Butterfly Business

“

Implement a program that facilitates upgrading the old butterfly collecting family business to the new technology era, so that they can transfer the old accountancy documents into an application.

”

Background

An old family museum that shows butterfly collections wants to upgrade their payment system. Until now they have kept their payment records in paper. They decided to scan every document they have using an unusual app before they piled them up temporarily inside another room. The documents produced by this app display every number like this:

Unfortunately, there was a fire in the other office that very night, and the fire sprinklers triggered, soaking every single paper. The only chance to recover the payment slips is by reading the strange digital documents generated by that app.

Every document contains several payment operations. Each one of those payment operations contains the following information:

- The date of the payment
- The credit card number used
- The amount of the purchase

Like this:

Rules

To spice things up a little bit, here are some constraints that will make the challenge more interesting:

- You’re not allowed to use loops
- When testing the validity and the issuers of credit cards, the assertEquals assertion must be used to test all cases, but it is allowed to appear only once within your test class

USER STORY 1.

We want to be able to parse all payment operations from the *butterfly-business.txt* document into a purchase object that has the following attributes:

- The date of the payment
- The credit card number
- The amount paid

USER STORY 2.

We need to ensure that the credit card numbers written in the document have no typos. To ensure it, we will apply a checksum validation on them. The Luhn's algorithm is explained in detail in this link. Create tests to ensure the proper behaviour of the algorithm.

USER STORY 3.

We want to keep track of which kind of credit card was used to pay. The credit card stored in the purchase should know the credit card issuer. If it didn't pass the Luhn validation then the issuer will be *invalid*.
To start, we can begin being able to identify Visa and MasterCard by implementing their corresponding rules. If it passes the Luhn validation but it's not recognized as either of the available issuers, then the issuer will be *unknown*.

USER STORY 4.

Add to the available credit card issuers the following ones:

- Visa Electron
- American Express
- Discover
- Maestro
- InstaPayment

You're only allowed to modify one class when adding these new credit card types. If you're not able to do that, then refactor your application until it is.

USER STORY 5.

We want to store payments information in a csv document so the accountant can read them easily. Payments should be displayed sorted by date (earliest first). The columns of the file will show the date, the credit card number, the credit card issuer and the amount paid with it. The first line of the document will contain the headers.

USER STORY 6.

We want to take advantage of technology by being able to ask business intelligence questions. From the payments information obtained, we want to be able to answer the following questions:

- How much money was obtained in total?
- Which month was the most profitable?
- What is the average paid amount per month?
- Which was the credit card issuer that was used the most?

Write this information in another file.

♜ Portions of Integers ♜

“

Implement a program that receives an integer number and it calculates its integer portions.

”

Background

A portion of a positive integer *n*, or an integer portion, is a way of writing *n* as a sum of positive integers. Two sums that differ only in the order of their summands are considered the same portion.

For example, the integer number four can be portioned in five distinct ways:

portions(4) -> [4], [3,1], [2,2], [2,1,1], [1,1,1,1]

And the integer number five can be portioned in seven distinct ways:

portions(5) -> [5], [4,1], [3,2], [3,1,1], [2,2,1], [2,1,1,1], [1,1,1,1,1]

Challenge

Develop the code capable of returning the previous lists of portions for a given integer number. The code should be able to find the number of integer portions of *n* for *n* as least as large as 50.

Rules

To spice things up a little bit, here are some constraints that will make the challenge more interesting:

- You’re not allowed to use loops
- The *assertEquals* assertion must be used to test the first 10 cases, but it is allowed to appear only once within your test class

USER STORY 1.

As a keen matematician I want to use a class with the following method so that I can obtain the integer portions when I provide the starting integer number.

```
List<List<Integer>> portions(int number)
```

USER STORY 2.

As an insecure matematician I want to ensure with unit tests that the output of the *n* numbers from 1 to 10 is calculated correctly.

