Programming 1

Aryobarzan Atashpendar (aryobarzan.atashpendar@uni.lu)

Lab 6 – Object-Oriented Programming

Hades II © Supergiant Games

Exercise 1 – Rectangle

Design a class **Rectangle** as follows:

1° Two properties *width* and *height* of type double.

2° A **constructor** to initialize the properties.

3° Getter methods for *width* and *height*.

4° A method **getArea()** which returns the area of the rectangle.

5° A method **getDiagonal()** which returns the diagonal of the rectangle based on the following formula: $d = \sqrt{width^2 + height^2}$

Test your **Rectangle** class in a separate **Launcher.java** file.

Example output for a 5x10 rectangle:

```
Width: 5
Height: 10
Area: 50
Diagonal: 11.180339887498949
```

Exercise 2 – Sating Zeus

The god of gods Zeus demands offerings from humans from time to time. To that end, humans prepare offerings by queuing up to hand them personally to Zeus.

Design a class **OfferingsQueue** as follows:

1° A property *offerings* which will hold the names (String) of the offerings.

2° A property *count* of type int keeping track of the number of offerings currently in queue. This count always starts at 0!

3° A **no-arg constructor** that creates an OfferingsQueue of size 10.

4° A **constructor** that creates an OfferingsQueue of the specified *size*

5° A method **printOfferings()** which prints the names of the offerings currently in the queue.

6° A method **add(name)** which adds the offering's name to the end of the queue. If there is no space left in the queue, a warning should be printed.

7° A method **offer()** which offers the first item in the queue to Zeus. Zeus will randomly (50/50) be either satisfied or dissatisfied by the offering.

- If he is satisfied, the offering is removed from the queue.
- If he is dissatisfied, the offering is moved to the end of the queue.
- If there are no offerings left, a warning should be printed.

8° A method **clear()** which empties the offerings queue.

Ensure your implementation reuses common code as much as possible (redundancy!).

Test your **OfferingsQueue** class in a separate **Launcher.java** file.

Sample main method:

```
public static void main(String[] args) {
        OfferingsQueue offeringsQueue = new OfferingsQueue(5);
        offeringsQueue.add("Bread");
        offeringsQueue.add("Urn of Gold");
        offeringsQueue.add("Goat");
        offeringsQueue.add("Laurel");
        offeringsQueue.add("Statue");
        offeringsQueue.add("Bag of Coins");

        offeringsQueue.offer();
        offeringsQueue.offer();
        offeringsQueue.offer();
        offeringsQueue.offer();
        offeringsQueue.offer();
        offeringsQueue.offer();
    }
```

Sample output:

```
Added offering: Bread
Added offering: Urn of Gold
Added offering: Goat
Added offering: Laurel
Added offering: Statue
No more offerings can be added at the moment. (Bag of Coins)
Zeus despises your offering, throwing it to the back: Bread
```

```
OFFERINGS:
        1: Urn of Gold
        2: Goat
        3: Laurel
        4: Statue
        5: Bread
Zeus is satisfied with your offering: Urn of Gold
OFFERINGS:
        1: Goat
        2: Laurel
        3: Statue
        4: Bread
Zeus is satisfied with your offering: Goat
OFFERINGS:
        1: Laurel
        2: Statue
        3: Bread
Zeus is satisfied with your offering: Laurel
OFFERINGS:
        1: Statue
        2: Bread
Zeus despises your offering, throwing it to the back: Statue
OFFERINGS:
        1: Bread
        2: Statue
Zeus despises your offering, throwing it to the back: Bread
OFFERINGS:
        1: Statue
        2: Bread
```

Exercise 3 – Solar Energy

To reduce pollution, ordinary citizens have started installing solar panels on their houses, allowing them to store energy, use it to power their home or to transfer energy to a neighbor!

Design a class **House** as follows:

1° It has a property *postalCode*. Choose an appropriate type!

2° It has a property *street*. Choose an appropriate type!

3° It has a property *solarInstallation* of type **SolarInstallation**.

4° It has getter methods for its properties.

5° It has a **constructor** to initialize its *postalCode* and *street*.

6° It has a method **transfer(house, amount)** allowing it to transfer a specific *amount* of energy stored in its *solarInstallation* to another **House**'s *solarInstallation*. The current date should also be printed to the console when announcing whether the transfer was successful or not.

Design a class **SolarInstallation** as follows:

1° It has a property *energy* of type double, which is initially 0.

2° It has a getter method for its property.

