COMP2511

WEEK 9

Would you rather have 4 or 7 eyes?

ADMIN STUFF

- Assignment-ii due TODAY 5pm
- Assignment-iii 8%, no penalty until Week 11 Tuesday
- Week 10 Sample Exam

A G E N D A

- Finding Patterns
  - Iterator Pattern
  - Command Pattern
  - Template Pattern
- Code and Design Smells
- Visitor Pattern

Finding Patterns

FINDING PATTERNS

Sorting collections of records in different orders.

FINDING PATTERNS

Strategy Pattern

This what Java does with the Collections.sort() method. A Comparator can be provided to determine the order in which elements are sorted.

FINDING PATTERNS

Listing the contents of a file system.

FINDING PATTERNS

Composite pattern

Both folders and files are filesystem entries. Files form the leaves, folders can contain files or other folders.

FINDING PATTERNS

Traversing through a linked list without knowing the underlying representation.

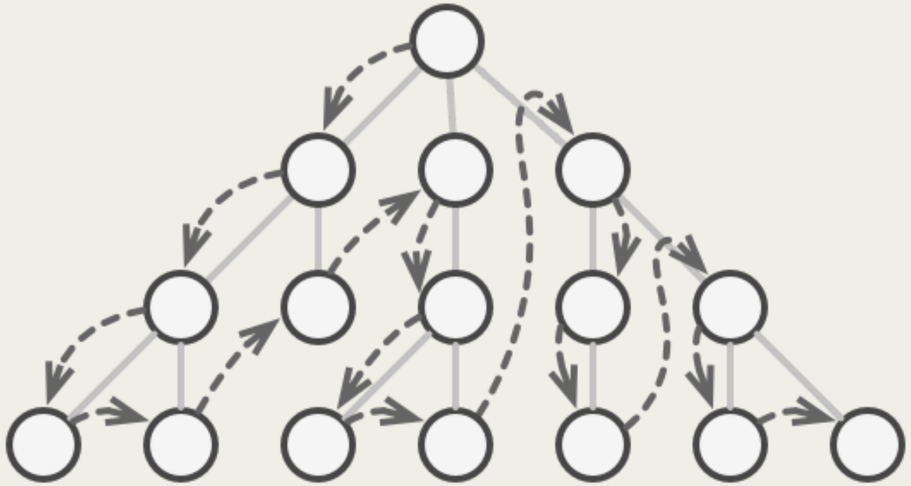
FINDING PATTERNS

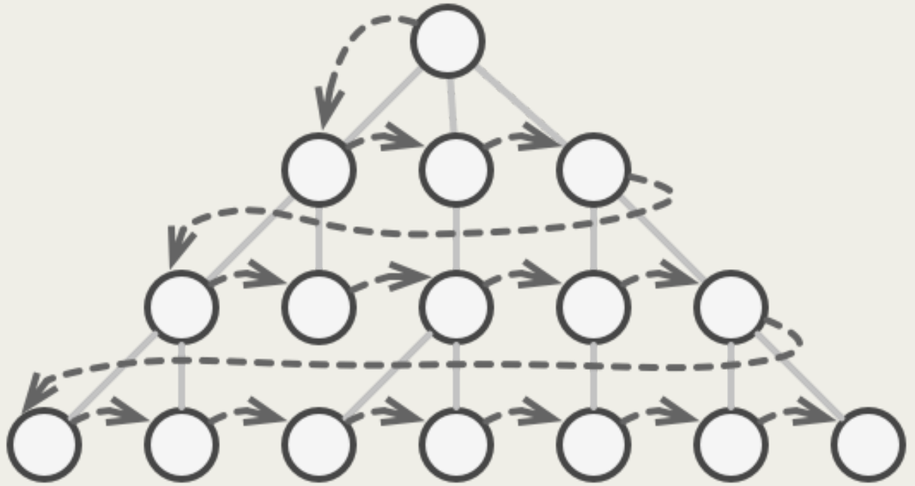
Iterator pattern

An iterator can be written and the collection can be traversed using iterator.next().

