

In this exercise you were given four methods to define, focusing on transforming values passed in to the methods through parameters to produce return values, often using for loops. There were two versions of this exercise – though the scenario was a little different in both cases the structure of the solution was the same in both.

METHOD 1

The first method was described as follows:

```
/* 50 COMPETENCY POINTS
 * Question 1: Balance after one day
 *
 * Given a starting loan balance, and an annual interest rate, return
 * the balance after the interest has been added.
 *
 * Compute a daily rate by dividing the annual rate by the number of days in a
 * year, 365.
 *
 * To compute the new balance, add one to the daily rate, then multiply by the
 * balance.
 */
```

You were given a “stubbed out” implementation of the method. A method stub is minimal implementation of a method that returns a default value. For example, for numeric types, the default value is zero, whereas for booleans it is false.

```
public double question1(double balance, double annualRate) {
    return 0.0; // change the return value!
}
```

The inputs to this method represent the an initial account balance and an annual interest rate. Both of these values are of type double. The method is supposed to return the balance that results after one day’s worth of interest has been added to the initial balance. Two specific directions were given:

1) Compute a daily rate by dividing the annual rate by the number of days in a year, 365.

For example, if the annual rate is 0.10 then the daily rate will be $0.10/365.0$, or approximately 0.000274. In code we could write:

```
double dailyRate = annualRate/365;
```

2) To compute the new balance, add one to the daily rate, then multiply by the balance.

The new balance will be calculated by the formula

$$\text{balance} * (1 + \text{dailyRate})$$

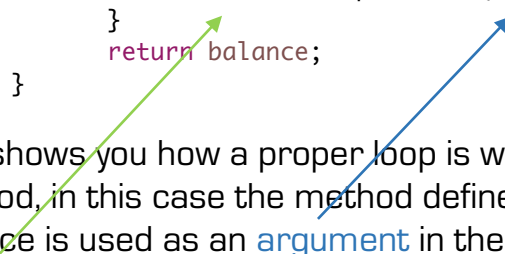
Putting this together we arrive at the following code:

```
public double question1(double balance, double annualRate) {  
    double dailyRate = annualRate/365;  
    return balance * (1 + dailyRate);  
}
```

METHOD 2

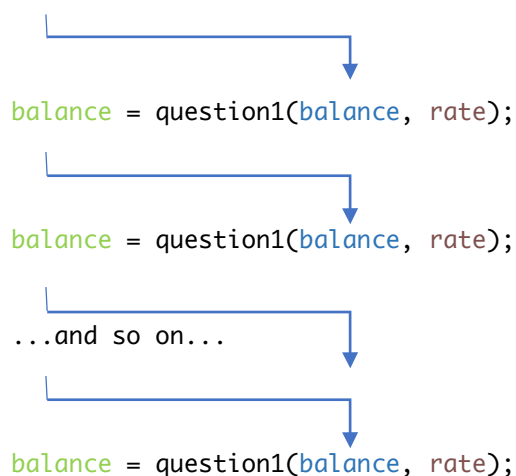
The second method was given to you:

```
/*
*****
* THIS ONE IS DONE FOR YOU - STUDY HOW IT IS DEFINED - IT MAY HELP YOU
* IN QUESTION 4
*****
* 0 COMPETENCY POINTS
* Question 2: Balance after several days
*
* Given a starting loan balance, an annual interest rate, and a number of
* days, return the balance after the interest has been added to the balance
* after each of those days.
*
* HINT: use a loop and the method from question 1.
*/
public double question2(double balance, double rate, int days) {
    for (int i=0; i<days; i=i+1) {
        balance = question1(balance, rate);
    }
    return balance;
}
```



