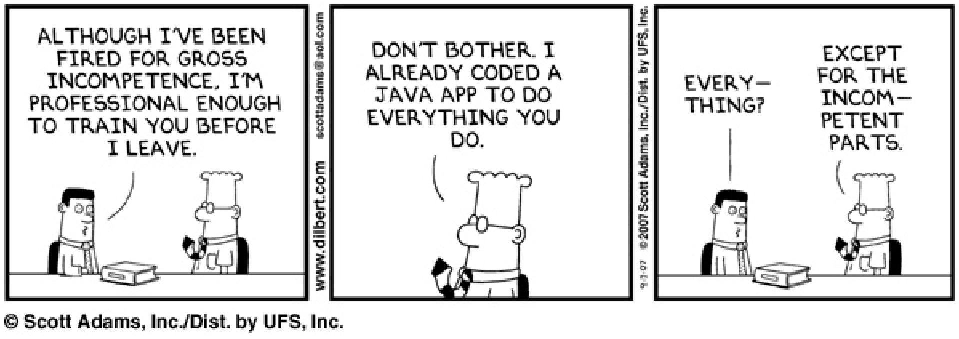
CS201 - Assignment 1, Part 2
Due: Wednesday February 19, at the beginning of class


ALTHOUGH I'VE BEEN FIRED FOR GROSS INCOMPETENCE, I'M PROFESSIONAL ENOUGH TO TRAIN YOU BEFORE I LEAVE.
DON'T BOTHER. I ALREADY CODED A JAVA APP TO DO EVERYTHING YOU DO.
EVERY-THING?
EXCEPT FOR THE INCOM-PETENT PARTS.
© Scott Adams, Inc./Dist. by UFS, Inc.

1 Java Practice

To give you some practice writing functions/methods in Java, for the first part of this assignment, you're going to write a few stand-alone functions. Because Java is object-oriented, if you just want to write stand-alone functions, you still have to write them within a class.

Create a new project in Eclipse called `assign1.2` (look at assignment 1, part 1 if you need a refresher on how to do this). Create a new class called `WarmUp`, which you will put your functions into. To write stand-alone functions that act like functions you may have seen in Python (or NetLogo) you need to write them inside a class and declare them `public static`. Once you've done this, you can call them directly within your `main` method. For example:

```
public class WarmUp{
    public static int reallyBoringFunction(){
        return 1;
    }

    public static void main(String[] args){
        System.out.println(reallyBoringFunction());
    }
}
```

Write the following functions in your `WarmUp` class:

1. Write a function 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
```

2. Write a function 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.

3. Write another function 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 functions with the same name as long as the parameters are different!

   For example:

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

   would give you back 1331.

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

4. 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

Create a new class (and file) called `Person`. 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` and `getAge`.
- `anotherYear`: takes no parameters but increases the persons age by a year.

For example, if you ran the following main method:

```
public static void main(String[] args){
    Person me = new Person("Steve", "Perry");
    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:

```
Steve Perry 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 `EvenBetterCard` class that we looked at in class on Friday (look in the lecture notes on the course web page for the code).

Create a new class called `MyCard` and copy and paste all of the code from the `EvenBetterCard` class into this file. You'll need to change the class name and constructor name to be `MyCard` (and probably delete the main method for now).

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 a function 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 wether or not the card is face up or face down. By default, when a card is constructed, ts should be face down (having it take a default values keeps us from having to add any extra parameters to the constructor). Add a method called `isFaceUp` that 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.

- Add the following method which will help us print out the card more easily:

  ```
  public String toString(){
      return getNumber() + " of " + getSuit();
  }
  ```

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

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

4 When You're Done

You should now have three files to submit in your project. Put comments at the top of each of these files with your name and the assignment number.

To group these files into a single file, you need to export your project. To do this:

1. Right-click on the project (on Mac, ctrl+click) and select `Export`.
2. You'll see a number of options. Open up the `Java` folder and select `JAR file` and click `Next`.
3. You should just see your project selected. Below, make sure **only** the following two options are checked:
   - "Export Java source files and resources"
   - "Compress the contents of the JAR File"
4. Click on the `Browse...` button and pick a location to save the output file. Give the file a name like "kauchak1.jar", where "kauchak" is your last name and "1" is the assignment number.
5. Click `Finish`.

If all went well this will generate a single `.jar` file wherever you picked to save it.

Submit this JAR file as assignment number "1.2" via the online submission mechanism on the course web page.