Lab Solutions: Experimenting with Enumerated Types

1 Instructions: Omitted.

2 Getting Ready: Omitted.

3 Using Integer Constants Instead of Enumerated Types: It is common practice to use a group of integer contants rather than an actual enumerated type. This part of the lab will help you see some of the problems with this approach.

1. Open `Constants`, `SemesterUtiltities`, and `Example1.java` in the editor.
2. Compile `Constants`, `SemesterUtiltities`, and `Example1.java`.
3. Execute `Example1`.
4. What output was generated?

```
Session starting months in 2006:
January
August
```

5. Change `FALL` to `TALL` in `Constants.java`.
6. Compile `Constants.java`.
7. Execute `Example1`.
8. What output was generated?

```
Session starting months in 2006:
January
August
```

9. You might be surprised that `Example1` executed given that it uses `Constants.FALL` which is no longer defined. Why did it execute? (You may not know the answer to this question but you should be able to make some conjectures.)

   The values of the constants in the `Constants` class are compiled into `Example1`.

10. Compile `Example1.java`.
11. What error was generated?

```
Example1.java:10: cannot find symbol
symbol  : variable FALL
location: class Constants
       for (int i=Constants.SPRING; i<=Constants.FALL; i++)
                                                  ^
1 error
```

12. Change `TALL` back to `FALL` in `Constants.java`.

13. Add the constant `SUMMER` to `Constants.java` and assign it the value 2.

14. Compile `Constants.java` and `Example1.java` (remembering that it is very important to compile `Example1.java` even though it did not change).

15. Execute `Example1`.

16. What output was generated?

```
Session starting months in 2006:
January
August
```

17. What is wrong with this output and what caused the problem?

    The summer session is missing because we added the `SUMMER` session "after" the `FALL` session.

18. Put `SUMMER` and `FALL` in the proper order (in `Constants.java`) and adjust their values accordingly.

19. Compile `Constants.java` and `Example1.java`.

20. Execute `Example1`.

21. What output was generated?

```
Session starting months in 2006:
January
August
August
```

22. What is wrong with this output and what caused the problem?

    The summer session is shown starting in August when, in fact, it starts in May. This is a problem with the `startingMonth()` method in the `SemesterUtilties` class shich should have been changed when `Constants.java` was changed (but the compiler did not force us to make a change).

23. Add the line:

```
System.out.println(SemesterUtilities.startingMonth(7));
```

    to the `main()` method in `Example1.java`.

24. Will this version of `Example1.java` compile?

    Yes, it is syntactically correct.

25. What output would be generated by this statement?

```
August
```

26. Why is this somewhat troubling?

Because the `startingMonth()` method can be passed any `int` but we'd really like the compiler to make sure that we are passing a valid semester (i.e., we'd like the compiler to perform type-checking).

27. What output would be generated by the statement:

```
System.out.println(Constants.SPRING);
```

`0`

28. What output would be more informative for the previous statement?

Something like "Spring" would certainly be much more descriptive.

4 Using String Constants Instead of Enumerated Types: It is also common practice to use a group of `String` contants rather than an actual enumerated type. This part of the lab will help you see some of the additional problems with this approach.

1. Open `Example2.java` in the editor.
2. Compile and execute `Example2.java`.
3. What output was generated?

```
Grade on first  attempt: B-
Grade on second attempt: B+
```

4. The method `String.compareTo(java.lang.String)` can be used to compare two `String` objects. Use this method to assign `second` to `better` if `second.compareTo(first)` is greater than 0 and to assign `first` to `better` otherwise.
5. What code did you add to the `main()` method in `Example2.java`.

```
if (second.compareTo(first) > 0) better = second;
else                             better = first;
```

6. Add the line:

```
System.out.println("Better grade: "+better);
```

to the end of the `main()` method in `Example2.java`.

7. Compile and execute `Example2.java`.
8. What output was generated?

```
Grade on first  attempt: B-
Grade on second attempt: B+
Better grade: B-
```

9. What is wrong with this output and why?

A "B+" is better than a "B-". Unfortunately, "B-" is lexicographically larger than

"B+".

10. How does a "B" compare to a "B-" at JMU and in Java?

    A "B" is better than a "B-" at JMU but "B-" is lexicographically larger than "B" in Java.

5 Using a Simple Enumerated Types: This part of the lab will help you see some of the advantages of using enumerated types. You will explore some of the other benefits later.

1. Open `Example3.java` in the editor.
2. Compile and execute `Example3.java`.
3. What output was generated?

   ```
   First  is better
   ```

4. What determines the order used by the `compareTo()` method?

   The order in which the values appear in the enumeration.

5. What `.class` files were created when you compiled `Example3.java`? (Hint: Look for all files that start with "Example3" and end with ".class".)

   ```
   Example3$LetterGrade.class
   Example3.class
   ```

6. Move the lines:

   ```
   public enum LetterGrade
   {
          F, D, DPLUS, CMINUS, C, CPLUS, BMINUS, B, BPLUS, AMINUS, A;
   }
   ```

   from `Example3.java` and into a file named `LetterGrade.java`.

7. Delete all of the `.class` files in the directory you created for this lab.
8. Compile `LetterGrade.java` and `Example3.java`.
9. What `.class` files were generated?

   ```
   Example3.class
   LetterGrade.class
   ```

10. Execute `Example3.java`.
11. What output was generated?

    ```
    First  is better
    ```

12. Suppose JMU instituted a grade of "D-". What changes would you need to make to `LetterGrade.java`?

    Add a `DMINUS` between `F` and `D`.

6 Creating a Simple Enumerated Types: This part of the lab will give you some experience creating a simple enumerated type.

1. Create an enumerated type named `Sessions.java` that includes a Spring, Summer, and Fall semester (in the appropriate order for a calendar year).

```
public enum Sessions
{
   SPRING, SUMMER, FALL;
}
```

2. Modify `SemesterUtilities.java` and `Example1.java` so that they work correctly with this enumerated type. (Hint: Think about using a "for each" loop.)

```
public class SemesterUtilities
{
    public static String startingMonth(Sessions semester)
    {
       String      month;

       month = "N/A";

       if      (semester.equals(Sessions.SPRING))  month = "January";
       else if (semester.equals(Sessions.SUMMER))  month = "May";
       else if (semester.equals(Sessions.FALL))    month = "August";

       return month;
    }
}
```

```
public class Example1
{
    public static void main(String[] args)
    {
       int         semester;

       System.out.println("Session starting months in 2006: ");

       for (Sessions s: Sessions.values())
       {
          System.out.println(SemesterUtilities.startingMonth(s));
       }
    }
}
```

Department of Computer Science