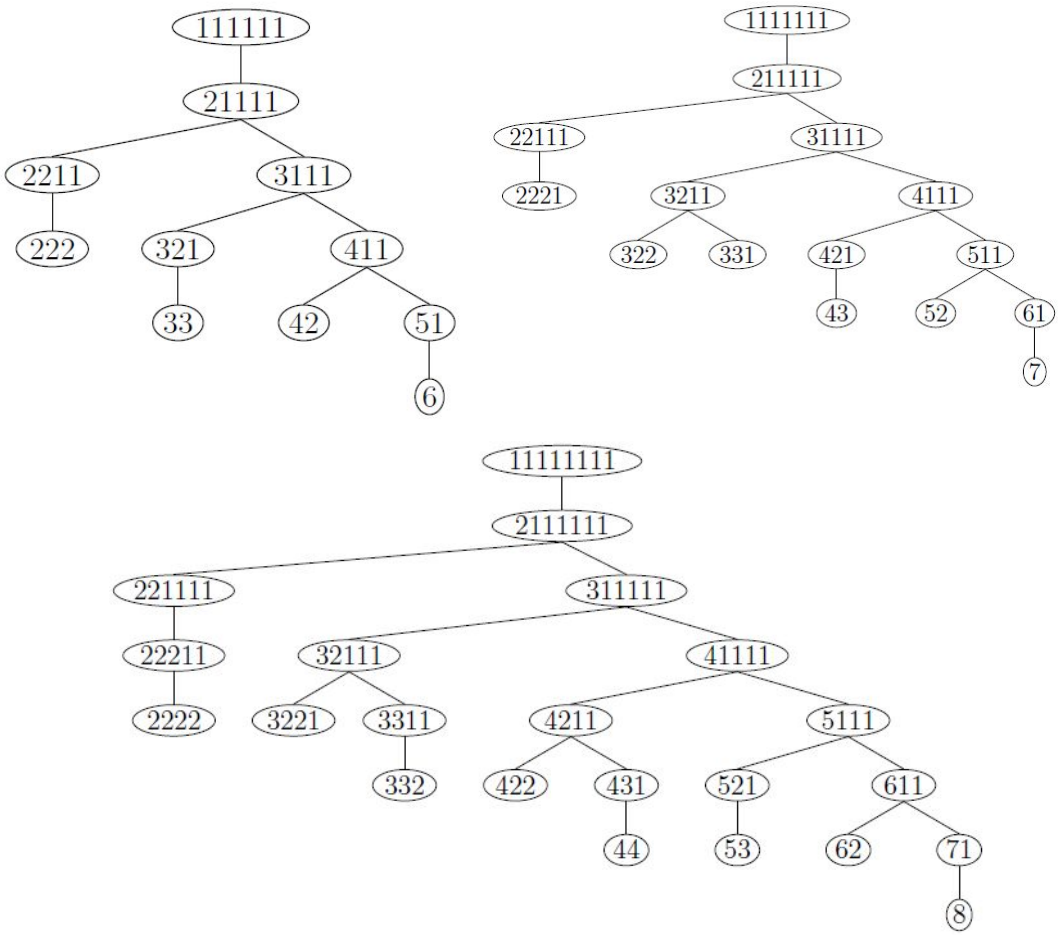
You can check the results of the calculations from 1 to 10 in the *portions-of-integers-test.txt* file.

USER STORY 3.

As a visual matematician I want to see the calculated portions written in a file.

You can check the expected result of the portions of 10 in the *portions-of-integers-10.txt* file.

Examples



♜ Sequence Resemblance ♜

“

Implement a program that calculates total resemblance value of a sequence.

”

Background

For two sequences of characters A and B, we define the resemblance of the sequences to be the length of the longest prefix common to both sequences. For example, the resemblance of the sequences *abc* and *abd* is 2, while the resemblance of the sequences *aaa* and *aaab* is 3.

Challenge

Develop the code capable of returning the sum of resemblances of a provided sequence with each of its suffixes. Ensure a right behaviour with unit tests.

Rules

To spice things up a little bit, here are some constraints that will make the challenge more interesting:

- You’re not allowed to use loops
- You’re not allowed to use more that one *;* (semicolon) per method
- The first instruction of each method must be the return keyword

Examples

`Input: ababaa`	»	`Output: 11`
`Input: aa`	»	`Output: 3`

For the first case, the suffixes of the sequence are *ababaa*, *babaa*, *abaa*, *baa*, *aa* and *a*. The resemblances of these suffixes with the sequence *ababaa* are 6,0,3,0,1, & 1 respectively. Thus, the answer is 6 + 0 + 3 + 0 + 1 + 1 = 11.

