Assignment 1: Food Expiry Dates Tracker (15% of course total)

- Due date is **1 Oct, 11:59pm**. Refer to Canvas for any updates.
- Make sure you also complete the corresponding online quiz at Canvas to receive full portion of the coding part. Refer to the assignment description on Canvas and Marking Guide for details.
- Late penalty is 10% per calendar day (each 0 to 24 hour period past due), max. 2 days late (20%).
- The assignment might be graded automatically. Make sure that your code compiles without warnings/errors and produces the required output. Also use the file names and structures indicated. Deviation from that might result in 0 mark.
- This assignment is to be done individually. Do not show another student your code, do not copy code from another person/online. Post all questions on Canvas (without your solution).
- You may use general ideas you find online and from others, but your solution must be your own.
- Your code MUST compile and run, or marks will be deducted. Unless otherwise specified, your code MUST start from an empty project in IntelliJ and use OpenJDK 16 (or any sub-versions).
- You may use any tool for development. But we will be grading your submission using IntelliJ IDEA Community, so make sure your code compiles and runs there. We support only IntelliJ.

Description

In this assignment you are going to implement a text-based query system of food expiry dates. This system will allow a user to maintain a list of food items recording a small collection of their attributes.

Overall Requirements

You must have (at least) the following three classes (in separate .java files):

- A class for holding food information: name (String), notes (String), price (double), and expiry date (LocalDateTime).
    - Name may contain more than one word (e.g., “Peanut butter cookies”), so is notes (e.g., “Contains nuts”). Name cannot be empty but notes can.
    - This class must correctly implement (override) toString(), such that when the object is passed to System.out.println() as a parameter the information will be displayed. At the end **also display how many days is left before the food expires** (see sample output).
- A class for a text menu
    - Have a field to store the menu's title (String). Come up with something fun!
    - Have a field to store the menu's options (as an array of Strings).
        - The menu option strings should not include numbers. So instead of having it store {“1. Do thing one”, “2. Do thing two”}; have it store {“Do thing one”, “Do thing two”} and your application generates the numbers automatically.
    - Have a method to display (print) the menu to the screen.
        - Your application must automatically place a rectangle of #'s around the menu's title, sized to the length of the title. This must be computed, not hard-coded!
        - Follow the title banner with **the current system date**.
        - Automatically number the options starting at 1.
    - Have a method to read user’s input and carry out the desired option.
        - Your application must be able to handle invalid inputs of the expected type. See the requirements below for details.

- A class for the main application
    - Contains the main() method which uses the menu and the food item classes to implement the application.
    - Creates an ArrayList of food items to hold the list of food items the user enters.
    - Be careful not to have much duplicate code in your application! Use methods.

For this assignment, put all your classes are in one package (cmpt213.assignment1.foodexpdatestracker).

Your system must be able to save the food-items-list in JSON format (e.g., a JSON array object). When the system runs for the first time, there is no JSON file to load and this list is empty (because no food items have been added). When the user exits the system, your system automatically generates a JSON file and save it in your project folder (use a hard-coded relative path starting with ./). When the user runs the system again (and subsequently), this JSON file will be loaded and used to populate the list. When the user exits again, this file will be updated with the new information. Both the loading and saving are done automatically by the application – the user doesn't need to do anything to trigger that.

Text Interface Requirements

- When you prompt the user to choose a menu option, if the user enters an invalid number, you must re-ask the user to enter a valid value.
    - You may assume the user always enters correct type of data: when asked for an int, it is OK if the application crashes when the user enters a non-int such as 'A'.
- **Main Menu Option 1: List all food items**
    - For each food item, first show the item number, then show one field per line including: food name, notes, price, and expiry date (in the format YYYY-MM-DD).
        - The food item number is an automatically generated counter (starting with 1).
        - The list is ordered by their expiry dates (oldest first).
    - If there are no food items, display "No food items to show." instead.
- **Main Menu Option 2: Add a food item**
    - Ask user for name, notes, price, and expiry date of the food item. Handle cases when an invalid number is inputted for prices (e.g., negatives) and as part of the expiry date.
        - Expiry date is specified by year, month, day. You can set the time to 11:59p.
        - To check if an expiry date is valid, look up the "LocalDate.of" static method.
    - When done, display an acknowledge message. For example, if "Peanut butter cookies" is added, display "Peanut butter cookies has been added to the list.".
- **Main Menu Option 3: Remove a food item**
    - List all food items currently in the system along with their automatically generated item number. Ordered by their expiry dates (oldest first).
        - If there are no food items, display "No food items to show." instead.
    - Otherwise, allow the user to select a food item (by number), or 0 to cancel.
    - Entering an invalid number (like -3, or a value more than the number of food items) needs to be handled by the application (tell the user that it is invalid and ask for a number again). Entering invalid data type (e.g., "hello") does not need to be handled.
    - When done, display an acknowledgement message. For example, if "English muffin" is removed, display "English muffin has been removed from the list.".
- **Main Menu Option 4: List expired food items**

