UNIVERSITÉ DU LUXEMBOURG

Faculty of Science, Technology and Medicine

Device Power Management System

Programming 1

Student ID: 0232378323
Mohamed Aymen Trabelsi

Table of Contents:

1. Abstract

1.1 Project Overview

A program called the Device Power Management System that enables users to control the power level of different electronic gadgets. With a particular emphasis on power management, the system attempts to replicate the actual behaviours of 4 gadgets: laptops, phones, tablets, and smartwatches. We also have 2 more devices that extend: Gaminglaptop and Smartphone.
To add devices, see details, switch power-saving modes, drain batteries, and charge gadgets, users can engage with the system. Depending on variables like power-saving modes and device-specific behaviours (such as intensive apps for tablets), each type of device performs differently. This project aims to create an extendable and interactive system that simulates the use of electronic gadgets in the real world while showcasing important object-oriented programming (OOP) concepts like inheritance, polymorphism, and encapsulation.

1.2 Project Goal

This project's primary objective is to develop a functional system that mimics how different gadgets manage their power. Adding devices, controlling battery levels, switching between power-saving modes, and charging should all be supported by the system. Users should have a good grasp of the system's architecture and be able to interact with a variety of devices, each of which has unique behaviours, at the project's conclusion. Because of the design's emphasis on scalability and flexibility, future additions of new devices will be simple. Through thoughtful design decisions and the application of numerous Java concepts, the project will exhibit strong OOP principles.

2.1.1 Device (Abstract Class)

Purpose: This class is the base for all device types and contains common properties and methods shared across devices.
Attributes:
name: The name of the device (String).
batteryLevel: The current battery level of the device (int, between 0 and 100).
isCharging: A boolean indicating if the device is currently charging.
powerSavingMode: A boolean indicating if the power-saving mode is activated.
Methods:
useBattery(): A method to simulate battery usage, which is overridden by each subclass to define specific behavior.
chargeBattery(int amount): Method to charge the battery by a specified amount.
togglePowerSavingMode(): Toggles the power-saving mode on or off.

2.1.3 Subclasses of Device

Laptop:

Purpose: A specific implementation of the Device class for laptops. This subclass overrides useBattery() to model how the battery drains based on different factors (docking station connection).
Attributes:
isDocked: A boolean indicating if the laptop is connected to a docking station.
Methods:
useBattery(): Drains battery based on whether the laptop is docked and whether power-saving mode is active.

Phone:

Purpose: A specific implementation of the Device class for phones. This subclass overrides useBattery() to handle specific battery drain behaviors: low power mode.**Attributes**:
isLowPowerMode: A boolean indicating if the phone is in low power mode.
Methods:
useBattery(): Drains battery based on whether low power mode is active and adjusts accordingly.

Tablets:

Purpose: A specific implementation of the Device class for tablets. This subclass overrides useBattery() to simulate battery usage depending on whether the tablet is running intensive apps.
Attributes:
isUsingIntensiveApps: A boolean indicating if the tablet is running power-intensive applications.
Methods:
useBattery(): Drains battery faster when intensive apps are running, and slower if power-saving mode is enabled.

Smartwatches

Purpose: A specific implementation of the Device class for smartwatches. This subclass also overrides useBattery() to simulate how a smartwatch drains its battery.
Attributes:
fitnessMode: A boolean indicating if fitness mode is in use.
Methods:
useBattery(): Drains battery faster when fitness mode is enabled, and slower when in power-saving mode.

2.2 Relationships Between Classes

2.2.1 Subtyping

- **Device -> Laptop, Phone, Tablet, Smartwatch**: The Device class serves as an abstract base class, and Laptop, Phone, Tablet, and Smartwatch are concrete subclasses. This

