# Exam Preparation – 11 October 2024

## **Pie Pursuit**

**Link:** [**https://alpha.judge.softuni.org/contests/java-advanced-retake-exam-10-april-2024/4547/practice#0**](https://alpha.judge.softuni.org/contests/java-advanced-retake-exam-10-april-2024/4547/practice#0)

*Get ready for the "Pie Pursuit" – a delightful journey where contestants indulge in a pie-eating frenzy. It will be a fun competition.* *Join us in this exciting adventure as contestants showcase their love for delicious treats and strive to emerge as the ultimate pie-eating champions!*

You will receive **two sequences on two separate lines**. Each sequence will consist of **integer values**,   
**separated by a single whitespace**.

The **first sequence** represents **contestants**, each value indicating the **total count of pieces of pie a contestant can consume**, while the **second sequence** represents the **pies**, signifying the **number of pieces each pie contains**.

### **Contest Rules**

* The competition continues **until no more pies** are left **or no more contestants** remain in the battle
* The contest kicks off with the **first contestant** facing the **last pie** in the sequence
* If the **contestant's pie-eating capacity** is **greater than or equal to the pie's size**, they manage to eat the entire pie, **subtracting the pie's value** (count of pie pieces)from **the contestant's value**(pie-eating capacity). The **pie is removed** from the collection.
  + If the contestant's **pie-eating** **capacity reaches 0**, bid them farewell from the contest (**remove the contestant** from the competition).
  + Otherwise, **move the satisfied contestant to the back of the lineup, with the remaining   
    pie-eating capacity**.
* If the **pie's size exceeds the contestant's pie-eating capacity**, the contestant consumes **as many pieces as possible, subtracting the eaten pieces** (contestant's capacity) from the pie's size.
  + The **pie remains in the sequence** with the value of pieces left.
  + If the **pie's size reaches 1 piece**, **remove the pie**, **adding the remaining piece to the next pie** in the sequence.
  + If this is the **last pie**, be careful, **you won’t be able to add a piece to the next pie. Just leave that last piece back in the sequence.**
  + Say goodbye to the contented contestant from the contest (**remove the contestant** from the competition).
* In the end:
  + If the **pies are over**, and **there are contestants left**, print on the console:   
    **"We will have to wait for more pies to be baked!**" and **the final state of the contestants' sequence on a separate line**.
  + If **both the pies and contestants are over**, it means that we have a champion. Print on the console: **"We have a champion!"**
  + If the **contestants are over**, but **there are pies left**, print on the console:   
    **"Our contestants need to rest!"** and **the final state of the pie sequence on a separate line**.

Let the Pie Pursuit unfold and discover who will be the ultimate champion!

### **Input / Constraints**

* On the **first line**, you will receive the integers, representing the **contestant's pie-eating capacity**, **separated** by a **single space**.
* On the **second line,** you will receive the integers, representing the **number of pieces each pie contains**, **separated** by a **single space**.

### **Output**

The following result should be **printed on the Console**, on **separate lines**:

* If the **pies are over**, and **there are contestants left**:
  + **"We will have to wait for more pies to be baked!**"
  + **The final state of the contestants' sequence:**

"**Contestants left: {contestant1}, {contestant2}, … {contestantn}**"

* If **both the pies and contestants are over**:
  + **"We have a champion!"**
* If the **contestants are over**, but **there are pies left**:
  + **"Our contestants need to rest!"**
  + **The final state of the pie sequence**
  + "**Pies left: {pie1}, {pie2}, … {pien}**"

### **Examples**

|  |  |  |
| --- | --- | --- |
| Input | Output | Comment |
| 5 8 4 6  3 7 2 9 | We will have to wait for more pies to be baked!  Contestants left: 2 | We take the first contestant (5) and the last pie (9). The contestant can eat only 5 pieces of pie, and there will be 4 pieces left. We remove the contestant and keep the 4 pieces in the sequence.  Now we have the following sequences:  8 4 6  3 7 2 4  The next contestant can eat all four pieces, and go at the back of the sequence, with the value of 4. We remove the pie, and we have the following sequences:  4 6 4  3 7 2  4 eats 2, and goes at the back with a value of 2:  6 4 2  3 7  The next contestant eats 6 pieces, and one is left to be added to the next pie:  4 2  4  The next contestant eats all 4 pieces and is unable to eat more. As reaching the value of zero, he quits the competition. The only contestant left is 2. |
| 4 6 8 10 12 16  16 12 10 8 6 4 | We have a champion! |  |
| 3 3 3 3 3  4 4 4 4 | Our contestants need to rest!  Pies left: 1 |  |
| 2 2 2  3 3 3 4 | Our contestants need to rest!  Pies left: 3, 4 |  |

