Point out at least one example of: Inheritance, Polymorphism, Cohesion, Identity, Encapsulation, and Abstraction

Inheritance:

```
public class Clerk extends Staff {
   private double registerMoney;
   private double bankWithdrawnMoney;

   public Clerk(String name) {
       super(name);
       this.registerMoney = 0.0;
       this.bankWithdrawnMoney = 0.0;
   }

public abstract class Staff {
   protected String name;
   protected int consecutiveDaysWorked;

   public Staff(String name) {
       this.name = name;
       this.consecutiveDaysWorked = 0;
   }
```

Inheritance: Inheritance can be seen in a class such as Clerk as a subclass of Staff. A Clerk class inherits the variables of Staff, while also gaining registerMoney and BankWithdrawnMoney

Polymorphism:

```
clerkg = new Clerk("Ginger");
clerkf = new Clerk("Fred");
```

When I set Ginger and Fred to clerks, it overrides the staff class and instead allows the superclass reference to hold a subclass object. There is no example of method overriding found in a function of my code though.

Cohesion:

```
public class ItemGenerator {

   private static final Random random = new Random();
   private static final String[] conditions = {"poor", "fair", "good", "very
good", "excellent"};

   public static Item generateRandomItem(int itemType) {

       String name = "Item" + random.nextInt(10000);
       double purchasePrice = 1 + (50 - 1) * random.nextDouble();
```

```
        double listPrice = 2 * purchasePrice;
        boolean isNew = random.nextBoolean();
        int dayArrived = 0;
        String condition = conditions[random.nextInt(conditions.length)];
        double salePrice = 0; // These two will be set when the item is sold
        int daySold = 0;
```

The class “ItemGenerator” is highly cohesive with the Item class. It has a single, well defined role, of creating new items for the Item class objects.

Identity:

```
public void cleanTheStore(Inventory inventory) {
    System.out.println(getName() + " is cleaning the store.");

    double damageChance;
    if (getName().equals("Ginger")) {
        damageChance = 0.05;
    } else if (getName().equals("Fred")) {
        damageChance = 0.20;
```

Each Staff object has their own Identity. the two objects in this class “Fred” and “Ginger” both interact differently with functions such as the cleanTheStore function.

Encapsulation:

```
public void checkRegister() {
    System.out.println(getName() + " is counting the money in the register.");

    if (registerMoney < 75.0) {
        System.out.println("Insufficient money in the register. Going to the
bank.");
        goToBank();
    } else {
        System.out.println("Money in the register: $" + registerMoney);
    }
}
```

goToBank is encapsulated in the checkRegister function. This hides the internal state in the checkRegister action to update the amount in the register.

Abstraction:

```
public class Main {

    public static void main(String[] args) {
        Simulation sim = new Simulation();
        sim.run();

    }
}
```

The store simulation hides implementation details and provides only functionality to the user, by putting the simulation in its own class. The simulation is the only thing necessary to run at this point although additional functionality to this abstract could be provided by allowing variables to be passed into the simulation such as days to simulate if needed.