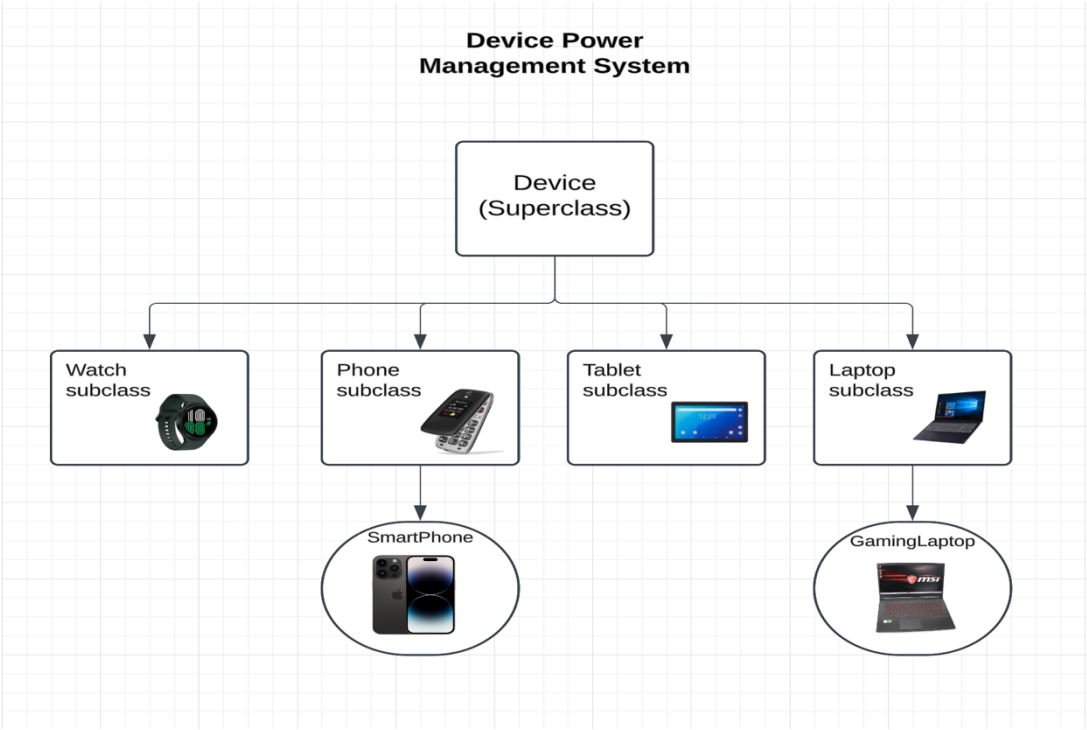
relationship is an example of subtyping, where the subclasses inherit the properties and methods of the Device class and extend them with their own specific behaviors (battery drain based on usage).