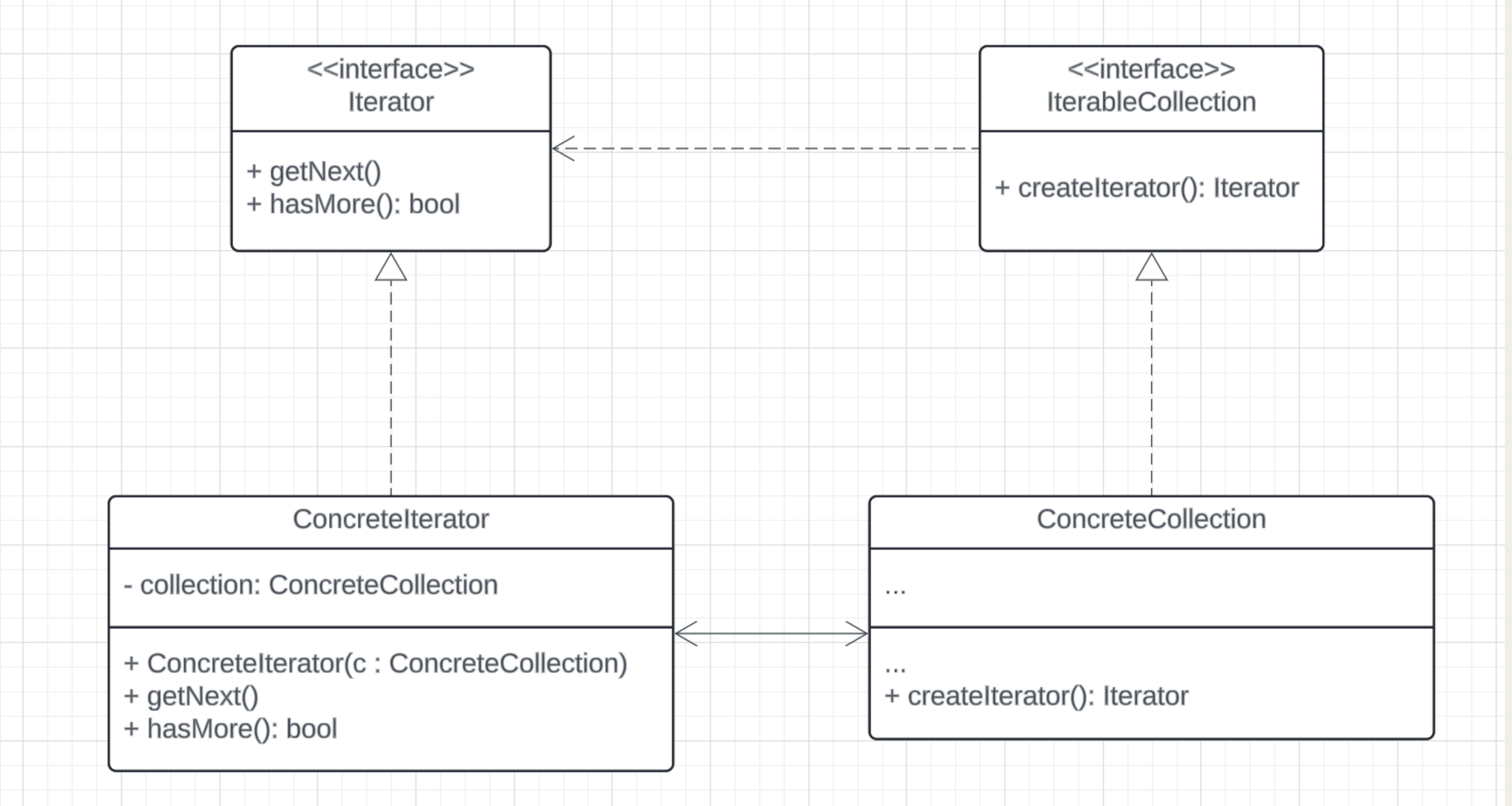
ITERATOR PATTERN

Iterator is a behavioral design pattern that lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree, etc.).





ITERATOR PATTERN



FINDING PATTERNS

Updating a UI component when the state of a program changes.

FINDING PATTERNS

Observer pattern

If the state of the program is the subject and the UI an observer, the UI will be notified of any changes to the state and update accordingly.

FINDING PATTERNS

Parsing and evaluating arithmetic expressions. Allowing users to remap their movement controls to different buttons on a game controller.

