1.(a) A class is a blueprint for creating objects, while an instantiation of a class is an individual object created from that blueprint. In the given scenario, a class could be "Employee", which defines the general properties and behaviours for all employees, such as their names, job titles, and the ability to calculate their salaries. An instantiation of the class would be an individual employee, for example, a specific manager or salesperson, with a unique name, job title, and salary, created using the "Employee" class as a template.

1.(b) First example is GUI Design. The program could have a base "GUI" class that contains the common design elements, layout, and functionality used by all the graphical user interfaces. The second example is the Employee Types. The program could have a base "Employee" class that contains common properties and methods for all employee types, such as name, job title, and the ability to calculate their salary. Then, specialised employee classes, such as "Manager", "OfficeStaff", and "SalesPersonnel", could inherit from the base "Employee" class, allowing them to reuse the common properties and methods.

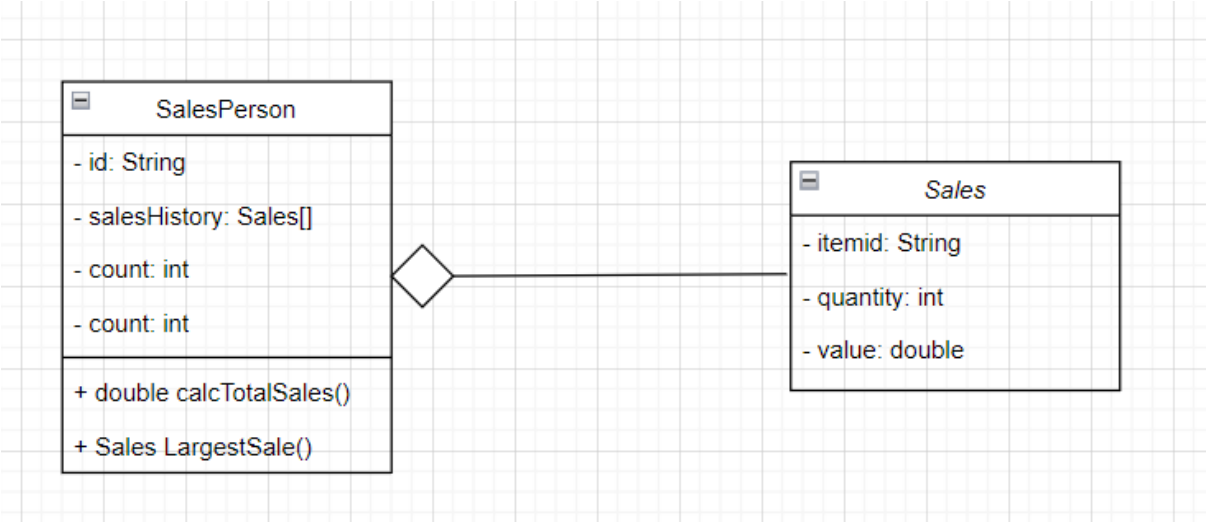
1.(c) It can facilitate the development as Libraries contain pre-built code for common tasks and functionality, allowing developers to avoid "reinventing the wheel" and focus on the unique aspects of their program. This can save time and effort during development and allows code reusability. Then we can improve the code's quality as Libraries are often developed and maintained by experienced programmers and undergo extensive testing and optimization. By using libraries, the administration program can benefit from high-quality, reliable, and a much more efficient code. Finally, Libraries often provide clear documentation and examples, making it easier for developers to understand and integrate the library's functionality into their program. This can help speed up development and reduce the chance of errors.

2(a)

```
public SalesPerson(String id) {
    this.id = id;
    salesHistory = new Sales[100];
}
```

2(b) Accessor methods make it easier for developers to understand the program's structure and how different classes interact with each other. They provide a clear and consistent way to access an object's properties and by using accessor methods, you can easily change the internal implementation of a class without affecting the rest of the program.

2(c)(i)



2(c)(ii) If the design of the Sales object is changed, such as adding new instance variables, modifying existing variables, or changing method signatures, it may cause compatibility issues with other parts of the program that rely on the current design. So, the new design could be incompatible with the old design.

2(d)

```
"C:\Users\randy\AppData\Local\Pro
102
2
2550.4
5000.0
```

2(e)

```
public double calcTotalSales() {
    double TotalSales = 0;
    for (int i = 0; i < count; i++) {
        TotalSales += salesHistory[i].getValue() * salesHistory[i].getQuantity();
    }
    return TotalSales;
}
```

2(f)

```java
public static String highest(SalesPerson[] salesPeople) {
    double maxSales = 0;
    String highestSellerId = null;

    for(SalesPerson salesPerson : salesPeople) {
        if (salesPerson != null) {
            double totalSales = salesPerson.calcTotalSales();
            if(totalSales > maxSales) {
                maxSales = totalSales;
                highestSellerId = salesPerson.getId();
            }
        }
    }

    return highestSellerId;
}
```

2(g)

```java
public static void addSales(Sales s, String id, SalesPerson[] salesPeople) {
    for (SalesPerson salesPerson : salesPeople) {
        if (salesPerson != null && salesPerson.getId().equals(id)) {
            salesPerson.setSalesHistory(s);
            break;
        }
    }
}
```

2(h)

1. Add a baseSalary and commissionRate fields to the SalesPerson class.
2. Add a calculateSalary() method in SalesPerson that calculates salary by baseSalary + (totalSales * commissionRate).
3. As the sales are made and calculated monthly, clear the salesHistory at the end of each month after calculating the salary.

2(i) The use of polymorphism that occurs in this suite of programs is called overloading. The SalesPerson class has two constructors with the same name but different parameters.