- **Laptop -> Gaming Laptop**
- **Phone -> Smartphone**

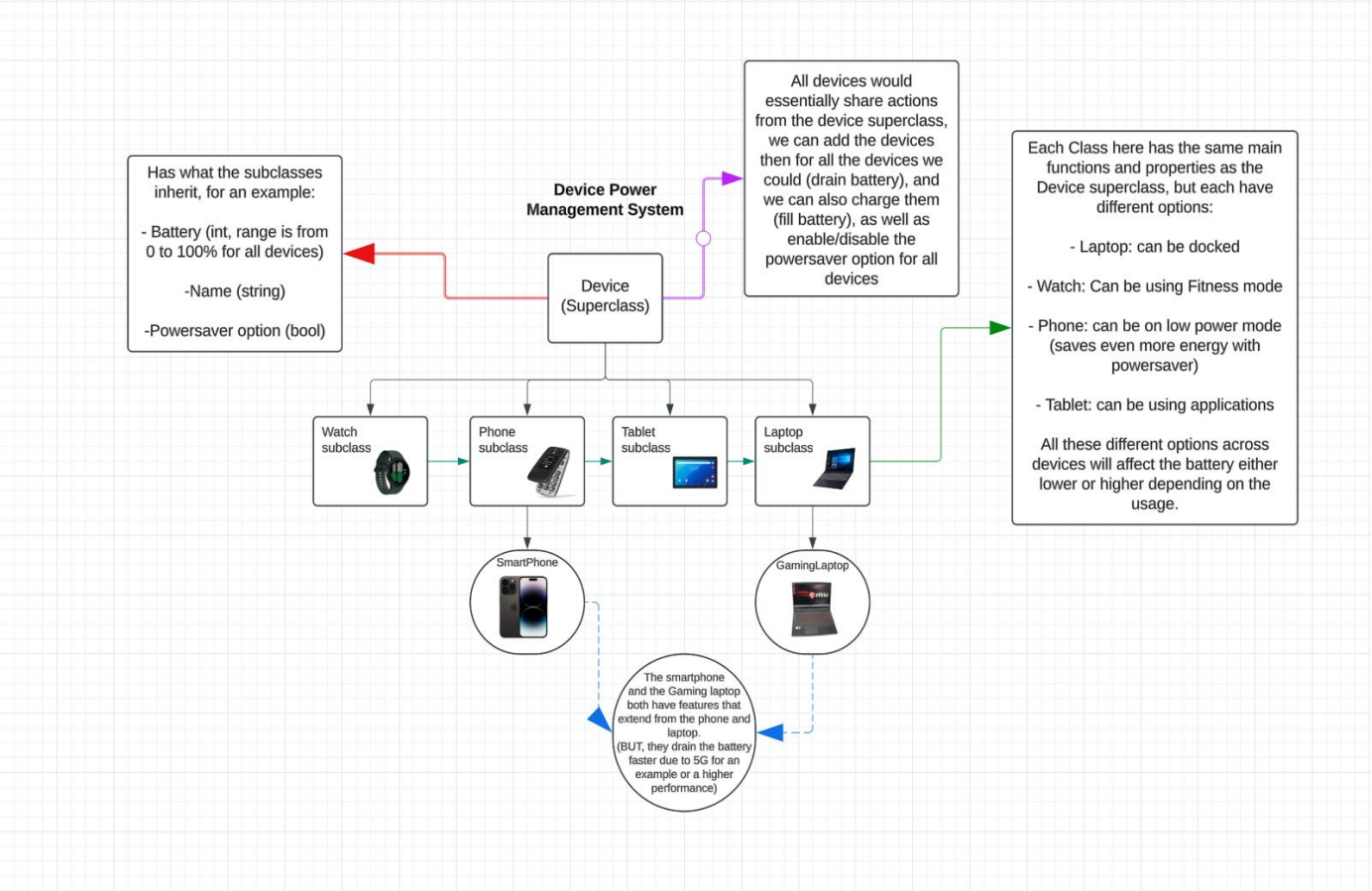
The aboce classes also extend to their daughter classes, the class Smartphone and the class Gaminglaptop share certain properties and methods. This shows us that a class can also be a subclass and a superclass at the same time depending on the inheritence situation.

2.3 Diagrams

This diagram shows the Superclass and the subtyping between the different devices:



This graph explains the relationship between the classes in specific detail:



3. Reasoning Behind OOP Design

3.1 Subtyping vs. Composition

The decision to use subtyping (inheritance) for this design is based on the need for different types of devices (Laptop, Phone, Tablet, Smartwatch) to share common functionality while still allowing for device-specific behaviors. Each device shares a common set of attributes and methods (such as battery level, charging status, and power-saving mode), but each also has its unique way of draining battery. Therefore, inheritance allows for code reuse, reducing redundancy, and provides a clean way to extend functionality.

Composition, in contrast, would have been more appropriate if devices had multiple distinct components (CPU, screen, and battery), which is not my goal for this project. We can perhaps introduce GPU, Camera and Heartsensor to the project and then combine the use og subtyping and composition.

3.2 Polymorphism

The use of polymorphism in this project will ensure that new devices can be added in the future without requiring significant changes to the existing code. New device types can be added by simply extending the Device class and overriding.

Example for the project:
When you loop through the devices and call useBattery():
Each object (Laptop, Phone, Smartwatch) responds differently:

- Laptop: Checks if it’s docked before applying battery drain logic.
- Phone: Adjusts battery usage based on "low power mode."
- Smartwatch: Modifies the drain rate depending on "fitness mode."

This is polymorphism since the same method call (useBattery) produces different results depending on the object's class.

4. Unique Features

4.1 Error Handling

The application includes robust error handling to manage invalid user inputs gracefully. For example:

- If a user enters a battery level outside the range of 0-100, they are prompted to re-enter a valid value. Or if the user enters “zero” the program will ask them to re-enter an integer value.
- Non-numeric or invalid boolean inputs (e.g., "yes" instead of true or false) are detected and the user is guided to provide appropriate input.
  This ensures the program remains user-friendly and avoids crashes due to invalid data.

4.2 Data Persistence

To enhance user experience, the program will support data persistence. This feature enables the application to save the state of all devices (their properties such as name, battery level, charging status, and power-saving mode) and restore it when the program is relaunched. This way users do not lose progress between sessions.