Lecture 03 + 04: Applying Object-Oriented Programming to a Problem

May 10th - 11th, 2022

The Problem:

- Design a system for durations of time

Object-Oriented Analysis

- Question: what is a duration? / what are we working with?
    - 3 hours
    - 1 week
    - 1 ms
    - 20 minutes
    - 4 hours, 20 minutes, 15 seconds, 2 microseconds
    - 1 year, 3 days
    - 10^9 years

1. What can we do with these things?

    - Convert to another unit of time (3 hrs = 180 minutes) -- to seconds
    - Display a standardized format of time
    - Compare durations: longer vs. shorter vs. the same (3 hrs > 120 minutes)
    - Add durations: 2:00 + 3:15 = 5:15
    - Subtract durations: 5 hours - 3 hours = 2 hours
        - Can we have negative time?
    - Decompose duration into a human friendly format

2. Assumptions

    - Do we allow negative durations?: NO
    - What is the smallest unit of time?: Seconds
    - Lower bound: Yes, 0 seconds
    - Upper bound: Unspecified
    - Distinguishing the same amount of time: 3 minutes vs. 180 seconds
        - Takeaway: Everything can be converted to seconds

Implementation

Interface

```java
/**
 * Represents a duration of time in seconds
 */

public interface Duration extends Comparable<Duration> {
  /**
   * Converts the duration into seconds
   *
   * @return the number of seconds equivalent to this duration
   */
  // no parameters because any sort of formatting is a data representation/implementation issue.
  // we use a long instead of an int to avoid integer overflows
  //  - ints are 32 bit, longs are 64 bit
  long inSeconds();

  /**
   * Formats the duration into HH:MM:SS
   *
   * @return a String of format of this duration in HH:MM:SS
   */
  String asHms();

  /**
   * Adds two Durations together
   *
   * @param other the other Duration to add to this Duration
   * @return the sum of this Duration and the other Duration
   */

  // other return type option: void (mutate this Duration)
  // we go with Duration to avoid mutation, but it's a design decision
  // Is a duration a mutable thing or a constant representation of a duration?
  Duration add(Duration other);

  /**
   * Test if this Duration and the other Object (if a duration) have the same seconds value
   *
   * @param o the Object to compare this Duration to
   * @return true if both objects are durations and have the same seconds value, false otherwise
   */
  boolean equals(Object o);

  /**
   * Returns the hash code for this duration
   * @return an integer representing the hash code
   */
  int hashCode();

  /**
   * Compares two durations
   *
   * @param d the object to be compared
   * @return &lt;0 if this &lt; that in terms of seconds <br>
   *         =0 if this == that in terms of seconds <br>
   *         &gt;0 if this &gt; that in terms of seconds
   */
  int compareTo(Duration d);
}
```

Design/Implement a Class Implementing the Interface

Data Representation

- How do we keep track of time?
    - Seconds only
    - Hours, minutes, and seconds
    - Minutes but as doubles

Implementation in Code

1. First implementation: HMSDuration
    - HMSDuration
2. Second implementation: HMSDuration and CompactDuration
    - Develop an *abstract* class - `AbstractDuration`
    - New access modifier - `protected`
        - Can be accessed from: any subclass or the same package
    - New Files:
        - AbstractDuration
        - CompactDuration
    - Abstract Classes abstract behavior across multiple child classes
    - Factory Methods
        - Primarily creates objects
        - Can create objects and return objects of several related types
        - The object returned is determined at runtime (dynamic dispatch)
3. The Factory Pattern □
    - `static` - method/variables/classes that are part of a given Class
        - Usage: `<ClassName>.<Static-Method-Name>`
    - DurationCreator
    - Make all other constructors `protected`, you can only create new objects using the `DurationCreator` class
    - Other note: `enum`s - enumerative data types

Testing our Class

- We use JUnit