *AccountingComparator*. For accounting purposes, tickets must be sorted based on the two criteria category and security code: first by the category of the ticket (lower category first) and if categories are the same then by the security code (alphabetical order)

After that, print the category and security code for each ticket in the method. You are free to specify how exactly, as long as both values are output next to each other or below each other (and thus the sorting is comprehensible). **(6 points)**

Test your implementation in the class Main: **(6 points)**

- Create 2 Paper Tickets, 2 App Tickets and 2 MoonBoot Tickets
- Test getProfit or getCategory functionality for these 6 tickets
- Create an instance of ConcertSystem and add all 6 tickets
- Test the four ConcertSystem methods