Exercise 1: Conversion Table (Recap)

As a recap of last week's topics, you have to program a **Conversion Table for Celsius and Fahrenheit values**! The difficulty lies in writing a method for converting from Celsius to Fahrenheit and determining the correct scaling for displaying the values.

A source code for the class `CelsiusFahrenheit` is given and must be *completed* and *adapted* at various points.

```java
import static java.lang.Math.round;
public class CelsiusFahrenheit {

    // Add missing method here

    public static void main(String[] args) {
        System.out.println("Celsius Fahrenheit Converter");
        System.out.println("============================");
        System.out.println("Celsius \t Fahrenheit");

        for (int c = 5; c < 20; c++) {

            System.out.println(c + "\t" + celsiusFahrenheit(c));
        }

    }
}
```

The following output is expected:

```
Celsius-Fahrenheit-Konverter
============================
Celsius     Fahrenheit
5           41
6           43
7           45
8           46
9           48
10          50
```

Exercise 2: Racing Car Game

We are going to create a simple **Car Racing Game**. First we create a class `RacingCar` with the following attributes:

- `racer` stores the name of the driver. This attribute must be a non-empty `string` and should be initialized when the object is instantiated.
- `speed` stores the speed of the car. This attribute can only contain non-negative integer values and must be **less than or equal to** a maximum speed.
- `pos` is an integer that specifies the position of the car and can only have non-negative values.

Each car also has the following additional attributes:

- `maxSpeed`, which specifies the maximum speed the car can have. This attribute should be initialized when the object is instantiated.
- `finish`, which stores the target distance that the car must travel. It should be set to -1 during initialization.

The class has the following methods:

- `start(initSpeed, finishDistance)`: sets the speed of the car to an initial value, sets the target distance to the passed value and also sets the position of the car to 0.
- `race(acceleration)`: takes an integer value for the acceleration; first adjusts the speed of the car and then updates the position of the car.
- `isFinished()`: calculates a Boolean value (**`true`** or **`false`**) and indicates whether the car has reached the finish line.

Now your tasks:

1. Create a **Class Diagram** to match the textual description. Discuss with the person sitting next to you.
2. **Implement** the class! Use the program code provided as a template. The `main` method must be executable without errors.

Exercise 3: Savings Account

The task is to implement the transactions *withdraw money* and *deposit money* as well as *display account balance* in a program. If the account balance is below 0 Euro, a warning is to be issued.

1. **Design** and **implement** a class `Sparbuch` that provides the appropriate methods for these transactions. Make sure that your class `Sparbuch` can be integrated into the already provided class `Bankautomat`. The `Bankautomat` class serves as the main program and provides an info screen for interaction with the user.
2. The class `Bankautomat` currently only asks for one option and then terminates the program. Can you extend the program so that it stays in the menu until the user selects the “Exit” option?
3. (Bonus) Currently, things can still go wrong if the user does not adhere to the expected input formats. Test the program with different inputs and try to implement the `Bankautomat` class more robustly.
4. (Bonus) When the program is terminated, the current account balance is lost. How could this be solved?