For the second case, the answer is 2 + 1 = 3.

♜ Werewolf Game ♜

“

Implement the Werewolf game as a one player console game.

”

Background

Werewolf is a group game where each person receives a card representing a secret role. Some of them will be werewolves, some of them will be special roles and the rest will be villagers. The goal of the game for the villagers is to figure out who the werewolves are, and for the werewolves is to kill enough villagers so their survival is ensured.

You can read everything about the game here.

Challenge

Develop a one player game where you can play the werewolf game against the computer.

Rules

To spice things up a little bit, here are some constraints that will make the challenge more interesting:

- You’re not allowed to use loops
- The main game does not know if players are humans or machines
- The main game does not know who has which role
- The main game does not know who has which personality
- You’re not allowed to modify more than one class in order to add more roles or personalities

USER STORY 1.

We want to be able to set up the beginning of a game. These are the requirements:

- The human player chooses how big the group of players will be
- The roles are distributed randomly following the rules of the game
- The human player is told which role has been assigned to him/her

USER STORY 2.

We want to be able to play a basic game. These are the requirements:

- The sequence of day and night stages are displayed following the rules of the game
- Everything that happens during the game is explained by the moderator
- During the night, every particular role wakes up when necessary to perform their actions
- During the day, every player votes for the person who they suspect is a werewolf, except werewolves who will vote for others. The voting is repeated until at least one player receives the majority of votes. Then this player is killed and is out of the game.
- If the human player gets killed, the game continues as usual until villagers or werewolves win.

USER STORY 3.

We want to be able to give hints to the players when they are voting during the day to determine who is the werewolf. In order to accomplish this, we want to give every player a personality. There are three types of personalities: insecure, regular and self-confident. Depending on the personality that each one has, they might show more or less insecurity when saying that they are not the werewolves. Having insecurities increases the chances that somebody will vote against you. Being very self-confident reduces the chances that somebody will vote that you are a werewolf. The human player character is also affected by this personality and will influence how the rest of the players will vote for him/her or not.

The ratio of personalities should be like this:

- 25% of the players will have an insecure personality
- 50% of the players will have a regular personality
- 25% of the players will have a self-confident personality

During the day, and from the human player perspective, right before voting, a screen will be displayed where each character presents a sentence explaining why they are not the werewolf. Depending on whether they have an insecure, regular or self-confident personality, the messages will sound more convincing. It's up to your team to decide which messages are displayed. But those that belong to the insecure personality should give hints of doubt, those with a regular personality just explain the case and those with self-confident personality present clearly and convincingly why they are not the werewolves.

Other non-human players are also affected by the personalities of others when voting, so it would be more likely that insecure players will be voted more often and self-confident players less often.

Space Invaders

> *Implement the space invaders game as a one player game.*

Background

Space Invaders is an arcade game created by Tomohiro Nishikado and released in 1978. It's one of the earliest shoot 'em ups and the first fixed shooter. The aim is to defeat waves of aliens with a laser to earn as many points as possible.

You can read everything about the game here.

Challenge

Develop a one player space invaders game with JavaFX.

Rules

To spice things up a little bit, here are some constraints that will make the challenge more interesting:

- You're not allowed to use loops

USER STORY 1.

We want to display the player's spaceship on the screen. This one has to be able to move around the screen with the movement arrows (up, down, left and right). This means that a diagonal move is also allowed.

USER STORY 2.

We want to shoot a laser beam with the spaceship when we press the spacebar. The laser beam will move forward at a constant speed until it disappears passed the top of the screen.

USER STORY 3.

We want to add enemy invader ships that shoot back at the player. Only enemies that have no other enemy in front are allowed to shoot. These enemies move left and right following the rules of the original game, and eventually they move forward as well. In the meantime, they also shoot laser beams back at the player. A laser beam destroys a ship: both the player's ship or enemy's ship.

USER STORY 4.

We want the player to have more than one life. Every laser beam that touches the player's ship reduces the amount of lives by one. When the total life-count reaches zero, then the game is over. We want to display the amount of lives left on the screen as well.

USER STORY 5.