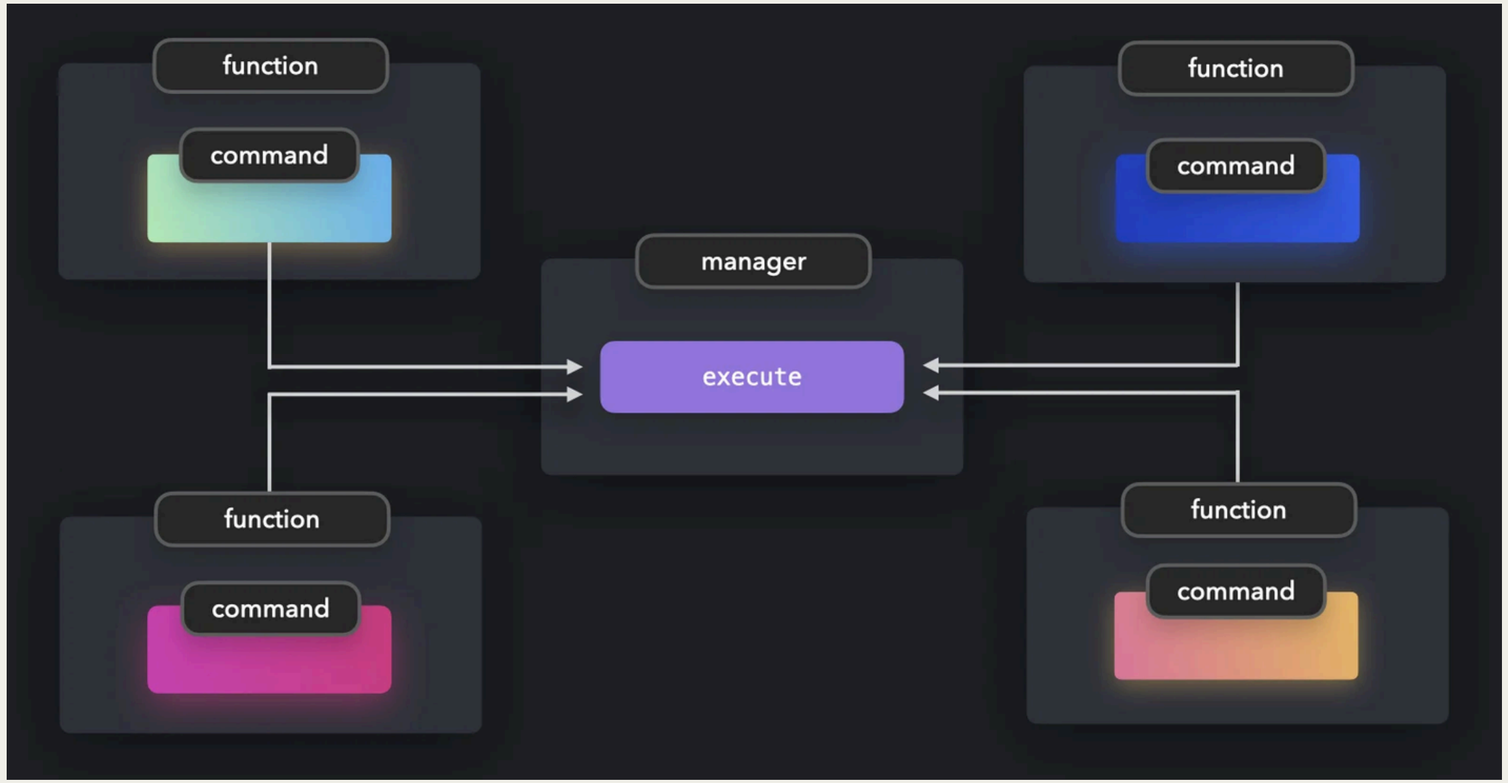
FINDING PATTERNS

Command pattern

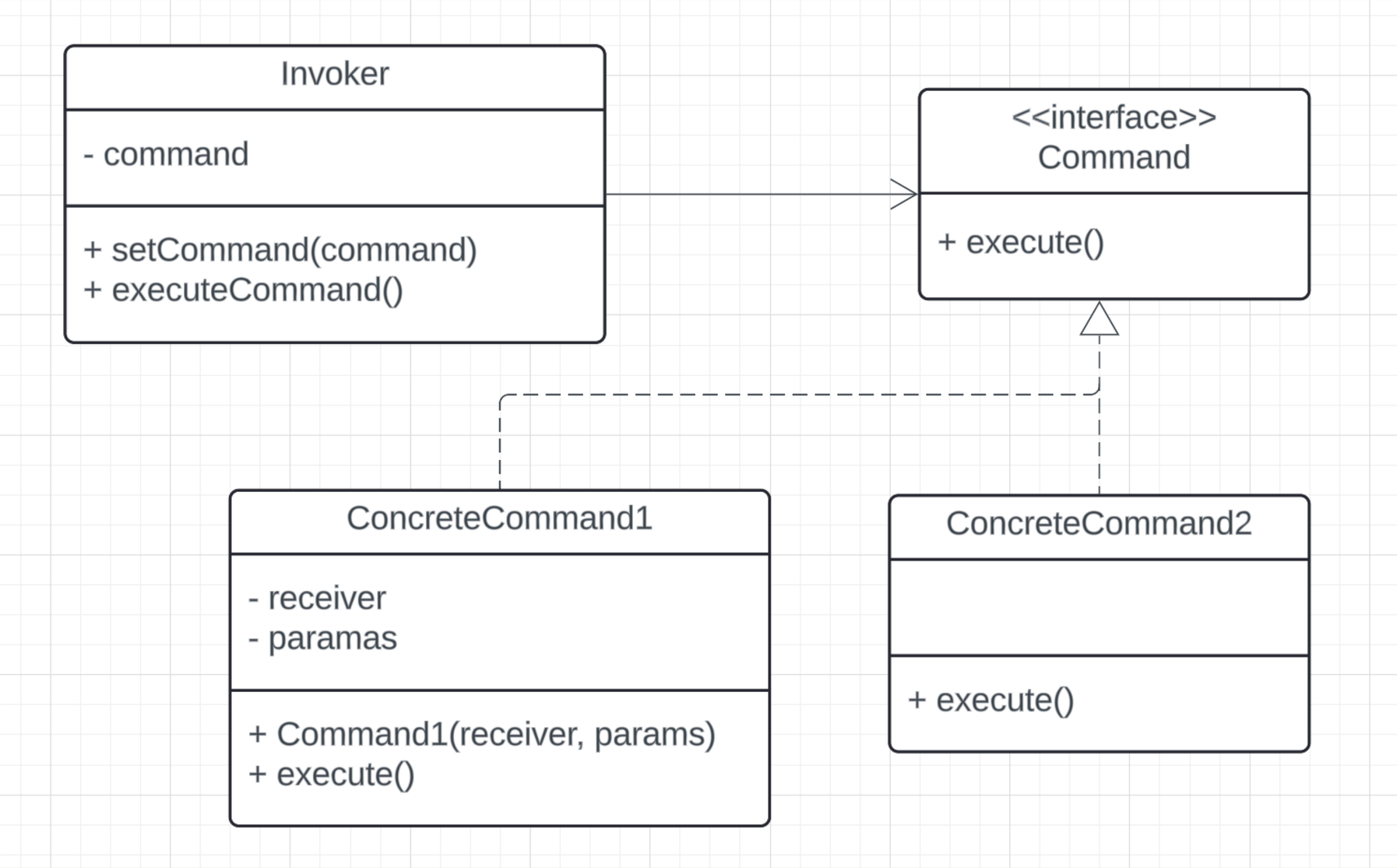
Each action (e.g. "jumping") needs to be decoupled from the source of input into its own object.