# 02. Beesy

**Link:** [**https://alpha.judge.softuni.org/contests/java-advanced-regular-exam-22-june-2024/5009/practice#1**](https://alpha.judge.softuni.org/contests/java-advanced-regular-exam-22-june-2024/5009/practice#1)

*Bees are indispensable for maintaining the health of ecosystems, promoting biodiversity, supporting agriculture, and ensuring food availability for human populations worldwide. Protecting bee populations and their habitats is crucial for the well-being of both ecosystems and humanity.*

You will be given an integer **n** for the **size** of the **field** with a **square** shape. On the next **n** lines, you will receive the **rows** of the **field**.

* Your bee will be placed in a **random position**, marked with the letter '**B**'.
* There will be flowers with nectar which need to be pollinated on **random positions**, marked with a **single digit**.
* There will be a **hive**, marked with the letter '**H**'.
* All of the empty **positions** will be marked with **'-'**.

The bee will have **15 units of energy initially**.

A command is received each turn **until the bee runs out of energy** or **reaches the hive**.

the bee must collect and take **at least** **30 units** **of nectar to the hive**. This would be the required quantity to make honey successfully.

Keep in mind that even if the **needed amount of nectar is collected, but the hive is not reached** the **bee continues to move** according to the commands.

You will be given **commands** for **the bee's movement**. The commands will be: "**up**", "**down**", "**left**", and "**right**". The bee moves towards the given direction.With **each** move, the **bee's energy** **decreases** by **1 unit.**

* If the **bee leaves the field** (goes out of the boundaries of the matrix) depending on the move command,  
   **it will be moved to the opposite side of the field.**   
  **Example:** In a 3x3 matrix the bee is at position **[1,2]** and receives the command "**right**" it will be moved to position **[1,0]**.
* If the bee **moves** to a **flower (a position marked with a digit)**, it collects the nectar (**the value of the digit is** **added** to the **total amount of collected nectar**) the **flower disappears** and should be replaced by **'-'**.
* If the bee **runs out of energy**, **and** the **total amount of collected nectar** **is less than 30 units,** the **program ends**. [The correct output should be printed on the Console](#Output).
* If the bee **runs out of energy** **and** the **total amount of collected** **nectar is at least 30 units**, the energy will be restored with the **amount of the difference** between the nectar **collected up to this turn** and the **minimum required** amount for making honey *(30).* The **value of the collected nectar is dropped to 30 units**. The energy **can be restored only once**.

**Example:** Collected nectar is equal to **40 units.** 40 – 30 = **10**. The bee's energy is **increased** by 10, the nectar is decreased to 30 units.

**Hint: Check for zero energy after restoration.**

* + The **next time the bee runs out of energy**, the movement discontinues.The **program ends**.   
    [The correct output should be printed on the Console](#Output).
* If the bee **reaches a position, marked with**  '**H**', the hive is reached and the **program ends**.

**Hint: If reaching the hive with zero energy, the success will depend on the amount of the collected nectar.**  
[The correct output should be printed on the Console](#Output).

### **Input**

* On the first line, you are given the integer **n** – the size of the **square** **matrix**.
* The **next n lines** hold the values for **every matrix row**.
* After that, you will get a **move command on each of the next lines**.

### **Output**

* On the first line:
* If the bee **reaches the hive** with at least **30 units of nectar collected**, print this message and stop the program:
* **"Great job, Beesy! The hive is full. Energy left: {energy}"**
* If the bee **reaches the hive** but **has not succeeded in collecting at least 30 units** of nectar:
* **"Beesy did not manage to collect enough nectar."**
* If the **bee runs out of energy**, **before returning** to the hive:
* **"This is the end! Beesy ran out of energy."**
* On the **next** lines.
* Print the **final state of the** **matrix, with the last known position of the bee, marked with 'B'**.

### **Constraints**

* The size of the **square** matrix *(field)* will be between **[2…10].**
* Only the letters '**B**' and '**H**' will be present in the matrix.
* The **flowers with nectar** are represented by **single positive digits** between **[0…9]**.
* There will always be **enough** commands to finish the program.