    - List all food items with expiry date before the current date, ordered by their expiry dates (oldest first). Food items that are expiring on the current date are not expired.
    - If there is no such food items, display "No expired food items to show." instead.
- **Main Menu Option 5: List food items that are not expired**
    - List all food items with expiry date on or after the current date, ordered by their expiry dates (oldest first). Food items that are expiring on the current date are not expired.
    - If there is no such food items, display "No expired food items to show." instead.
- **Main Menu Option 6: List food items expiring in 7 days**
    - List all food items that has not but will expire in 7 days including the current date, ordered by their expiry dates (oldest first). E.g., food items with expiry date up to and including 8 Sep will be included if the current date is 1 Sep.
    - If there is no such food items, display "No food items expiring in 7 days to show." instead.
- **Main Menu Option 7: Exit**
    - Display the message "Thank you for using the system." or something fun, then exit the application. It will also automatically save the food-items-list in JSON format.

Your text UI need not match the sample exactly, but it should be of equal quality and include all the required information.

Coding Requirements

- Your code must conform to the programming style guide for this course. See course website.
- All classes must have both class-level and method-level JavaDoc comments describing the purpose of the class and methods. Field-level comments are optional.

Suggestions

- Think about the design before you start coding.
    - List the classes you expect to create.
    - For each class, decide what its responsibilities will be.
    - Think through some of the required features. How will each of your classes work to implement this feature? Can you think of design alternatives?
- Write your code in progressing level of details.
    - Your code does not need to be fully functional at the firs time. Start with just printing a message in a method to have the logic correct.
    - Use refractoring to improve your code in later stages.
- You can reuse some of the methods to save time. For example, think about how to reuse the "List all food items" option in the "Remove a food item" option.

Submission Instructions

- Submit a ZIP file of your project to the corresponding folder on Canvas. Name it using this format: **<firstname_lastname>_<studentID>_Assignment1.zip** (e.g., John_Smith_012345678_Assignment1.zip). See course website for directions on creating and testing your ZIP file for submission. **Deviation from that will result in mark deductions**. If you have any difficulties in submitting your file on Canvas, email it to the instructor.
- Email a copy to yourself at your SFU account before the deadline for save keeping.

- If you use any libraries they have to be included in the project as well.
- All submissions will automatically be compared for unexplainable similarities.

Useful Resources

- You MUST use this GSON library for handling JSON: https://github.com/google/gson
  - Read in particular the toJson and fromJson methods
  - Optional: take a look at methods in GsonBuilder to make your json file look nicer
- A video tutorial for File I/O with GSON by Dr. Brian Fraser: https://youtu.be/HSuVtkdej8Q
- Java API of the LocalDateTime class: https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/time/LocalDateTime.html
  Tutorial on formatting numeric print output (look at how DecimalFormat is used to pad zero's): https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Notes

The GSON library does not natively support reading/writing LocalDateTime objects. To make it work you will have to define how the operations are done when creating the Gson object for reading/writing. Here is the code snippet before calling the toJson and fromJson methods from a Gson object.

```
Gson myGson = new GsonBuilder().registerTypeAdapter(LocalDateTime.class,
                new TypeAdapter<LocalDateTime>() {
                  @Override
                  public void write(JsonWriter jsonWriter,
                    LocalDateTime localDateTime) throws IOException {
                      jsonWriter.value(localDateTime.toString());
                  }

                  @Override
                  public LocalDateTime read(JsonReader jsonReader) throws IOException {
                    return LocalDateTime.parse(jsonReader.nextString());
                  }
                }).create();
```