COMMAND PATTERN

Command is a behavioral design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request’s execution, and support undoable operations.



COMMAND PATTERN



FINDING PATTERNS

Creating a skeleton implementation for a payment processing algorithm that varies in areas depending on the type (e.g. card vs PayPal)

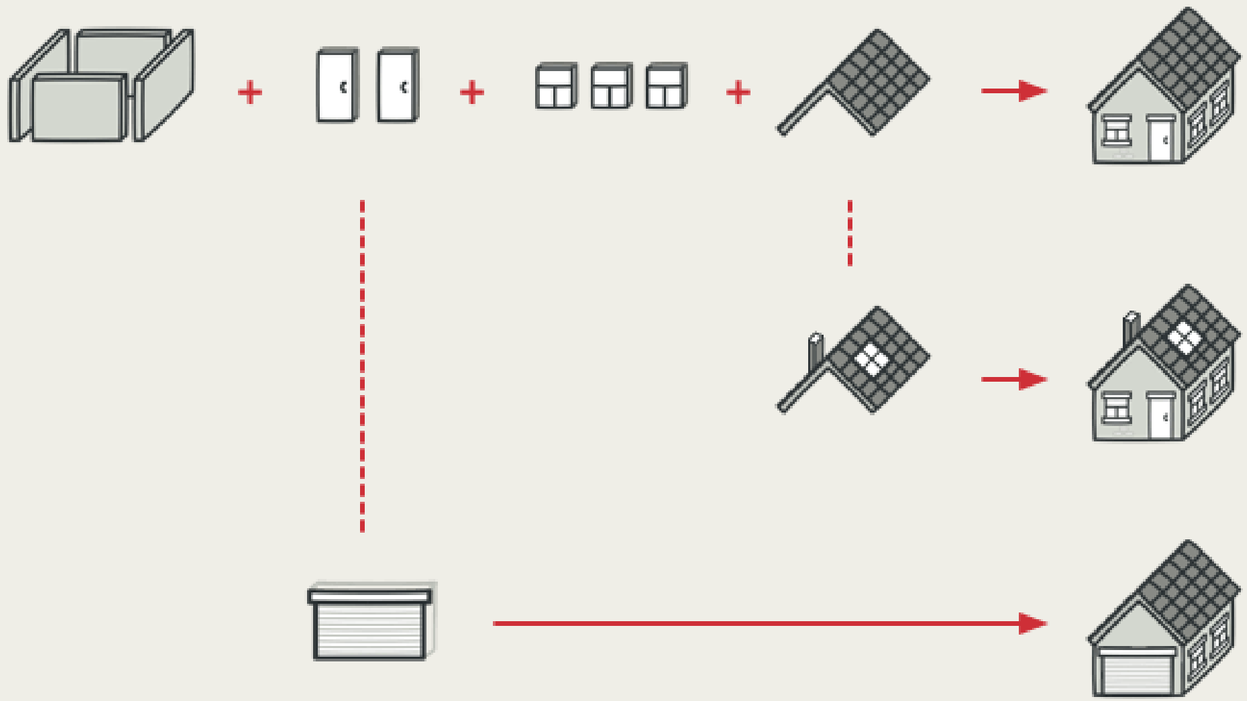
FINDING PATTERNS

Template pattern

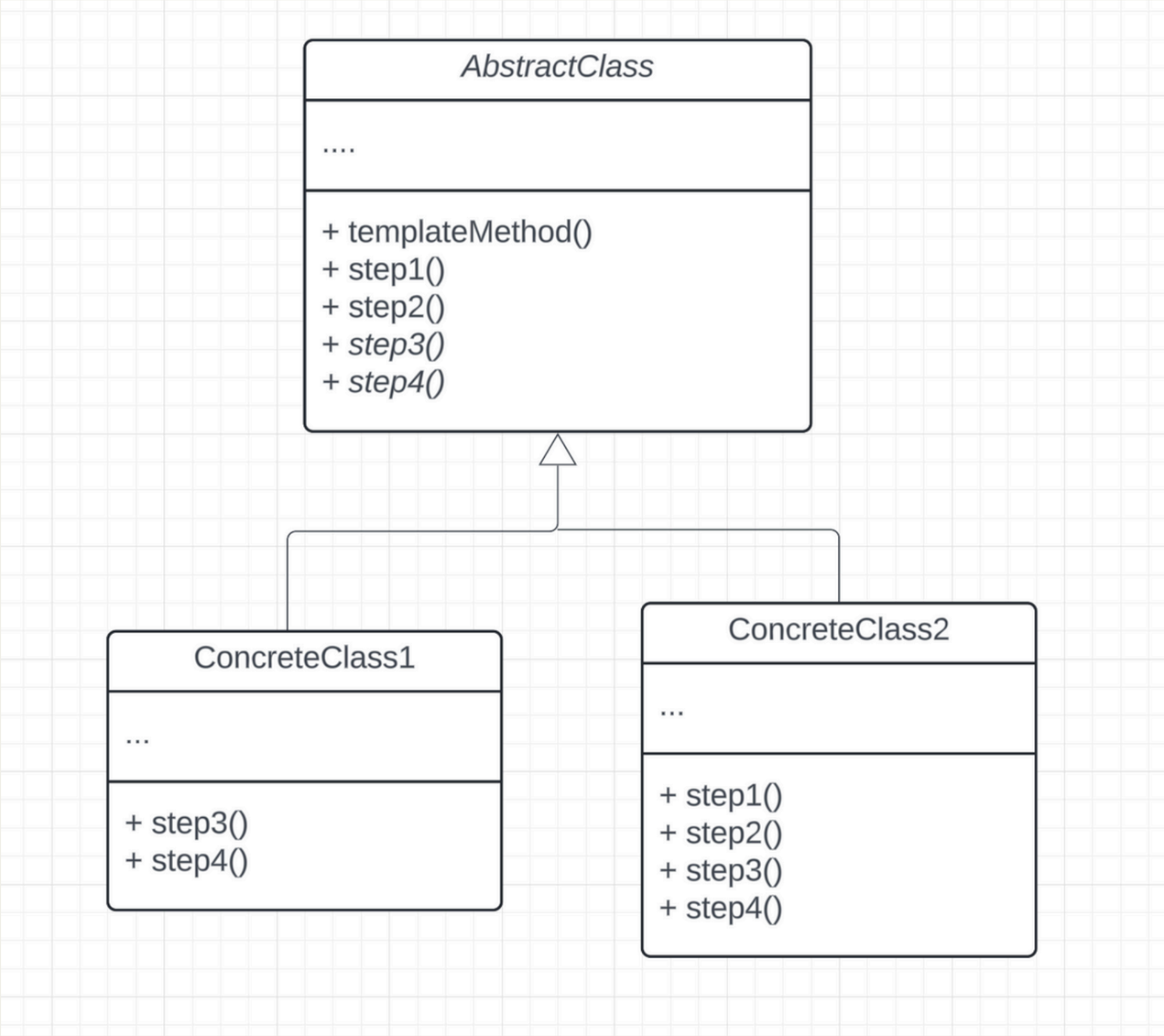
The template implements the common parts of the payment processing, and call on subclasses like card and PayPal which would implement the parts that differ.

TEMPLATE METHOD

Template Method is a behavioral design pattern that defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.



TEMPLATE METHOD



TEMPLATE METHOD

```java
public class PokemonLoader {
    public void load() {
        System.out.println("Loading Pokemon files...");
        // Some Pokemon code

        System.out.println("Creating needed Pokemon objects...");
        // Some Pokemon code

        System.out.println("Downloading Pokemon sounds and videos...");
        // Some Pokemon code

        System.out.println("Cleaning temporary files...");
        // Some code

        System.out.println("Loading local Pokemon files...");
        // Some Pokemon code
    }
}
```

TEMPLATE METHOD

```java
public class PokemonLoader {
    public void load() {
        System.out.println("Loading Pokemon files...");
        // Some Pokemon code