### **Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  --B--  H-987  -4812  5----  2----  down  right  right  down  left  left  left  down  left  up  up  up | Great job, Beesy! The hive is full. Energy left: 4  -----  B----  -----  -----  2---- |
| **Comment** | |
| The bee starts from the **position [0,2]**. The first command is **"down"**. The bee moves to **position** **[1,2]** and gets **9** units of nectar. The initial **energy drops to 14 units**. The next command is **"right" -> position [1,3]** gets **8 units of nectar, and the energy drops to 13 units.** Then the command **"right"** again, **position ->** **[1,4]** and **nectar -> 17 + 7**, **energy -> 12.** The next commands are **"down", "left","left",** and **"left" ->** and the bee collects accordingly **2,1,8** and **4** units of nectar. Position -> [2,1], Nectar -> 24 + 2 + 1 + 8 + 4 = 39, Energy -> 8.  Then **"down"** and **"left"** commands follow and your bee adds another **5** units of nectar to the previous amount. Finally, we get a command **"up"** three times, but **the bee steps on the letter 'H'** and that means it has reached the hive successfully collecting **44 units** of nectar.  Position -> [1,0], Nectar -> 39 + 5 = 44, Energy -> 4. | |
| **Input** | **Output** |
| 4  B999  --5-  ---H  ----  right  right  right  down  left  left  left  left  down | Great job, Beesy! The hive is full. Energy left: 6  ----  ----  ---B  ---- |
| **Comment** | |
| The bee starts its movement following the given commands – three times **"right"** and collects the digits *(amounts of nectar)* -> **9+9+9 =27**. Next is a **"down"** command followed by four **"left"** commands *(collects another* ***5*** *units of nectar on the second* ***"left"*** *command)* on the **fourth** left command the bee **leaves** the field boundaries and then **appears** on the **opposite** side of the field **[1,3].** Finally, the bee gets the **"down"** command to reach the hive and the program ends with **32** units of nectar collected. | |
| **Input** | **Output** |
| 4  B---  1991  ----  ---H  down  right  right  right  down  down  left | Beesy did not manage to collect enough nectar.  ----  ----  ----  ---B |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  B-----  111111  ------  111111  ------  H-----  down  right  right  right  right  right  down  left  left  left  left  left  down  right  right  right  right  right | This is the end! Beesy ran out of energy.  ------  ------  ------  --B111  ------  H----- |

# 03. Surfers

**Link:** [**https://alpha.judge.softuni.org/contests/java-advanced-retake-exam-12-august-2024/5012/practice#2**](https://alpha.judge.softuni.org/contests/java-advanced-retake-exam-12-august-2024/5012/practice#2)

*Surfers on the beach represent a vibrant blend of activity, community, and connection with nature. The beach serves as both a playground and a social hub, where surfers prepare for their time in the waves, relax, and share their experiences with others.*

### **Preparation**

Download the skeleton provided in Judge. **Do not** change the **packages**!

**Pay attention to name the package surfers, all the classes, their fields, and methods the same way they are presented in the following document. It is also important to keep the project structure as described.**

### **Problem description**

Your task is to create a repository that stores surfers by creating the classes described below.

### **Surfer**

First, write a class **Surfer** with the following properties:

* **name: String**
* **age: int**
* **experience: int**
* **ownsASurfBoard: Boolean** *(note that it must be a* ***B****oolean not* ***b****oolean)*
* **money: int**

The class **constructor** should receive the **name, age, experience, ownsASurfBoard, and money**. You need to create the appropriate **getters and setters**. All Surfer names will be **unique.** It is guaranteed that there **will be no duplicates** of names.

Override the **toString()** method in the following format:

* **"Surfer {surfer name} is {surfer age} years old and has {surfer experience} years surfing experience."**

### **Beach**

**Next**, write a class **Beach**. The **Beach** class should have those **properties**:

* **name: String**
* **surfboardsForRent: int**
* **surfers: List<Surfer>**

The class **constructor** should receive **name,** and **surfboardsForRent.** Also, it should initialise the **Surfers** with a new **collection** instance.Implement the following features:

* **Method addSurfer(Surfer surfer)** – **adds** an **entity** to the **collection** of surfers if **the surfer has his surfboard** or he has **enough** money to **rent** it.If the surfer **does** not have his **surfboard**, the beach offers the possibility to rent a surfboard for **50 units of money**. If a surfboard is rented, the total number of surfboards for rent on the given beach decreases by one.

