SDV503 Assessment One

Console Calendar App

Software Design Document

Rhylei Tremlett
April 30, 2022

Table of Contents

Object-Orientated Programming

Object-orientated programming (OOP) is programming paradigm that focuses on objects and classes. OOP has four fundamental concepts known as the four pillars, these pillars are Encapsulation, Abstraction, Inheritance and Polymorphism.

Encapsulation

Encapsulation is the bundling of data and methods into a single component, encapsulation also hides information from other components in the code by restricting access to only methods within the object.

Abstraction

Abstraction aims to hide details about implementation and focus only on what is essential to operation. Abstraction is also a way of simplifying code and increasing maintainability by reducing the impact of change.

Inheritance

Inheritance is one of the most noteworthy features of classes, a class can define generic attributes and properties that will be passed on to any object created by the class. This makes code modular and reduces repetition in your code.

Polymorphism

Polymorphism is a convenient tool, which is closely related to inheritance, when we have a method that is applicable to multiple different objects, we can just give them all the same method.

Below is an example of how a piece of procedural code might be converted to OOP.

Figure 1, a simple program that displays the info about a dog.

```
let dogName = "Dan";
let dogColour = "Brown and Black";
let dogBreed = "German Shepard";
let dogAge = 10;

function dogInfo(name, colour, breed, age) {
  return `My dog's name is ${name}, it is a ${colour} ${breed}, it is ${age}
  years old.`;
}

console.log(dogInfo(dogName, dogColour, dogBreed, dogAge));
```

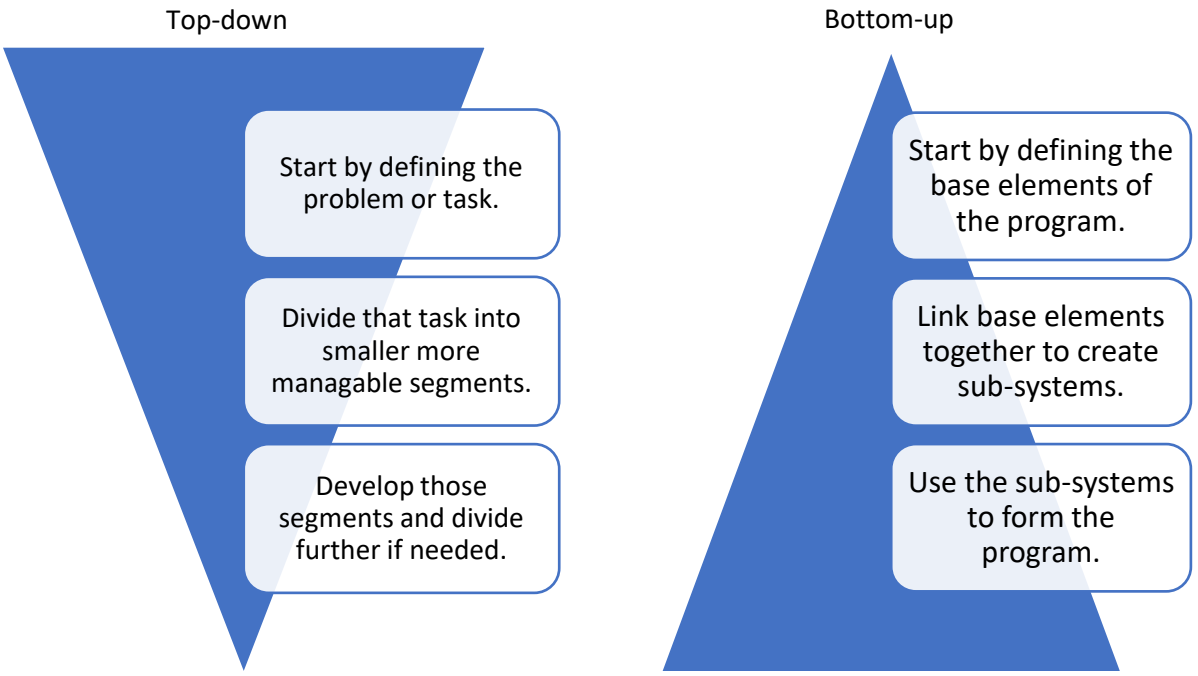
Figure 2, The same as Figure 1, but now rewritten to use an Object-orientated approach.

```
class Dog {
  constructor(name, colour, breed, age) {
    this.name = name;
    this.colour = colour;
    this.breed = breed;
    this.age = age;
  }

  info() {
    return `My dog's name is ${this.name}, it is a ${this.colour}
    ${this.breed}, it is ${this.age} years old.`;
  }
}

const myDog = new Dog("Dan", "Brown and Black", "German Shepard", 10);
console.log(myDog.info());
```