3° It has a method **store(amount)** allowing it to increase its stored *energy* by a specific *amount*. If the indicated *amount* is negative, a warning should be shown instead.

4° It has a method **use(amount)** allowing it to use a specific *amount* of its stored *energy*. If the indicated *amount* is negative, a warning should be shown instead. If there is not sufficient *energy* to use, a warning should be shown instead.

Test your classes in a separate **Launcher.java** file by trying to store, use and transfer energy!

Sample output:

```
Stored 500.0 energy. (Total: 500.0)
Stored 150.0 energy. (Total: 150.0)
100.0 energy used. (Remaining: 400.0)
400.0 energy used. (Remaining: 0.0)
Stored 400.0 energy. (Total: 550.0)
400.0 energy transferred from Place d'armes (1) to Huffington Street (2000) on Thu Oct 19 12:50:09 CEST 2023.
Insufficient energy: 0.0 (required: 100.0)
```

Exercise 4 – Time and date

For this exercise, you want to represent specific dates and times, while also being able to manipulate them.

1° Create a class **Date** with attributes *day*, *month* and *year*.

- Add a method **isLeapYear()** that returns whether the year is a leap year or not.
- Add a method **daysInMonth()** that returns the number of days in the current month.
- The **constructor** of the class sets the initial values of the attributes based on parameters, which are checked upon their validity and adapted if necessary.
- Add a method **advance()** which advances the current date by a day.
- Add a method **format(boolean us, String delimiter)** which returns the formatted date as a String. Use leading zeroes for days or months below 10. The *delimiter* string delimits the 3 parts. If *us* is true, the month precedes the day.

2° Create a class **Time** with attributes *hours*, *minutes* and *seconds*.

- The **constructor** of the class sets the initial values of the attributes based on parameters, which are checked upon their validity and adapted if necessary.
- Add a method **tick()** which advances the current time by a second. The method returns a boolean value indicating whether a new day has begun. Note that hours here are always stored as values between 0 and 23.
- Add a method **format(boolean us)** which returns the formatted time as a String. Use leading zeroes for hours, minutes or seconds below 10. The format is hh:mm:ss. If *us* is true, hours are formatted as values between 1 and 12 and a suffix (AM/PM) is added at the end.
- Add the methods **secondsSinceMidnight()** and **secondsUntilMidnight()**, whose returned values should be self-explanatory based on the method names.

3° Create a class **DateTime** which has an attribute of type **Date** and another of type **Time**.

- Add a method **tick()** which advances the time by one second and advances, if necessary, the date by one day.
- Add a method **print(boolean us, String delimiter)** which uses the **format()** methods of **Date** and **Time** to print the current date and time to the console.

Test your classes in a separate **Launcher.java** file.

Exercise 5 – Giveaways! (Advanced)

Terri Aki owns 3 shops in Luxembourg. To celebrate the business' 50-year anniversary, the owner decides to award their customers with giveaways based on the price of the items they purchase. As the shops are spread out all over Luxembourg, Terri Aki wants to ensure that customers from every region have a chance to win a giveaway. Write a program that helps Terri Aki to manage the giveaways in the different shops.

1° Design a class **Item** with the attribute *price* and a **constructor** which sets the initial value of the price. Also write a getter method for this attribute.

2° Create a class **Shop** with the attribute *localNumberOfGiveaways* and the class attribute *maxNumberOfGiveaways*.

- Add a **constructor** which sets the initial value of *localNumberOfGiveaways*.
- Add a method **buy(Item item)** which:
  - prints the *price* of the purchased **Item**.
  - prints an appropriate message to the console if the given shop has no giveaways left or if there are no more giveaways left nationally.
  - randomly awards a giveaway based on the *price* of the **Item** purchased if there are still giveaways left at the shop. There is a 2% chance to win a giveaway for items cheaper than 20€, a 5% chance for items between 20€ and 100€, and a 10% chance for items which cost more than 100€. It prints a message to the console detailing whether the customer has won a giveaway or not. Don't forget to update the number of remaining giveaways!

Test your implementation in a separate **Launcher.java** file by reading the total number of giveaways from the console, then creating 3 shops among which the total number of giveaways is evenly distributed. Afterwards, simulate customers buying items of randomly varying prices from the 3 shops, until all giveaways have been won and there are none left nationally.

Class attribute

If you don't know about class attributes/variables, see this helpful article with an example.

Final Fantasy Crystal Chronicles: Echoes of Time © Square Enix