This code constructs a custom Gson object that allows additional configurations not available by default. The same manner can be used to make your json file look nicer (optional). For more details, refer to: https://www.javadoc.io/doc/com.google.code.gson/gson/latest/com.google.gson/com/google/gson/GsonBuilder.html

Sample Input/Output

(first time running the system, system date is 2021-09-19)*

```
#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-19
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
```



```
Choose an option by entering 1-7: 1
No food items to show.

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-19
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 8
Invalid selection. Enter a number between 1 and 7
Choose an option by entering 1-7: 0
Invalid selection. Enter a number between 1 and 7
Choose an option by entering 1-7: 2
Enter the name of the new food item: Peanut butter cookies
Enter the notes of the new food item: Contains nuts
Enter the price of the new food item: 4
Enter the year of the expiry date: 2021
Enter the month of the expiry date (1-12): 9
Enter the day of the expiry date (1-28/29/30/31): 19
Peanut butter cookies has been added to the list.

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-19
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 1

Food Item #1
Name: Peanut butter cookies
Notes: Contains nuts
Price: 4.00
Expiry date: 2021-09-19
This food item will expire today.

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-19
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
```



```
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 7
Thank you for using the system.
```

(Second time running the system, note how the food-items-list is populated with the food item added in the previous run, system time is 2021-09-20)*

```
#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 1

Food Item #1
Name: Peanut butter cookies
Notes: Contains nuts
Price: 4.00
Expiry date: 2021-09-19
This food item is expired for 1 day(s).

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 2
Enter the name of the new food item: Chocolate Cheese cake
Enter the notes of the new food item:
Enter the price of the new food item: 12.45
Enter the year of the expiry date: 2021
Enter the month of the expiry date (1-12): 6
Enter the day of the expiry date (1-28/29/30/31): 31
Error: this date does not exist.
Enter the day of the expiry date (1-28/29/30/31): 30
Chocolate Cheese cake has been added to the list.

#########################
# My Food Items Tracker #
#########################
```



```
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 1

Food Item #1
Name: Chocolate Cheese cake
Notes:
Price: 12.45
Expiry date: 2021-06-30
This food item is expired for 82 day(s).

Food Item #2
Name: Peanut butter cookies
Notes: Contains nuts
Price: 4.00
Expiry date: 2021-09-19
This food item is expired for 1 day(s).

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 3

Food Item #1
Name: Chocolate Cheese cake
Notes:
Price: 12.45
Expiry date: 2021-06-30
This food item is expired for 82 day(s).

Food Item #2
Name: Peanut butter cookies
Notes: Contains nuts
Price: 4.00
Expiry date: 2021-09-19
This food item is expired for 1 day(s).

Enter the item number you want to remove (0 to cancel): 0
#########################
# My Food Items Tracker #
#########################
```



```
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 3

Food Item #1
Name: Chocolate Cheese cake
Notes:
Price: 12.45
Expiry date: 2021-06-30
This food item is expired for 82 day(s).

Food Item #2
Name: Peanut butter cookies
Notes: Contains nuts
Price: 4.00
Expiry date: 2021-09-19
This food item is expired for 1 day(s).

Enter the item number you want to remove (0 to cancel): 2
Peanut butter cookies has been removed from the list.

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 2
Enter the name of the new food item: Salmon sushi roll
Enter the notes of the new food item: Need more wasabi
Enter the price of the new food item: 6.5
Enter the year of the expiry date: 2021
Enter the month of the expiry date (1-12): 9
Enter the day of the expiry date (1-28/29/30/31): 21
Salmon sushi roll has been added to the list.

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
```



```
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 2
Enter the name of the new food item: Mooncake
Enter the notes of the new food item: Serve with tea
Enter the price of the new food item: 14.99
Enter the year of the expiry date: 2021
Enter the month of the expiry date (1-12): 9
Enter the day of the expiry date (1-28/29/30/31): 29
Mooncake has been added to the list.

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 1

Food Item #1
Name: Chocolate Cheese cake
Notes:
Price: 12.45
Expiry date: 2021-06-30
This food item is expired for 82 day(s).

Food Item #2
Name: Salmon sushi roll
Notes: Need more wasabi
Price: 6.50
Expiry date: 2021-09-21
This food item will expire in 1 day(s)

Food Item #3
Name: Mooncake
Notes: Serve with tea
Price: 14.99
Expiry date: 2021-09-29
This food item will expire in 9 day(s).

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
```



```
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 4

Food Item #1
Name: Chocolate Cheese cake
Notes:
Price: 12.45
Expiry date: 2021-06-30
This food item is expired for 82 day(s).

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 5

Food Item #1
Name: Salmon sushi roll
Notes: Need more wasabi
Price: 6.50
Expiry date: 2021-09-21
This food item will expire in 1 day(s).

Food Item #2
Name: Mooncake
Notes: Serve with tea
Price: 14.99
Expiry date: 2021-09-29
This food item will expire in 9 day(s).

#########################
# My Food Items Tracker #
#########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 6

Food Item #1
Name: Salmon sushi roll
Notes: Need more wasabi
Price: 6.50
```



```
Expiry date: 2021-09-21
This food item will expire in 1 day(s).

########################
# My Food Items Tracker #
########################
Today is: 2021-09-20
1: List all food items
2: Add a new food item
3: Remove a food item
4: List expired food items
5: List food items that are not expired
6: List food items expiring in 7 days
7: Exit
Choose an option by entering 1-7: 7
Thank you for using the system.
```