        System.out.println("Creating needed Pokemon objects...");
        // Some Pokemon code

        System.out.println("Downloading Pokemon sounds and videos...");
        // Some Pokemon code

        System.out.println("Cleaning temporary files...");
        // Some code

        System.out.println("Loading local Pokemon files...");
        // Some Pokemon code
    }
}
```

```java
public class YuGiOhLoader {
    public void load() {
        System.out.println("Loading Yu-Gi-Oh! files...");
        // Some Yu-Gi-Oh! code

        System.out.println("Creating needed Yu-Gi-Oh! objects...");
        // Some Yu-Gi-Oh! code

        System.out.println("Downloading Yu-Gi-Oh! sounds and videos...");
        // Some Yu-Gi-Oh! code

        System.out.println("Cleaning temporary files...");
        // Some code

        System.out.println("Loading local Yu-Gi-Oh! files...");
        // Some Yu-Gi-Oh! code
    }
}
```

TEMPLATE METHOD

```java
public abstract class BaseGameLoader {
    public void load() {
        byte[] data = loadLocalData();
        createObjects(data);
        downloadAdditionalFiles();
        cleanTempFiles();
        initialiseProfiles();
    }

    public abstract byte[] loadLocalData();
    public abstract void createObjects(byte[] data);
    public abstract void downloadAdditionalFiles();
    public abstract void initialiseProfiles();

    protected void cleanTempFiles() {
        System.out.println("Cleaning temporary files...");
        // Some code...
    }
}
```