What is the top-down model?

It is a design approach or a programming style in which the largest and most complex pieces are defined and then broken down into smaller more manageable pieces. Whereas in a bottom-up approach all the smaller components are defined and incorporated until you have a large and complex working program.

Top-down vs Bottom-up



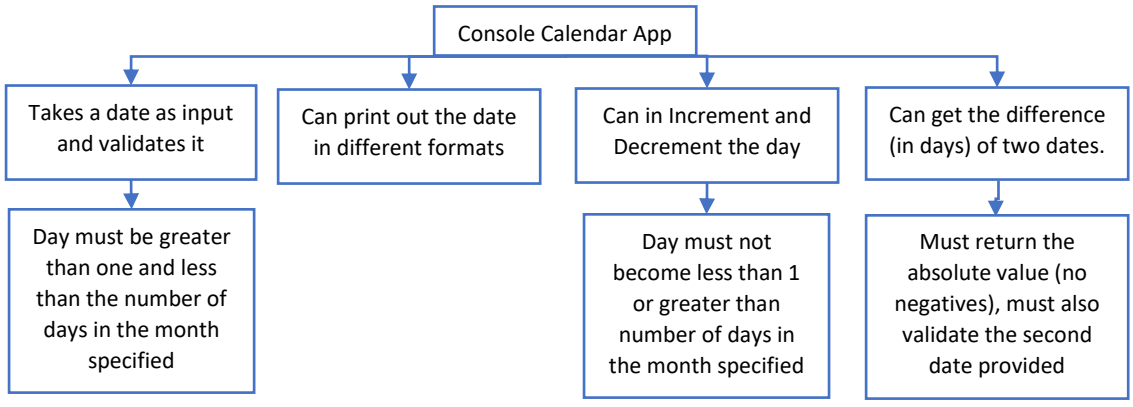
Advantages of Top-down:

- Helps to identify where to begin.
- Creating components becomes easier as you develop more of them.
- Makes it easier for teams to work on a project.

Advantages of Bottom-up:

- Works well when there is existing code that can be reused.
- Specific functionality is defined at the beginning.

Top-Down Model



Requirement Analysis

Requirement Analysis is the process of identifying what the program needs to do or be able to manage as well as specify any specific methods that need to be used or styles that need to be followed.

Requirement One

The first set of requirements specified in the assessment outline are as follows.

1. Design a class called Date.
2. The class should store a date in three integers: month, day, and year.
3. There should be member functions to print the date in the following formats: 12/25/10, December 25, 2010, 25 December 2010.
4. Input Validation: Do not accept values for the day greater than 31 or less than 1. Do not accept values for the month greater than 12 or less than 1.

Requirement Two

The Date class version should have the following overloaded operators:

1. Prefix and postfix increment operators. These operators should increment the object's day member.
2. Prefix and postfix decrement operators. These operators should decrement the object's day member.
3. Subtraction operator. If one Date object is subtracted from another, the operator should give the number of days between the two dates. For example, if April 10, 2010, is subtracted from April 18, 2010, the result will be 8.

The specifics for the increment and decrement methods are specified below in requirement three.

Requirement Three

Demonstrate the class's capabilities in a program using JavaScript. The class should detect the following conditions and handle them accordingly:

1. When a date is set to the last day of the month and incremented, it should become the first day of the following month.
2. When a date is set to December 31 and incremented, it should become January 1 of the following year.
3. When a day is set to the first day of the month and decremented, it should become the last day of the previous month.
4. When a date is set to January 1 and decremented, it should become December 31 of the previous year.

