1) (15%) Create a UML class diagram for the following: A visitor class hierarchy consisting of abstract base class V plus subclasses V1 and V2 which visit subclasses C1 and C2 of base class C. Diagram V, V1, V2, C, C1, and C2, indicating which the methods visit and accept in the appropriate classes, using <abstract> to indicate which classes and methods are abstract.

![UML Diagram](image)

2) (20%) Visitor subclass V1 performs the following simple task: if a V1 instance visits a C1, it prints “I am visiting a C1” else if it visits a C2 instance it prints “I am visiting a C2”. Write the entire base class V and the subclass V1 so that it visits C1 and C2 accordingly. Do not use any “instanceof” or “if” statements. Use the space below or on the back of this sheet.

```java
public abstract class V {
    public abstract void visit(C1 c1);
    public abstract void visit(C2 c2);
}

public class V1 extends V {
    public void visit(C1 c1) { System.out.println("I am visiting a C1");}
    public void visit(C2 c2) { System.out.println("I am visiting a C2");}
}
```
3) (10%) Consider:

```java
C c = new C1();
V v = new V1();
```

Since those instances of C1 and V1 were both cast to their respective superclasses as variables c and v, is it still possible to write a line of code that creates a visitation between c and v, without explicit casting or use of instanceof? If yes, write it and explain why it works, and if no, explain why not.

Yes, this will work with the command `c.accept(v);`

Since every C1 is a C, a reference to a C1 object can be held in a C variable. However, the C1-ness of that object never changes! Once c has “accepted” the visitor v, the appropriate method will be used in v depending on the type of object c really is. This is similar to having an array of Animates, but using the appropriate visitation depending on whether the animate is a Dog or a Cat.

4) (15%) Given

```java
C1 c1 = new C1();
V1 v1 = new V1();
```

Use a UML sequence diagram to detail a visitation between c1 and v1. If you are not fully happy with your diagram, add English as necessary.
5a) (5%) Could both V and C be interfaces? Explain.

Yes, they could, since neither V nor C defines any concrete methods or variables.

5b) (5%) Provide a design reason why it might be practical necessity for V or C to be an interface.

Java only allows objects to inherit directly from one other object, but allows them to implement many different interfaces. In this way, Java provides a form of multiple inheritances. For example, suppose we had an object descended from one object hierarchy, but we wanted it to be a visitor as well. We could have it extend its superclass and implement a visitor interface.

Another example might be a case where you wanted objects to be both visitors and visitees.

6a) (5%) Describe the rationale (idea behind) the Visitor Pattern, i.e., its purpose, and why it is useful.

The Visitor Pattern is one means of enabling two objects to communicate with each other (also called message passing) in an Object Oriented design.

It is useful for a number of reasons:

- The pattern allows us to group similar functionality (e.g. drawing, moving, sound, etc.) in one place for many different types of objects
- The pattern is an elegant way of using different methods for different objects without the need for large switch style statements using instanceof
- The pattern allows us to separate functionality that is not essential to the object, so that if it must be changed later, no further changes are needed to our object’s code (e.g. 2D graphics becomes 3D graphics without any changes to the class itself)
- Allows for flexibility, since all that is needed in the visitee’s code is one simple method (i.e. accept(V v) {v.visit(this);}) to allow any number of visitors to interact with it
- Allows us to retain data across many visits (possibly to multiple classes) as an instance variable in the visitor class itself
6b) (10%) Referring to the above hierarchy, what would be two representative uses for visitors (such as V1 or V2)?

Visitors could be used to draw objects to screen, move objects, make sounds for different objects, perform calculations based on the data in the object visited (e.g. weight visitor as demonstrated in class), or just about any other situation where different behavior is expected of different types of objects.

6c) (5%) Outline the steps required to incorporate a new “visitee” subclass C3 in the C hierarchy, so that it follows this pattern.

1. Create a new subclass C3 that extends C
2. Implement the abstract accept method from C as a concrete accept method, following the pattern in the other subclasses of C
3. Add visit(C3) methods to every class in the V hierarchy, including as an abstract method in V

6d) (5%) Outline the steps required to incorporate a new visitor subclass V3 into this pattern.

1. Create a new subclass V3 that extends V
2. Implement the abstract visit methods from V as concrete visit methods for every subclass of C
3. No further changes are necessary for anything in the C hierarchy

7) (5%) What are two reasons for making a class abstract?

We might make a class abstract to provide a common framework for subclasses, yet force them to provide their own implementations of methods. By making the class abstract, we guarantee that it can never be instantiated, but can act as a category for subclasses.

We also might use this to more closely match the real world concepts we are modeling. Therefore, while we might have a Pet that is also a Cat, Dog, or Snake, we would never have a Pet that is not also some other type of animal. Pet, in this sense, is an abstract concept.