TEMPLATE METHOD

```java
public class PokemonLoader extends BaseGameLoader {
    @Override
    byte[] loadLocalData() {
        System.out.println("Loading local Pokemon files...");
        // Some Pokemon code
    }

    @Override
    void createObjects(byte[] data) {
        System.out.println("Creating needed Pokemon files...");
        // Some Pokemon code
    }

    @Override
    void downloadAdditionalFiles() {
        System.out.println("Downloading Pokemon sounds and videos...");
        // Some Pokemon code
    }

    @Override
    void initialiseProfiles() {
        System.out.println("Loading local Pokemon files...");
        // Some Pokemon code
    }
}
```

```java
public class YuGiOhLoader extends BaseGameLoader {
    @Override
    byte[] loadLocalData() {
        System.out.println("Loading local Yu-Gi-Oh! files...");
        // Some Yu-Gi-Oh! code
    }

    @Override
    void createObjects(byte[] data) {
        System.out.println("Creating needed Yu-Gi-Oh! files...");
        // Some Yu-Gi-Oh! code
    }

    @Override
    void downloadAdditionalFiles() {
        System.out.println("Downloading Yu-Gi-Oh! sounds and videos...");
        // Some Yu-Gi-Oh! code
    }

    @Override
    void initialiseProfiles() {
        System.out.println("Loading local Yu-Gi-Oh! files...");
        // Some Yu-Gi-Oh! code
    }
}
```

FINDING PATTERNS

A frozen yogurt shop model which alters the cost and weight of a bowl of frozen yogurt based on the toppings that customers choose to add before checkout.

FINDING PATTERNS

Decorator pattern

Toppings (new behaviours) are added dynamically at runtime by users (customers).

Code and Design Smells

CODE AND DESIGN SMELLS

Mark, Bill and Jeff are working on a PetShop application. The PetShop has functionality to feed, clean and exercise different types of animals. Mark notices that each time he adds a new species of animal to his system, he also has to rewrite all the methods in the PetShop so it can take care of the new animal.

CODE AND DESIGN SMELLS

Mark, Bill and Jeff are working on a PetShop application. The PetShop has functionality to feed, clean and exercise different types of animals. Mark notices that each time he adds a new species of animal to his system, he also has to rewrite all the methods in the PetShop so it can take care of the new animal.

Code smell - Divergent change
Design problem - Open Closed Principle, high coupling

CODE AND DESIGN SMELLS

```java
public class Person {
    private String firstName;
    private String lastName;
    private int age;
    private int birthDay;
    private int birthMonth;
    private int birthYear;
    private String streetAddress;
    private String suburb;
    private String city;
    private String country;
    private int postcode;

    public Person(String firstName, String lastName, int age, int birthDay,
                  int birthMonth, int birthYear, String streetAddress, String suburb,
                  String city, String country, int postcode) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.birthDay = birthDay;
        this.birthMonth = birthMonth;
        this.birthYear = birthYear;
        this.streetAddress = streetAddress;
        this.suburb = suburb;
        this.city = city;
        this.country = country;
        this.postcode = postcode;
    }
    // Some various methods below
    // ....
}
```

CODE AND DESIGN SMELLS

Data clumps, long parameter list

Refactor by making more classes for birthday and address ("Extract Class"/ "Introduce Parameter Object")

Design problem - DRY and KISS

CODE AND DESIGN SMELLS

```
public class MathLibrary {
    List<Book> books;

    int sumTitles {
        int total = 0
        for (Book b : books) {
            total += b.title.titleLength;
        }
        return total;
    }
}

public class Book {
    Title title; // Our system just models books as titles (content doesn't matter)
}

public class Title {
    int titleLength;

    int getTitleLength() {
        return titleLength;
    }

    void setTitleLength(int tL) {
        titleLength = tL;
    }
}
```

CODE AND DESIGN SMELLS

Inappropriate intimacy (accessing public fields)

Message chains - students might bring up Law of Demeter here

Data classes/Lazy classes Design smell - High coupling, from encapsulation being broken

Fixes - make things private, just delete the classes and represent titles as strings

