Create a class **Sim** with below attributes:

**simId - int**

**customerName - String**

**balance - double**

**ratePerSecond - double**

**circle - String**

Write **getters, setters** and **parameterized constructor** as required.

Implement static method - **transferCustomerCircle** in Solution class.

This method will take first parameter as **array of Sim** class objects,

second parameter as circle to be transferred (which is String parameter

**circle1**) and third parameter as new circle (which is String parameter

**circle2**).

Method will transfer the customer to new circle (circle2), where the circle attribute would match second parameter (circle1).

Method will return array of Sim objects for which circle is transferred.

Return array should be sorted in descending order of ratePerSecond (assuming ratePerSecond is not same for any of the Sim objects).

This method should be called from main method and display the simId, customerName,circle and ratePerSecond of returned objects (as per sample output).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Consider below sample input and output:

Input:

1

Raj

100

1.5

KOL

2

Chetan

200

1.6

AHD

3

Asha

150

1.7

MUM

4

Kiran

50

2.2

AHD

5

Vijay

130

1.8

AHD

Circles

AHD

KOL

Output:

4 Kiran KOL 2.2

5 Vijay KOL 1.8

2 Chetan KOL 1.6

// Sim.java

public class Sim {

private int simId;

private String customerName;

private double balance;

private double ratePerSecond;

private String circle;

// Parameterized constructor

public Sim(int simId, String customerName, double balance, double ratePerSecond, String circle) {

this.simId = simId;

this.customerName = customerName;

this.balance = balance;

this.ratePerSecond = ratePerSecond;

this.circle = circle;

}

// Getters

public int getSimId() {

return simId;

}

public String getCustomerName() {

return customerName;

}

public double getBalance() {

return balance;

}

public double getRatePerSecond() {

return ratePerSecond;

}

public String getCircle() {

return circle;

}

// Setters

public void setSimId(int simId) {

this.simId = simId;

}

public void setCustomerName(String customerName) {

this.customerName = customerName;

}

public void setBalance(double balance) {

this.balance = balance;

}

public void setRatePerSecond(double ratePerSecond) {

this.ratePerSecond = ratePerSecond;

}

public void setCircle(String circle) {

this.circle = circle;

}

}

// Solution.java

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

public class Solution {

public static Sim[] transferCustomerCircle(Sim[] sims, String circle1, String circle2) {

// Filter and transfer the circle of matching Sim objects

return Arrays.stream(sims)

.filter(sim -> sim.getCircle().equals(circle1))

.peek(sim -> sim.setCircle(circle2)) // Transfer the circle

.sorted(Comparator.comparingDouble(Sim::getRatePerSecond).reversed()) // Sort by ratePerSecond in descending order

.toArray(Sim[]::new);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = 5; // Number of Sims

Sim[] sims = new Sim[n];

// Sample data input

sims[0] = new Sim(1, "Raj", 100, 1.5, "KOL");

sims[1] = new Sim(2, "Chetan", 200, 1.6, "AHD");

sims[2] = new Sim(3, "Asha", 150, 1.7, "MUM");

sims[3] = new Sim(4, "Kiran", 50, 2.2, "AHD");

sims[4] = new Sim(5, "Vijay", 130, 1.8, "AHD");

String circle1 = "AHD";

String circle2 = "KOL";

// Calling the method

Sim[] result = transferCustomerCircle(sims, circle1, circle2);

// Display output

for (Sim sim : result) {

System.out.println(sim.getSimId() + " " + sim.getCustomerName() + " " + sim.getCircle() + " " + sim.getRatePerSecond());

}

scanner.close();

}

}

**Explain the below code:**

return **Arrays.stream(sims)**

**.filter(sim -> sim.getCircle().equals(circle1))**

**.peek(sim -> sim.setCircle(circle2)) // Transfer the circle**

**.sorted(Comparator.comparingDouble(Sim::getRatePerSecond).reversed()) // Sort by ratePerSecond in descending order**

**.toArray(Sim[]::new)**;

**Explanation:**

1. **Arrays.stream(sims)**:
   * Converts the sims array into a stream, allowing us to apply a sequence of operations (such as filtering, mapping, sorting, etc.) on the elements in the array.
   * sims is an array of Sim objects, so Arrays.stream(sims) creates a stream of Sim objects.
2. **.filter(sim -> sim.getCircle().equals(circle1))**:
   * Filters the stream so that only Sim objects with a circle attribute that matches circle1 are kept.
   * The filter method takes a **predicate** (a function that returns a boolean). Here, sim -> sim.getCircle().equals(circle1) is a lambda expression that checks if each Sim object's circle matches the circle1 parameter.
   * Only Sim objects that satisfy this condition are allowed to proceed in the stream.
3. **.peek(sim -> sim.setCircle(circle2))**:
   * peek is a stream operation that allows you to inspect or modify elements in the stream without changing the stream itself.
   * Here, peek(sim -> sim.setCircle(circle2)) is used to update the circle attribute of each Sim object in the filtered stream to circle2.
   * This step essentially performs the "transfer" of each Sim object to the new circle circle2 by updating their circle attribute.
4. **.sorted(Comparator.comparingDouble(Sim::getRatePerSecond).reversed())**:
   * Sorts the Sim objects in descending order based on their ratePerSecond attribute.
   * Comparator.comparingDouble(Sim::getRatePerSecond) creates a comparator that compares Sim objects based on their ratePerSecond attribute in ascending order.
   * .reversed() is then called to flip the order to descending, meaning Sim objects with higher ratePerSecond values appear first in the sorted stream.
5. **.toArray(Sim[]::new)**:
   * Converts the final sorted stream back into an array of Sim objects.
   * toArray(Sim[]::new) specifies that the result should be a Sim array (Sim[]), where each element of the stream will populate the array in its sorted order.

This entire chain processes the original sims array to:

* Filter out only those Sim objects with the specified circle1,
* Update their circle to circle2,
* Sort them in descending order by ratePerSecond, and
* Return them as a new array.

**What is new in toArray(Sim[]::new)?**

In the expression toArray(Sim[]::new), the new refers to the creation of a new array of Sim objects.

Here's a breakdown of the syntax:

* **Sim[]::new** is a **method reference** that tells the stream how to create a new array of type Sim[] (an array of Sim objects).
* The ::new part is shorthand for the array constructor. When we say Sim[]::new, it’s equivalent to saying, "create a new array of Sim objects."
* **toArray(Sim[]::new)** specifies that the resulting elements of the stream should be placed into a newly created array of Sim objects.

In simpler terms, this method reference Sim[]::new lets the stream know the type of array to create and tells it to use the new constructor for the array.