*The sample input/output is for demo purposes only. We'll use different test cases when grading.

Academic Honesty

It is expected that within this course, the highest standards of academic integrity will be maintained, in keeping with SFU's Policy S10.01, "Code of Academic Integrity and Good Conduct." In this class, collaboration is encouraged for in-class exercises and the team components of the assignments, as well as task preparation for group discussions. However, individual work should be completed by the person who submits it. Any work that is independent work of the submitter should be clearly cited to make its source clear. All referenced work in reports and presentations must be appropriately cited, to include websites, as well as figures and graphs in presentations. If there are any questions whatsoever, feel free to contact the course instructor about any possible grey areas.

Some examples of unacceptable behavior:

- Handing in assignments/exercises that are not 100% your own work (in design, implementation, wording, etc.), without a clear/visible citation of the source.
    - Modifying work you copied from elsewhere is not your own work.
- Using another student's work as a template or reference for completing your own work.
- Using any unpermitted resources during an exam.
- Looking at, or attempting to look at, another student's answer during an exam.
- Submitting work that has been submitted before, for any course at any institution.

All instances of academic dishonesty will be dealt with severely and according to SFU policy. This means that Student Services will be notified, and they will record the dishonesty in the student's file. Students are strongly encouraged to review SFU's Code of Academic Integrity and Good Conduct (S10.01) available online at: http://www.sfu.ca/policies/gazette/student/s10-01.html.

Assignment 1: Food Items Tracker (15% of course total) Marking Guide

Must be done individually.

Total = [30] marks. Use the formula Quiz_completed?(Quiz_mark+Program_mark):(0.5*Program_mark).

Quiz_mark

[5] Quiz: Online quiz.

Program_mark has two parts [20]+[5]

0 on Program_mark if cannot compile (communicate this to the student).

[20] Tracker Functionality

- [2] Able to save and load added items for subsequent running of the program
    - No marks for this point if methods other than JSON is used.
- [4] Robust keyboard input: when given an invalid value gracefully asks for retry (need not handle invalid types, like "hi" for a number).
    - Test it for each option/action, -1 for each, max. penalty is losing all 4 marks in this point
- [2+2] Able to add and remove items correctly.
- [2] Able to list expired items correctly.
- [2] Able to list items that are not expired correctly.
- [2] Able to list items that are expiring in 7 days correctly.
- [4] Good listing of items (using toString and show days left until expiration).

[5] Code Quality and Style

- Reasonable object oriented structure
    - must at least 3 non-trivial classes.
- Very minor violations of style guide have no penalty
    - (e.g., having "int myCount=0;" (spacing wrong) once)
- Lose a few marks for consistent problems
    - (like always getting the spacing wrong).
- Larger penalties possible for horrific code (0 on Program_mark)

Some specifics to check with suggested deductions (to Program_mark)

- -5 Incorrect indentation, brackets, or spacing (use IDE's reformat if needed).
- -5 Poor intention revealing class, method/variable names.
    - (e.g., making everything public or static, undecipherable names)
- -3 Poor/missing JavaDoc comment on a class (not needed on fields)
- -3 Uneven menu
- -3 incorrect file names, class names (first letter must be uppercase), file structure (must start from empty project, use OpenJDK16 & Google Gson)

Forward to instructor if material is suspiciously similar to another submission or code posted online.