We want to load the enemies from a file, so that it's possible to have more than one level. Killing each enemy ship gives you points and finishing a level also gives you extra points. We want to display those points on the screen.

USER STORY 6.

We want to add obstacles in each level so that it's harder to aim at ships from both sides. When a laser beam reaches an obstacle, it disappears and removes one block of the obstacle.

USER STORY 7.

We want to add packets that appear at random moments from the top of the screen and fall down until they disappear through the bottom of the screen. If the player's ship touches them, then they disappear and something special happens. There should be a way to visually identify which packet is which with a proper display. There are three types of packets: +1 life, +points, -1 life.

USER STORY 8.

We want to have a menu that appears when we start the game and whenever we press the *escape* key (pausing the game). If there is no ongoing game, the menu shows the options: play and exit. If there is an ongoing game, then it shows the options: continue and exit.

USER STORY 9+ (optional)

We want to add further features like:

- Explosions
- A slow-motion packet where your spaceship moves at normal pace but everything else (enemies, laser beams and packets) move super slow for a limited amount of time
- Background music or sounds
- Background images per level
- Personal touch: anything else that you might find interesting to add

♜ Pacman ♜

> *Implement the pacman game as a one player game.*

Background

Pac-Man, stylized in all capitals, is an arcade game designed by Toru Iwatani of Namco and first released in Japan as Puck Man in May 1980. It was released the United States in October 1980 when top arcade games were stark space shooters, such as Galaxian and Asteroids. Pac-Man established the conventions of the maze chase genre, and is considered one of the classics of the medium and an icon of 1980s popular culture.

You can read everything about the game here.

Challenge

Develop a one player Pac-Man game with JavaFX.

Rules

To spice things up a little bit, here are some constraints that will make the challenge more interesting:

- You're not allowed to use loops
- The game is not allowed to know with which ghost it's interacting with

---

USER STORY 1.

We want to load a Pac-Man level from a file. This file will explain where the walls, the ghosts, their respawn area and the Pac-Man are. We want to display them on the screen.

USER STORY 2.

We want to tell the Pac-Man in which direction it has to continue moving by using the keys: up, down, left and right. Pac-Man will not walk through walls and will follow the original rules of the game for movement.

USER STORY 3.

We want the player to start the game with three lives. We want them to be displayed on the screen.

USER STORY 4.

We want to place small dots called Pac-Dots all around the corridors so that the Pac-Man can eat them to collect points. We want the points to be displayed on the screen. The player will receive one extra life after obtaining 10,000 points.

USER STORY 5.

We want to add the four multicolored ghosts Blinky, Pinky, Inky and Clyde. They move in random directions. If they touch Pac-Man, this one loses a life. We also want to add near the corners of the maze, the four flashing Power Pellets that provide Pac-Man with the temporary ability to eat the ghosts and earn bonus points. These Power Pellets swap the state of the ghosts so that they run away from Pac-Man at a slower speed and they die if they are eaten by it. When a ghost is eaten its eyes go to the respawn area and wait there until they respawn. After a while, the ghosts that haven't been eaten return to their original state.

USER STORY 6.

We want that each ghost has its own personality and strategy regarding movement. Their behaviour should be as follows:

- The red enemy chases Pac-Man
- The pink enemy aims for a position in front of Pac-Man's mouth.
- The blue enemy is "fickle" and sometimes heads towards Pac-Man, and other times away.
- The orange enemy's behavior is random. It alternates from behaving like the red enemy when at some distance from Pac-Man and aiming towards the lower-left corner of the maze whenever it gets too close to him.

USER STORY 7.

We want to play more than one level, so that when every Pac-Dot from a level is eaten, then the game progresses to the next level. These levels will be loaded from files as well.

USER STORY 8.

We want to have a menu that appears when we start the game and whenever we press the *escape* key (pausing the game). If there is no ongoing game, the menu shows the options: play and exit. If there is an ongoing game, then it shows the options: continue and exit.

USER STORY 9+ (optional)

We want to add further features like:

- Explosions
- A slow-motion packet where your spaceship moves at normal pace but everything else (enemies, laser beams and packets) move super slow for a limited amount of time
- Background music or sounds
- Background images per level
- Personal touch: anything else that you might find interesting to add