# Outline

A solution to the GoGundMe recurring donations command-line problem. This solution is implemented in Java and follows the requirements mentioned in the description.

# Solution Overview

The solution to this problem is implemented using Java’s standard libraries. The code is structured in a way that the related data and behavior is organized together.

## Data Model

Each real-world object is considered as an Entity, so we will have three entities in this problem. They are 1) Donor, 2) Campaign, 3) Donation. Each entity is a class that extends from an AbstractEntity class.

## Execution

The entry-point into the application for the outside world is from Main class, this class contains the main() method. However Main is essentially a pass-through which will forward the input to the logical entry-point of the application

An executor named GfmDonationsExecutor is the logical starting point for the application. It receives the input from Main and performs three actions on it. Actions it performs are 1) Use the parser to read the input lines 2) Use the InputEntitiesExecutor to process the data 3) Use the InputEntitiesExecutor to print the data

Reason for architecting this way is, this will divide the responsibility among different logical units with a definite line between them. The Parser will parse the input data and construct a logical payload, the InputEntitiesExecutor will use the logical payload to perform donations while adhering to requirements and GfmDonationsExecutor acts as a co-ordinator between the two.

## Parser

Its worth digging down on how the Parser actually parses the data. Parser will find out if the input is in a file or coming directly from cmd-prompt. Then it parses individual input lines, basing on the first word, it figures what kind of entity to create, and uses a builder to create these entities.

Why use a separate Builder? Different entities need different type and different number of arguments. So builder can do it efficiently while keeping the code clean.

## Entities

As stated above, we have three entities for this problem, Usually in production, these entities will be stored in a database. All these entities inherit from an abstract base entity. The advantage of this architecture is, 1) we can add all the common logic (ex: load, save, validate etc) into the base entity 2) we will get a common handle to process these entities.

Donor and Campaign entities are slightly “overloaded” because they have logical methods as well apart from getters. Usually in production applications, a child class is created for these Entities, child classes will contain the logical methods. But that seem like an overkill for this application.

## Code organization

Find the packages and a brief description below

| 1. com.gofundme.recurringdonations.entities - Contains class files for all the entities, entityBase class and entityType enum  2. com.gofundme.recurringdonations.executors - Contains the class files that perform donations, prints results and co-ordinates among all logical entities  3. com.gofundme.recurringdonations.inputparser - Contains class files used for parsing the input  4. com.gofundme.recurringdonations.tests - Contains class files for Junit tests and txt files used by tests. |
| --- |
| PS: The code is default-formatted by eclipse |
|  |

# Build and Run

## Prerequisites

* Unix/Linux or Mac
* Java 11 or higher

## Steps to run directly from executable

1. Find the Jar and the executable attached
2. Create an input.txt file in the same directory where the executable is in
3. To run with input.txt file as input

| chmod +x gfm-recurring  ./gfm-recurring input.txt |
| --- |

1. To run with commands directly as inputs

| cat input.txt | ./gfm-recurring |
| --- |

## Steps to build

1. Download the attached source code
2. Create a runnable jar file

| *jar cf jar-file path\_to\_root\_directory* |
| --- |

Or easier way is to do Eclipse->select\_src->right-click-export->Runnable JAR file

1. Convert the JAR file to an executable

| echo '#!/usr/bin/java -jar' > gfm-recurring cat gfm-runnable.jar >> gfm-recurring |
| --- |

1. Then use the steps in previous section to run

## JUnit tests

1. The submission includes Junit tests to test individual units in the solution.
2. The tests will 1) test the parser and 2) test the logic of calculating donations.
3. The parser tests directory also contain two sample input.txt files
4. To run the JUnit tests: The easiest way is to open in eclipse->Select the test file->Run As->Unit test.  
   Follow commands below to run from command line

| java -cp .gfm-runnable.jar org.junit.runner.JUnitCore InputLineExecutorTests java -cp .gfm-runnable.jar org.junit.runner.JUnitCore InputParserTests |
| --- |

## Sample runs

1. Input file as the input parameter

| kiran@kirans-air Desktop % ./gfm-recurring input.txt  Donors: Greg: Total: $300 Average: $150.0 Janine: Total: $50 Average: $50.0  Campaigns: HelpTheKids: Total: $200 SaveTheDogs: Total: $150 kiran@kirans-air Desktop % |
| --- |

1. Commands directly in the command line

| kiran@kirans-air Desktop % cat input.txt | ./gfm-recurring  Donors: Greg: Total: $300 Average: $150.0 Janine: Total: $50 Average: $50.0  Campaigns: HelpTheKids: Total: $200 SaveTheDogs: Total: $150 kiran@kirans-air Desktop % |
| --- |