This shows you how a proper loop is written, and shows you how to call another method, in this case the method defined in question 1. Notice that the value of `balance` is used as an **argument** in the call to `question1`, and it is also the **variable** to which the new balance is saved to. This is what achieves the compounding effect in the interest calculation. We can visualize successive calls to `question1` as follows

```
balance = question1(balance, rate);
└──────────────────────────────────┘
balance = question1(balance, rate);
└──────────────────────────────────┘
balance = question1(balance, rate);
└──────────────────────────────────┘
...and so on...
└──────────────────────────────────┘
balance = question1(balance, rate);
```



There is a hint given too: **STUDY HOW IT IS DEFINED - IT MAY HELP YOU IN QUESTION 4**

METHOD 3

The third method was described as follows:

```
/* 50 COMPETENCY POINTS
 * Question 3: Balance at end of period (payment made at end of period)
 *
 * Given a starting loan balance, an annual interest rate, a number of days,
 * and a payment that is made at the END of the period, return the remaining
 * loan balance (i.e. after interest has compounded daily during the period,
 * and the payment has been deducted at the end).
 *
 * HINT: use the method from question 2
 */
```

You were again given a “stubbed out” implementation of the method:

```
public double question3(double balance, double rate, int days, double payment) {
    return 0.0; // change the return value!
}
```

The inputs to this method are a double representing an initial balance, a double representing an annual interest rate, an int representing the number of days over which to compound interest, and a double representing a payment made at the end of the compounding period.

The scenario here is that we’re paying down a loan. At the end of each loan period a payment is due. The key here is to realize that most of the work has already been done by the second method – we can call that method (see example of how to call method in question2 above) to get the compounding done, and then subtract the payment:

```
public double question3(double balance, double rate, int days, double payment) {
    return question2(balance, rate, days) - payment;
}
```

For the other version of the exercise the scenario was putting money into a retirement account. For that situation the contribution is made at the start of the compounding period – so we add the contribution to the initial balance **before** the compounding is done:

```
public double question3(double balance, double rate, int days, double contribution) {
    return question2(contribution + balance, rate, days);
}
```

METHOD 4

The fourth method was described as follows:

```
/* 50 COMPETENCY POINTS
 * Question 4: Balance at the end of several periods
 *
 * Given a starting loan balance, an annual interest rate, a number of days,
 * a payment that is made at the END of the period, and the number of periods,
 * return the remaining loan balance.
 *
 * HINT: use a loop and the method from question 3.
 */
```

Here's the provided method stub:

```
public double question4(double balance, double rate, int daysPerPeriod, double payment,
int periods) {
    return 0.0; // change the return value!
}
```

The inputs are: the initial balance, the annual interest rate, the number of days in the period of time between payments, and the amount of the payment.

The idea here is to call question3 repeatedly, and update the balance each time through the loop. Recall how this was done in question 2: the balance was used both as an argument in the method call and as the variable where the result was stored. The pattern is the same in this method:

```
public double question4(double balance, double rate, int daysPerPeriod, double payment,
int periods) {
    for (int i=0; i<periods; i=i+1) {
        balance = question3(balance, rate, daysPerPeriod, payment);
    }
    return balance;
}
```

METHOD 5

The fifth method was described as follows:

```
/* PROFICIENCY
 * Question 5: How many periods until loan is paid off
 *
 * Determine how many payments need to be made to pay off the entire loan
 * amount.
 *
 * Determine how many payments are needed by repeated making payments as long
 * as the balance is greater than zero. Once the balance is zero or negative,
 * return the number of payments that were made.
 */
```

Here's the provided method stub:

```
public int question5(double balance, double rate, int daysPerPeriod, double payment) {
    return 0; // change the return value!
}
```

The idea behind this method is to count how many payments need to be made to bring the balance down to less than or equal to zero. One way to determine this is to try ever increasing numbers of payments and see when the remaining balance dips to zero or lower.

```
public int question5(double balance, double rate, int daysPerPeriod, double payment) {
    int numberOfPayments = 0;
    for (int i=0; balance > 0; i=i+1) {
        balance = question3(balance, rate, daysPerPeriod, payment);
        numberOfPayments = i+1;
    }
    return numberOfPayments;
}
```