Java Code Style Guide

Clean & Simple Code

Which option are you?

```
// Option 1:
final String HIGHEST_PRIORITY = "clean code";
System.out.println("My priority: " + HIGHEST_PRIORITY + ".");

// Option 2:
char[]w="\0rd0ct33lri".toCharArray();
for(int x=0xA;w[x]>0;System.out.print(w[x--]));
```

It is always best to write simple, clear code that is easy to understand, debug and maintain.

Use Meaningful Names

All identifiers must have meaningful, human-readable, English names. Avoid cryptic abbreviations such as `dspl()`, `cntStd()`, or `stdRegYYYYMMDD`. Instead use:

```
void display();
int countStudents();
Date dateStudentRegistered;
```

Exception 1: loop variables may be `i`, `j` or `k`. However, prefer the for-each loop when possible.

```
for (int i = 0; i < 10; i++) {
    ...
}
```

Exception 2: variables with very limited scope (<20 lines) may be shortened if the purpose of the variable is clear.

```
void swapCars(Person person1, Person person2)
{
    Car tmp = person1.getCar();
    person1.setCar(person2.getCar());
    person2.setCar(tmp);
}
```

Naming Conventions

Constants must be all upper case, with multiple words separated by '_':

```
final int DAYS_PER_WEEK = 7;
```

Functions must start with a lower case letter, and use CamelCase. Functions should be named in terms of an action:

```
double calculateTax();
boolean verifyInput();
```

Class names must start with an upper case letter, and use CamelCase. Classes should be named in terms of a singular noun:

```
class Student;
class VeryLargeCar;
```

Constant should have the most restrictive scope possible. For example, if it is used in only one class, then define the constant as private to that class. If a constant is needed in multiple classes, make it public.

Generally favour local variables over "more global" variables such as class fields. In almost all languages, global variables are terrible!

Do not use prefixes for variables: don't encode the type (like `iSomeNumber`, or `strName`), do not prefix member variables of a class have with `m_`.

```
class Car {
    private String make;
    private int serialNumber;
    ...
}
```

Boolean variables should be named so that they make sense in an if statement:

```
if (isOpen) {
    ...
}
while (!isEndOfFile && hasMoreData()) {
    ...
}
```

Use named constants instead of literal numbers (magic numbers). It is often acceptable to use 0 and 1; however, it must be clear what they mean:

```
// OK:
int i = 0;
i = i + 1;

// Bad: What are 0 and 1 for?!?
someFunction(x, 0, 1);
```

Indentation and Braces {...}

There's **always** time for **perfect indentation**.

Tab size is 4; indentation size is 4. Use tabs to indent code.

```
if (j < 10) {
→   counter = getStudentCount(lowIndex,
→   →   highIndex);
→   if (x == 0) {
→   →   if (y != 0) {
→   →   →   x = y;→   →   // Insightful comment here
→   →   }
→   }
}
```

Opening brace is at the end of the enclosing statement; closing brace is on its own line, lined up with the start of the opening enclosing statement. Statements inside the block are indented one tab.

```
for (int i = 0; i < 10; i++) {
→   ...
}

while (i > 0) {
→   ...
}

do {
→   ...
} while (x > 1);
```

```
if (y > 500) {
→   ...
} else if (y == 0) {
→   ...
} else {
→   ...
}
```

Exception: `if` statements with multi-line conditions have the starting brace aligned on the left to make it easier to spot the block in the `if` statement.

```
if (someBigBooleanExpression
→   && !someOtherExpression)
{
→   ...
}
```

All `if` statements and loops should include braces around their statements, even if there is only one statement in the body:

```
if (a < 1) {
    a = 1;
} else {
    a *= 2;
}
while (count > 0) {
    count--;
}
```

Your indenation may differ in cases where it improves readability. For example with a multi-line array, complex conditionals or complex calculations.