If the surfer does not have enough money to rent a surfboard return the following **String**:

* **"{surfer name} has not enough money to rent a surfboard."**

If the surfboards on this beach have run out return the following **String**:

* **"There are no free surfboards."**

If the surfer is added successfully to the beach return the following **String**:

* **"Surfer {surfer name} added."**
* **Method removeSurfer(String name)** – removes a **Surfer** by **given name,** if such **exists**, and **returns boolean** (**true** if it is removed, otherwise – **false**)
* **Method getMostExperiencedSurfer()**– **returns String** - the **most experienced Surfer** by **age of experience** in the given beach return:
* **"{surfer name} is most experienced surfer with {surfer experience} years experience."**

If there are no surfers on the beach, return:

* **"There are no surfers."**
* **Method getSurfer(String name)** – **returns** the **Surfer** with the **given name,** otherwise – returns **null**
* **Method getCount()** – **returns** the **count** of **Surfers** in the given beach
* **Method getSurfersWithPersonalSurfboards()** **–** **returns** **String** **–** returns the list of **surfers** in the given beach who have their surfboards in the following format:
* **"Surfers who have their own surfboards: {surfer name1}, {surfer name2}, …{surfer namen}"**

If there are no surfers on the beach, return:

* **"There are no surfers."**
* **Method getReport()** – **returns** a **String** in the following **format** (print the Surfers in **order of addition**):
  + **"Beach {beach name} was visited by the following surfers:  
    1. {surfer name} with {surfer experience} years experience.  
    2. {surfer name} with {surfer experience} years experience.  
    (…)**

1. **{surfer name} with {surfer experience} years experience."**

If the surfer has **0** years of experience return:

**n. {surfer name} with no experience.**

If there are **no** Surfers on some beaches print **only** this text:

* **"There are no surfers on {beach name} beach."**

### **Constraints**

* The **name** of the **Surfer** will always be **unique**.
* You will always have a Surfer added before receiving methods manipulating the Beach's surfers.

### **Examples**

This is an example of how the **Beach** class is **intended to be used**.

|  |
| --- |
| **Sample code usage** |
| *//Initialize the repositories (Beach)* Beach malibu = new Beach("Malibu", 3);  Beach playaLaRopa = new Beach("Playa La Ropa", 2);  Beach veleka = new Beach("Veleka", 0); *//Initialize entities (Surfer)* Surfer john = new Surfer("John", 40, 10, true, 100);  Surfer mike = new Surfer("Mike", 20, 1, false, 59);  Surfer charlie = new Surfer("Charlie", 55, 19, true, 50);  Surfer hulk = new Surfer("Hulk", 18, 0, false, 49);  Surfer asen = new Surfer("Asen", 28, 10, false, 500);   System.*out*.println(malibu.addSurfer(mike));  System.*out*.println(malibu.addSurfer(john));  System.*out*.println(malibu.getSurfersWithPersonalSurfboards());  System.*out*.println(veleka.addSurfer(charlie));  System.*out*.println(veleka.addSurfer(asen));  System.*out*.println(playaLaRopa.addSurfer(hulk));  System.*out*.println(veleka.getMostExperiencedSurfer());  System.*out*.println(malibu.getReport());  System.*out*.println(playaLaRopa.getReport());  System.*out*.println(veleka.getReport());  //Surfer Mike added.  //Surfer John added.  //Surfers who have their own surfboards: John  //Surfer Charlie added.  //There are no free surfboards.  //Hulk has not enough money to rent a surfboard.  //Charlie is most experienced surfer with 19 years experience.  //Beach Malibu was visited by the following surfers:  //1. Mike with 1 years experience.  //2. John with 10 years experience.  //There are no surfers on Playa La Ropa beach.  //Beach Veleka was visited by the following surfers:  //1. Charlie with 19 years experience. |

### **Submission**

Submit **single .zip file**, containing **surfers** package, **with the classes inside** (**Beach**, **Surfer** and the **Main** **class)**, there is no specific content required inside the **Main** class e.g. you can do any kind of local testing of your program there. However, there should be **main(String[] args)** method inside.