System Design

Input validation

Input validation is when a program can identify whether the provided input is acceptable or not. Invalid inputs are quite varying, in a program that expects a two-digit number to be provided you could receive a string of random characters, a Boolean or even Undefined or NaN as the input.

By implementing proper input validation, we can sort out all values that do not accept the criteria we set. This means that our program can function without running into errors due to things like performing math on a string value.

Below I will explain the various kinds of inputs and then provide a list of examples.

Valid cases

In my program the only valid inputs are strings that contain a date that can be converted by JavaScript's Date object constructor.

- 05/23/2022
- 05/23/22
- 05 23 22
- 05-23-22
- May/23/2022
- 23 May 2022

Edge cases

Edge cases occur on either side of a boundary, the edges of the boundary are the highest and lowest acceptable values as well as the values that are just out of the range. I will refer to the acceptable edge cases as inner edge and the non-acceptable edge cases as the outer edge.

Inner edge cases:

- 1-1-100
- 12-31-199999

Outer edge cases:

- 0-1-100
- 1-0-100
- 1-1-99
- 13-31-199999
- 12-32-199999
- 12-31-200000

Invalid cases

An invalid case is any input beyond the outer edge case or any input that is of the wrong format or data type.

- 1st of January 2022
- 15/06/2022
- 15, 06, 2022 (as separate values)
- True
- Words, words, words…

- Undefined
- NaN

Expected Outcome vs Actual Outcome

Expected outcomes is what we would expect to be output by the program, actual outcome is what was output by the program.

Valid Inputs	Expected Outcome	Actual Outcome
05/23/2022	5/23/2022	5/23/2022
05/23/22	5/23/2022	5/23/2022
05 23 22	5/23/2022	5/23/2022
05-23-22	5/23/2022	5/23/2022
May/23/2022	5/23/2022	5/23/2022
23 May 2022	5/23/2022	5/23/2022
Inner Edges	Expected Outcome	Actual Outcome
1-1-100	1/1/100	1/1/100
9-13-275760	9-13-275760	9-13-275760
Outer Edges	Expected Outcome	Actual Outcome
0-1-100	Error, invalid input	Error, invalid input
1-0-100	Error, invalid input	Error, invalid input
1-1-99	Error, invalid input	Error, invalid input
10-13-275760	Error, invalid input	Error, invalid input
9-14-275760	Error, invalid input	Error, invalid input
9-13-275761	Error, invalid input	Error, invalid input
Invalid Inputs	Expected Outcome	Actual Outcome
1st of January 2022	Error, invalid input	Error, invalid input
15/06/2022	Error, invalid input	Error, invalid input
[15, 06, 2022]	Error, invalid input	Error, invalid input
{'Day':15, 'Month':6, 'Year':2022}	Error, invalid input	Error, invalid input
Words, words, words…	Error, invalid input	Error, invalid input
Undefined	Error, invalid input	Error, invalid input
NaN	Error, invalid input	Error, invalid input

Pseudo code

Pseudo code is a tool that we use to plan out our code and assess our algorithms, it also allows other people to better understand our program without having to read all our code. Pseudo code can be written in whatever language you speak, because there is no syntax required for pseudo code. Pseudo code must follow a logical structure and is a useful tool to evaluate the algorithms we produce in our heads.

Constructor

Use the Date() constructor to create a new valid date.

IF the Date() constructor returns “invalid date” then tell the user that their input was invalid.

Split the new date into separate day, month, and year properties.

PrintDates

LOG Month/Day/Year.

LOG Month(as text) Day, Year.

LOG Day Month(as text) Year.

NOTE: since the getMonth() returns 0 - 11, I will create an array of month names and use that to print the months as text.

Increment

Use the setDate() operator to set the date equal to one higher than the current day property.

Then update the month and year variables according to the new date.

Decrement

Use the setDate() operator to set the date equal to one lower than the current day property.

Then update the month and year variables according to the new date.

Subtract

IF the date to compare is invalid then tell the user.

ELSE return the absolute value of the new date and the current date property.

Legend:

Green is for statements

Blue is for variables

Orange is for methods.