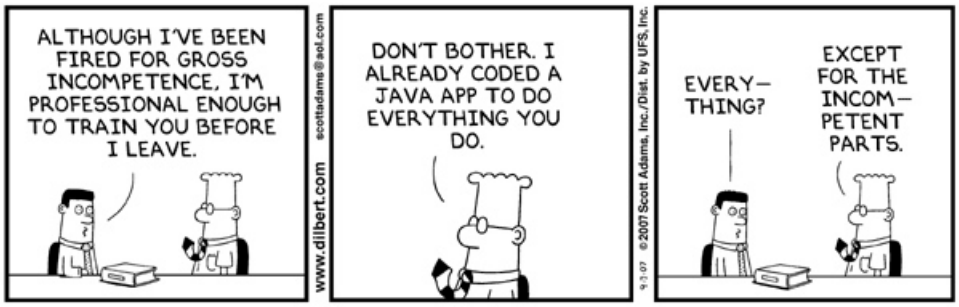
CS62 - Java Practice



1 WarmUp

Static functions are functions that you can call without instantiating the class. If you're within the class (in the main method) you can simply call them by their name. If you're outside of the class, you have to say the class name, followed by a period, followed by the name of the static method.

Write the following static methods in the `WarmUp` class (I've provided you with the method header for the first, but you'll need to add the others:

- Write a static method called `printVertical` that takes a `String` as a parameter and prints all of the characters in the string vertically on a line by themselves.

  For example:

  ```
  printVertical("I love CS")
  ```

  would show in the console:

  ```
  I

  l
  o
  v
  e

  C
  S
  ```

– Write a static method called `savings` that takes a dollar amount and an interest rate (e.g. an interest rate of 1% would be passed as 0.01) and calculates how much money you would have if you put that money in a savings account with that interest rate, *including the original amount*

  For example:

  ```
  savings(1000, .1)
  ```

  would give you back 1100.

– Write another static method also called `savings` that takes a **third** parameter, the number of years that you plan to leave the money in the savings account. The function should calculate how much money you'll have if you leave it in the savings account for that number of years. You can assume the interest is only compounded yearly (i.e. calculated and added up each year). Notice that in Java you can define two methods with the same name as long as the parameters are different!

  For example:

  ```
  savings(1000, .1, 3)
  ```

  would return 1331.

  You **must** use your other `savings` method in writing this method.

– Write a *recursive* function called `hailstorm` which takes a number as a parameter and prints out the hailstorm sequence starting at this number. The hailstorm sequence is calculated as follows:

  – print out the number
  – if the number is 1, the sequence is done.
  – if the number is even, then the sequence is continued with the number divided by 2.
  – if the number is odd, then the sequence is continued with three times the number plus 1.

  (See `http://plus.maths.org/content/mathematical-mysteries-hailstone-sequences` for more examples).

2 My First Class (from scratch)

Fill in the details for the `Person` class. The class should support the following methods:

- A constructor that takes a first name and a last name. The default age of a person should be 18.

- `getFirst`, `getLast` which don't take any parameters, but return `Strings` representing the first and last name respectively.
- `getAge` which doesn't take any parameters and returns an integer representing the age.
- `anotherYear`: takes no parameters but increases the persons age by a year (and doesn't return anything).

For example, if you ran the following main method:

```
public static void main(String[] args){
    Person me = new Person("Teran", "Cole");
    System.out.println(me.getFirst() + " " + me.getLast() + " is " + me.getAge());

    me.anotherYear();
    System.out.println("and now: " + me.getAge());
}
```

You would see printed out in the console:

```
Teran Cole is 18
and now: 19
```

You may use whatever instance variables you like to implement this. Make sure to declare all instance variables and methods as either `private` or `public` as appropriate.

3 A Better Card Class

For the last part of this assignment, I want you to change the functionality of the `Card` class that we looked at in class. Make the following changes to the class:

- Change the `getNumber` method to return a `String`. We want it to return "Ace", if it's a 1, 2-10 (as a `String`) if it's a 2-10, "Jack" for 11, "Queen" for 12 and "King" for 13. You should **NOT** change the type of the instance variable `number`. Instead, simply rewrite the `getNumber` function to calculate the appropriate `String` depending on the stored number.

  To convert an integer into a `String` there is static method called `Integer.toString()` that you can call. For example,

  ```
  String s = Integer.toString(10)
  ```

  would have the `String` 10 (i.e. "10") in the variable `s`.

- We want to keep track of whether or not the card is face up or face down. By default, when a card is constructed, it should be face down (having it take a default value keeps us from having to add any extra parameter to the constructor). Add a method called `isFaceUp` that doesn't have any parameters aand returns `true` if the card is face up and `false` if the card is face down.
- Add a method called `flip` that changes whether a card is face up or face down, that is, if it's face up it will be face down after and if it's face down it will be face up after. The method should not take any parameters and should not return a value, but will change the state of the card.

After you make these changes, you should be able to run the following main method:

```
public static void main(String[] args){
    Card card = new Card(10, "hearts");
    Card jack = new Card(11, "spades");
    Card ace = new Card(1, "clubs");

    System.out.println(card.toString());
    System.out.println(jack.toString());
    System.out.println(ace.toString());

    System.out.println("--------CHEATING------");
    card.cheat();
    System.out.println(card.toString());
    System.out.println(jack.toString());
    System.out.println(ace.toString());

    System.out.println("--------BEFORE FLIPPING-------");
    System.out.println(card.toString() + ": " + card.isFaceUp());
    System.out.println("--------AFTER FLIPPING-------");
    card.flip();
    System.out.println(card.toString() + ": " + card.isFaceUp());
    System.out.println("--------AFTER FLIPPING-------");
    card.flip();
    System.out.println(card.toString() + ": " + card.isFaceUp());
}
```