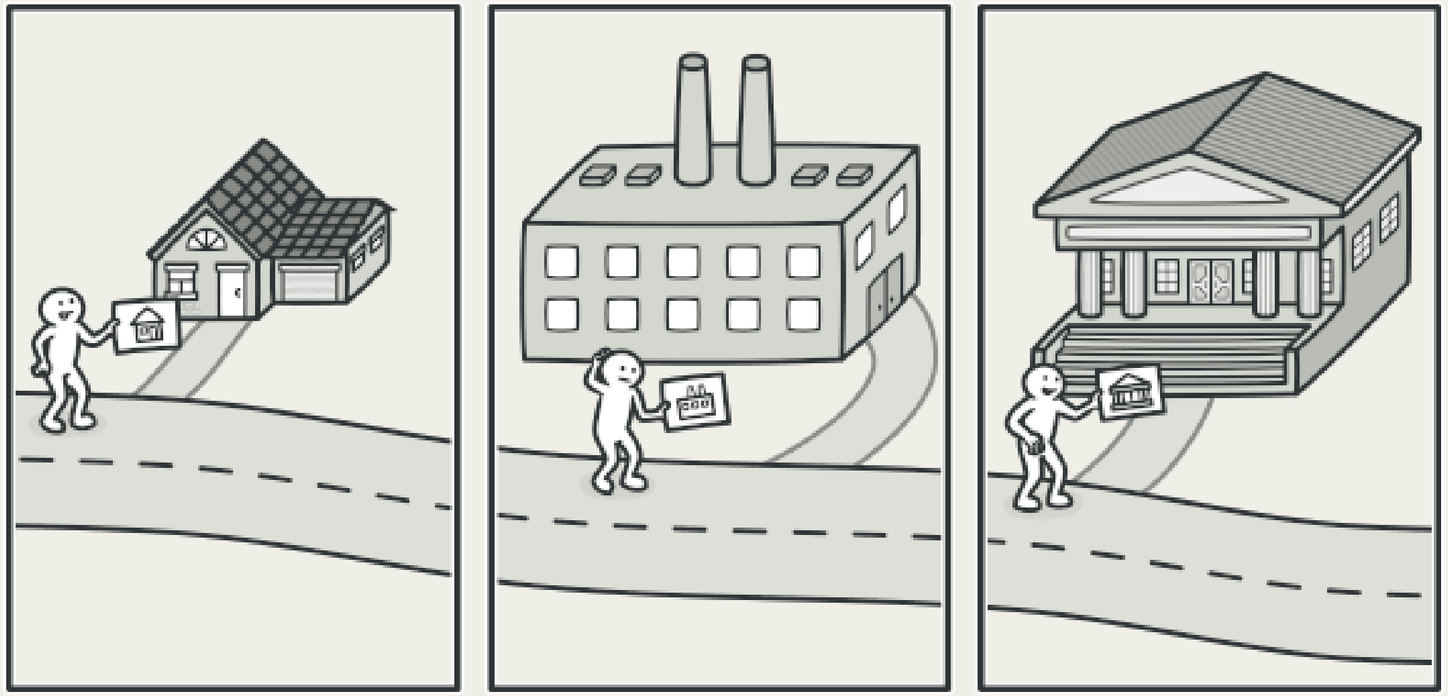
CODE AND DESIGN SMELLS

1. How do these code smells cause problems when developing code?
    - Reusability, Maintainability, Extensibility
    - Second and third examples are opposite problems (not enough classes vs too many classes) - you can take any refactoring too far
2. Is a code smell always emblematic of a design problem?
    - No - e.g "switch statements" and "comments" are often listed as code smells but are not always actually smells

