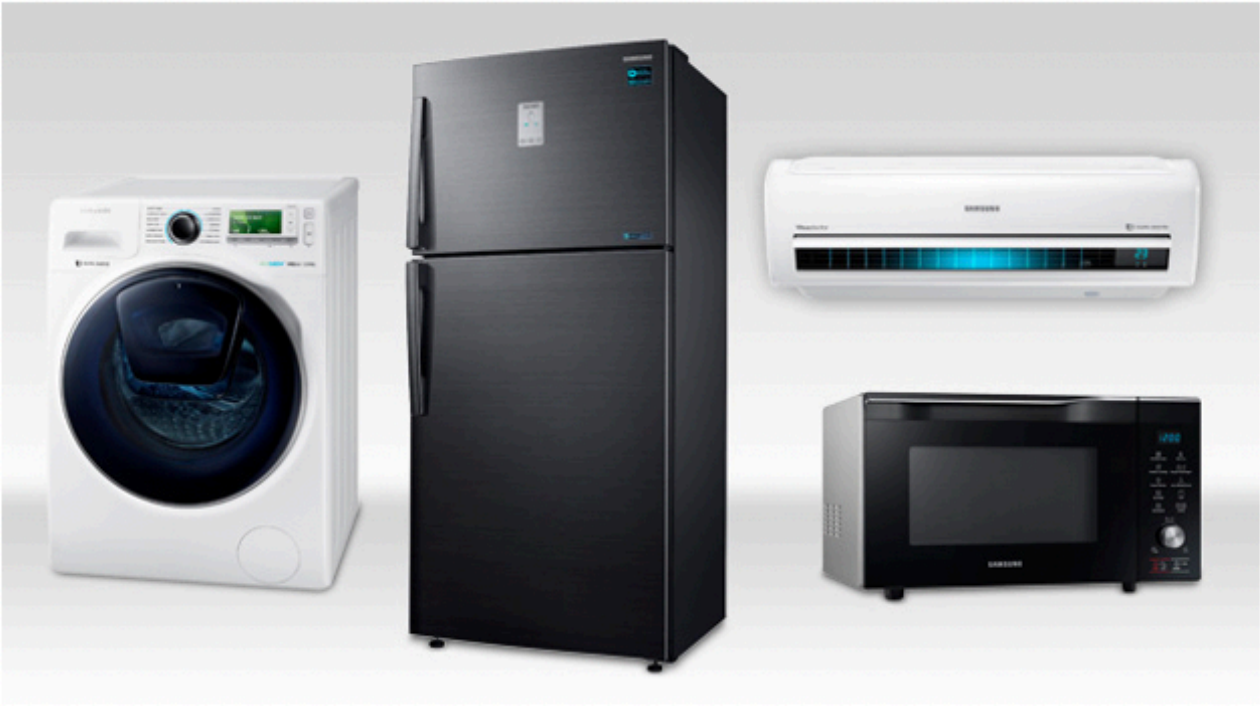
ICS3U/B

Object-Oriented Programming

SKYLINE PROJECT



Overview

This project will explore Processing as well as the use of objects and classes in order to develop an animated/interactive household appliance. Students will strive to break down a scenario, use the debugger and explore how their code works to solve various problems.

Key Deliverables

Processing Program (e.g. AdamDrenthMicrowave.zip)
Includes: • Zipped folder – [Name][Appliance].zip e.g. AdamDrenthMicrowave.zip • Headers for all files • Headers for all appropriate methods • Comments outlining steps • Descriptive variables and methods

Expectations

1. demonstrate the ability to use different data types, including one-dimensional arrays, in computer programs;
2. use proper code maintenance techniques and conventions when creating computer programs;
3. use a variety of problem-solving strategies to solve different types of problems independently;
4. Demonstrate an understanding of the software development process.

Assignment

A basic project (Level 3) *must* include a digital representation of the appliance that changes with some key press or mouse click. At least one class will be written. The program will also include some static (unchanging) elements.

A more advanced project (Level 4) will meet all the basic requirements and may include animation, varied types of interaction (mouse/keyboard), examples of the appliance having a state (e.g. must turn on first or close door), or other additional features.

Appliance examples include toaster, coffee maker, microwave, fridge, clothes washer

The Code

A basic program must have two or more pde files with additional methods and constructors that are used by the program. Students will strive to use object-oriented concepts like:

- Encapsulation
- Single Responsibility Principle
- KISS principle: Keep it simple, silly
- DRY principle: Don't repeat yourself

Commenting

Commenting is an important part of any program and you should always use it. When done well and when it is used for planning purposes, it creates 'clean code' that is easy to develop and change.

Good commenting includes:

- A header for every file and class detailing the purpose, creator/author and sources
- A header for every method that is **not** a constructor, getter or setter
- Descriptive variables and method names, as these create documentation within the code
- Comments for every major step within a large method

Method headers should address any return values, as well as pre- and post-conditions where appropriate. Use JavaDoc formatting.