1. **Variables**:

- In the `Student` class, `name`, `age`, and `studentId` are private instance variables used to store information about a student.

- Similarly, `major` in the `UndergraduateStudent` class and `Graduationyear` in the `GraduateStudent` class are variables specific to each type of student.

2. **Operators**:

- Operators like `==`, `&&`, and `||` are not explicitly used in the provided code, but Java operators are typically used for comparison, logical operations, and arithmetic calculations.

3. **Control Structures**:

- The `switch` statement in the `Main` class controls the flow of execution based on the user ID entered during authentication.

- The `if-else` statement in the `Main` class handles authentication by checking if the entered credentials match those in the simulated database.

4. **Methods**:

- The `displayInfo()` method in the `Student`, `UndergraduateStudent`, and `GraduateStudent` classes is used to display information about a student.

- These methods are called to print student details when the user successfully logs in.

5. **Classes**:

- The `Student` class represents a generic student with basic information such as name, age, and student ID.

- `UndergraduateStudent` and `GraduateStudent` classes extend the `Student` class, inheriting its properties and behaviors while adding specific attributes like major and graduation year.

- The `Main` class contains the `main` method and orchestrates the authentication process and subsequent actions based on the user input.

6. **Inheritance**:

- Inheritance is demonstrated through the `extends` keyword. Both `UndergraduateStudent` and `GraduateStudent` inherit from the `Student` class, inheriting its constructor and methods.

- This allows for code reuse and promotes a hierarchical relationship among classes.

7. **Encapsulation**:

- Encapsulation is achieved by making instance variables private and providing public methods (`displayInfo()`) to access and manipulate the data.

- Access to student information is restricted to these methods, preventing direct modification from outside the class.

8. **Polymorphism**:

- Polymorphism is exhibited through method overriding. The `displayInfo()` method is overridden in the `UndergraduateStudent` and `GraduateStudent` classes to provide specialized behavior for each type of student.

- Despite being invoked through the same method name, different implementations are executed based on the actual object type.

9. **Abstraction**:

- Abstraction is evident in the `Student` class, which abstracts the common attributes and behaviors of all types of students.

- The internal implementation details of how student information is stored and accessed are hidden, providing a simplified interface for interacting with student objects.

These concepts collectively contribute to the clarity, flexibility, and maintainability of the code, making it easier to understand, extend, and modify.