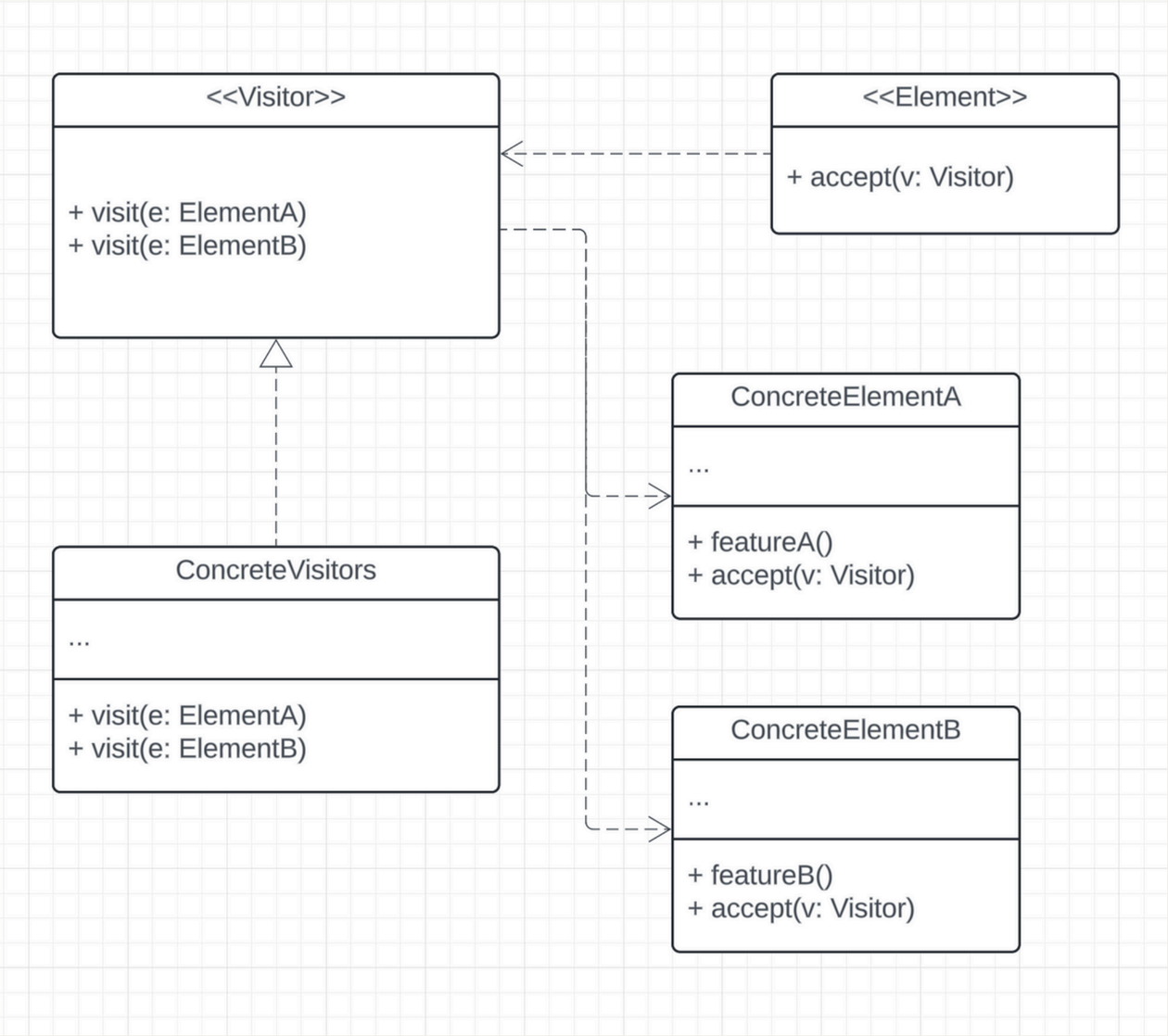
Visitor Pattern

VISITOR PATTERN

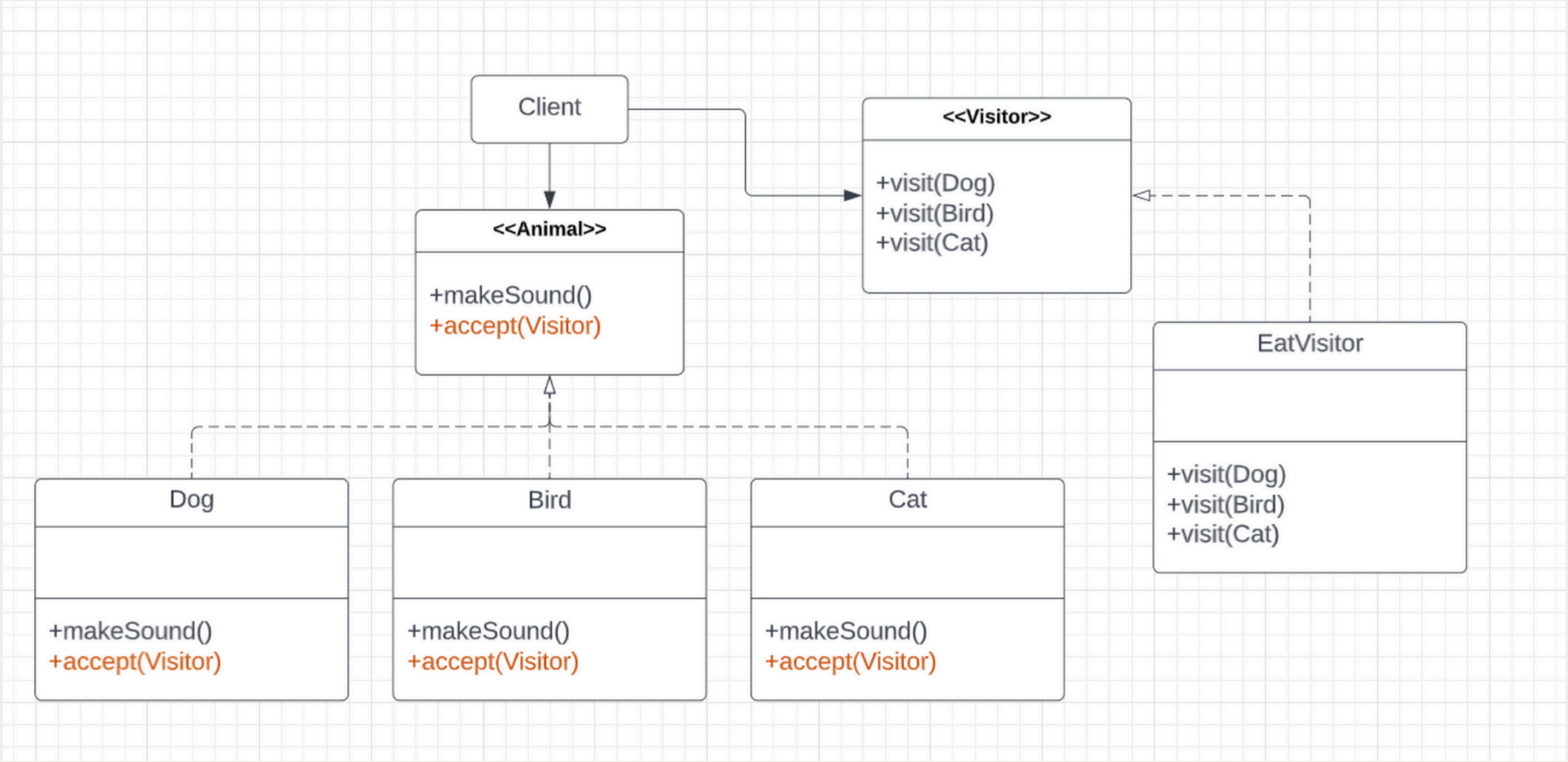
Visitor is a behavioural design pattern that lets you separate algorithms from the objects on which they operate.



VISITOR PATTERN



VISITOR PATTERN



VISITOR PATTERN

In this scenario we have Computers, Keyboards and Mouses which all are of type ComputerComponent. We want to be able to 'visit' different types of Computer components by logging the following messages:

- Looking at computer Corelli with memory 500 GB.
- Looking at keyboard Mechanical keyboard which has 36 keys.
- Looking at mouse Bluetooth mouse.

In particular though, anyone which is visiting a Computer must be validated prior to being able to visit.

Extend/modify the starter code to use the Visitor Pattern to allow different computer components to be visited.

ITS LAB TIME

YASSS