CS 246 **Software Design and Development**

Cleverness

Begat-O-Matic *Exploration*

Goal

The purpose of this exploration is for you to explore string tokenizers, message formats, and basic object orientation — among other clever things.

Requirements

Start with the supplied stub `BegatOMatic.java` file. The `main` method of the *BegatOMatic* class has room for **exactly** two lines of code in it, one of which is already there, and the other you will figure out and write yourself. The highly special purpose of this program is simply to output nine verses from Genesis chapter 5, verses 9-17 specifically, which have a very regular structure. The printing procedure goes like this:

Print three times, filling in new data between the '< >' angle brackets each time:

```
<verse number>. And <father> lived
<number of years father lived before he begat son> years, and begat <son>:
<verse number + 1>. And <father> lived after he begat <son>
<number of years father lived after he begat son>, and begat sons and daughters:
<verse number + 2>. And all the days of <father> were <total years father lived> years: and he died.
```

Replace `<verse number>` with `<verse number + 3>`, replace `<father>` with `<son>` and repeat with new data.

Create any classes of your own you think might be useful and helpful, but you **must** use the *java.text.MessageFormat* class in your solution.

Furthermore, you **may not** use classes from any other package (other than *java.lang*), or written by anyone else not in the class. In other words, there is only one import statement you are allowed (and you're not allowed NOT to use it!).

This means, of course, that any file I/O or database I/O is forbidden. Hardwiring data is likewise forbidden (except for *lookup strings*, format templates and certain constants.). This is intended as a test of your ingenuity, to see what you can come up with given these constraints. Do your best!

Remember, the output of your program must look **exactly** like Genesis 5:9-17, i.e.:

```
9. And Enos lived ninety years, and begat Cainan:
10. And Enos lived after he begat Cainan eight hundred and fifteen years, and begat sons and daughters:
11. And all the days of Enos were nine hundred and five years: and he died.
12. And Cainan lived seventy years, and begat Mahalaleel:
13. And Cainan lived after he begat Mahalaleel eight hundred and forty years, and begat sons and daughters:
14. And all the days of Cainan were nine hundred and ten years: and he died.
15. And Mahalaleel lived sixty and five years, and begat Jared:
16. And Mahalaleel lived after he begat Jared eight hundred and thirty years, and begat sons and daughters:
17. And all the days of Mahalaleel were eight hundred and ninety and five years: and he died.
```

Grading Criteria

The rubric below is meant to guide you in your quest for exceptional quality.

	Exceptional 100%	**Good 90%**	**Acceptable 70%**	**Developing 50%**	**Missing 0%**
Correctness 30%	Output matches exactly (tested with the supplied `brt0.sh` script, used **without** modification). No input required and no hardwired data (except lookup strings).	Output matches exactly. No input required, and no hardwired data (except lookup strings).	Output matches almost exactly. No input required, and no hardwired data (except lookup strings).	Output matches almost exactly. Some input required, and/or hardwired data (other than lookup strings).	Output does not match.
Elegance 40%	Used MessageFormat class and String.split method appropriately, had minimal repeated code to do the output, and used **encapsulation** to handle the data manipulating and formating.	Used MessageFormat class and String.split method appropriately, had minimal repeated code to do the output.	Used MessageFormat class and String.split method appropriately, but had too much repeated code to do the output.	Used MessageFormat class or String.split method appropriately, but not the other.	Used classes from other classes, and did not use MessageFormat class or String.split method.
Style 20%	Heeded all requirements in our class formatting guidelines.	Paid attention to whitespace and brace alignment. Wrote good comments.	Paid attention to whitespace and brace alignment. Wrote some comments.	Paid attention to whitespace and brace alignment, but no comments.	No attention paid to formatting.
Applied the principle of Discovery of Self-driven knowledge 10%	See below.	See below.	See below.	See below.	See below.

The last 10% will be earned by applying the principle of Discovery and Self-driven knwledge with the following tasks. These tasks are cumulative, so you have to do earlier ones before doing later ones, and their weight increases the further you go.

1. Write a helper class and use the supplied `brt1.sh` script without modification (1%).
2. Use that same helper class and the supplied `brt2.sh` script modified only in the `buildJar` function definition. Also, made `BegatOMatic2` a subclass of `BegatOMatic` or `BegatOMatic1` in the most appropriate way (2%). Don't build your jar file the way Sierra and Bates suggest to do it. Specifically, figure out how to avoid making a manifest file.
3. Use that same helper class and the supplied `brt3.sh` script with a one-line addition (to set up a data environment) so that `BegatOMatic3` works with the additional use of the `java.io.File` class **only** (3%).
4. Use that same helper class and the supplied `brt4.sh` script with a one-line addition (to set up a data environment) so that `BegatOMatic4` works by using a class from the `java.util` package (Hint: You may not use system environment variables, but you can use properties). Note that this one-line addition is different than the one needed for the `brt3.sh` script (4%).

The real challenge is to do number 4, but without DIRECTLY using ANY class from the `java.io` package. Up to 5 bonus points will be given in the remarkable case you figure out how to do it (with `brt5.sh` and `BegatOMatic5` used appropriately).

The `brt?.sh` scripts are all found in the `clever.zip` file or the `/home/cs246/samplecode/clever` directory.

When you are finished, tar up your scripts and java files (do not include `*.class` files), and submit via the Linux Lab `submit` command to: burton, cs246, cleverness.

Comments

For your consideration, here are some comments past CS246 students have made about this exploration:

- I really enjoyed exploring a couple of different classes that I didn't know existed. Being able to think about other ways of producing a program and using different methods I believe makes me a better programmer because I am able to have a better understanding of what all Java can do and find different ways of approaching thinking and solving problems. It has been interesting being able to collaborate more with other students. This seems to have always been verboten, but it does enhance the processes of critical thinking and brings other perspectives to solving a problem. Programming is more of an art than most people give it credit for, and admittedly I'm not an artist, I just enjoy learning and trying to upgrade my thinking processes and skills.
- What I noticed is that for me to build any kind of useful design, I really need to fully understand the problem I'm working with. This pattern matching exercise took me some time to understand, mostly through trial and error. Once I got a working solution, then I felt like I understood the delta points and could re-factor properly. This problem space understanding just takes time for me. Except for some before class discussion with the whole class, I didn't collaborate with anyone for this and I wish I had.
- I liked the amount of new information that I learned about Java while exploring solutions to this program: especially the usefulness of try/catch blocks. I applied part of the gersy principle (go beneath the surface) but failed to apply the other part (don't dig too deep). I dug in the wrong direction several times and too deep while looking for solutions. However, I did learn a few valuable things from digging too deep. I would have liked to have collaborated more with others on this program.
- At first this project seemed really difficult and I had no idea what to do. I worked with [a classmate] and we figured out quite a bit together. It's very useful working together because as we would work I would catch syntax errors or any other small issues that there might be with the code before we even tried to compile. It is so useful to have two different points of view because you both catch different things. I love collaboration. Once I realized the power of the [secret] things kind of flowed from there. It took me a little while to figure out the syntax for the message format and the tokenizer, but that wasn't too difficult. This was a very beneficial exploration. I learned a lot!