Statements and Spacing

Declare each variable in its own definition, rather than together (`int i, j`).

```
int *p1;
int p2;
```

Each statement should be on its own line:

```
// Good:
i = j + k;
l = m * 2;

// Bad (what are you hiding?):
if (i == j) l = m * 2;
    cout << "Can ya read this?" << endl;
```

All binary (2 argument) operators (arithmetic, bitwise and assignment) and ternary conditionals (?:) must be surrounded by one space. Commas must have one space after them and none before. Unary operators (!, *, &, - (ex: -1), + (ex: +1), ++, --) have no additional space on either side of the operator.

```
i = 2 + (j * 2) + -1 + k++;
if (i == 0 || j < 0 || !k) {
    arr[i] = i;
}
myObj.someFunction(i, j + 1);
```

Add extra brackets in complex expressions, even if operator precedence will do what you want. The extra brackets increase readability and reduce errors.

```
if ((!isReady && isBooting)
    || (x > 10)
    || (y == 0 && z < (x + 1)))
{
    ...
}
```

However, it is often better to simplify complex expressions by breaking them into multiple sub-expressions that are easier to understand and maintain:

```
boolean isFinishedBooting = (isReady || !isBooting);
boolean hasTimedOut = (x > 10);
boolean isOldFirmware = (y == 0 && z < (x + 1));
if (!isFinishedBooting
    || hasTimedOut
    || isOldFirmware)
{
```

```
    ...
}
```

Classes

Inside a class, the fields must be at the top of the class, followed by the methods.

```
class Pizza {
    private int toppingCount;

    public Pizza() {
        toppingCount = 0;
    }
    public int getToppingCount() {
        return toppingCount;
    }
    ...
}

class Topping {
    private String name;
    ...
    public String getName() {
        return name;
    }
}
```

Enums

Prefer the use of enums over integer constants if you indicating some specific things, such as north/south/east/west directions.

```
class Money {
    public enum Coin {
        PENNY,
        NICKLE,
        DIME,
        QUARTER,
```

```
        LOONIE
    }

    private List coins = ...;
    ...
    public void addCoin(Coin coin) {
        if (coin == Coin.PENNY) {
            System.out.println("Found a penny!");
        }
        coins.add(coin);
    }
}
```

Comments

Comments which are on one line should use the `//` style. Comments which are or a couple lines may use either the `//`, or `/* ... */` style. Comments which are many lines long should use `/* ... */`.

Each class must have a descriptive comment before it describing the general purpose of the class. These comments should be in the JavaDoc format. Recommended format is shown below:

```
/**
 * Student class models the information about a
 * university student. Data includes student number,
 * name, and address. It supports reading in from a
 * file, and writing out to a file.
 */
class Student {
    ...
}
```

Each member function in a class may have a comment describing what it does. However, the name of the function, combined with an understanding of the class, should be enough to tell the user what it does.

Comments should almost always be the line before what they describe, and be placed at the same level of indentation as the code. Only very short comments may appear inline with the code:

```
// Display final confirmation message box.
callSomeFunction(
        0,              // Parent.
        "My App",       // Title
        "test");        // Message
```

Comments vs Functions

Your code should not need many comments. Generally, before writing a comment, consider how you can refactor your code to remove the need to "freshen" it up with a comment.

When you do write a comment, it must describe why something is done, not what it does:

```
// Cast to char to avoid runtime error for
// international characters when running on Windows
if (isAlpha((char)someLetter)) {
    ...
}
```

Other

Either post-increment or pre-increment may be used on its own:

```
i++;
++j;
```

Avoid using goto. When clear, design your loops to not require the use of `break` or `continue`.

All switch statements should include a `default` label. If the `default` case seems impossible, place an `assert false;` in it. Comment any intentional fall-throughs in `switch` statements:

```
switch(buttonChoice) {
case YES:
    // Fall through
case OK:
    System.out.println("It's all good.");
    break;
case CANCEL:
```

```
    System.out.println("It's over!");
    break;
default:
    assert false;
}
```

Use plenty of assertions. Any time you can use an assertion to check that some condition is true which "must" be true, you can catch a bug early in development. It is especially useful to verify function pre-conditions for input arguments or the object's state. Note that you must give the JVM the `–ea` argument (enable assertions) for it to correctly give an error message when an assertion fails.

Never let an assert have a side effect such as `assert i++ > 1;`. This may do what you expect during debugging, but when you build a release version, the asserts are removed. Therefore, the i++ won't happen in the release build.