

ASSIGNMENT

B. LIKHITHA

Assignment 1: Student Encapsulation

Create a Student class:

- Private variables: id, name
- Public getters & setters
- Display details using getter methods.

Concepts: Data hiding, getter/setter



The screenshot shows a Java IDE with a file named 'Student.java'. The code defines a 'Student' class with private variables 'id' and 'name', public setter methods 'setId' and 'setName', and public getter methods 'getId' and 'getName'. A 'main' method is also present, which creates a 'Student' object, sets its ID to 101 and name to 'Likhitha', and prints these details using the getter methods. The output window on the right shows the execution results: 'ID: 101', 'Name: Likhitha', and a success message '=== Code Execution Successful ==='.

```
Student.java
1- class Student {
2-     private int id;
3-     private String name;
4-     public void setId(int id) {
5-         this.id = id;
6-     }
7-     public void setName(String name) {
8-         this.name = name;
9-     }
10-    public int getId() {
11-        return id;
12-    }
13-    public String getName() {
14-        return name;
15-    }
16-    public static void main(String[] args) {
17-        Student s = new Student();
18-        s.setId(101);
19-        s.setName("Likhitha");
20-        System.out.println("ID: " + s.getId());
21-        System.out.println("Name: " + s.getName());
22-    }
23-}
```

Output

ID: 101
Name: Likhitha
=== Code Execution Successful ===

Assignment 2: Bank Account

Create BankAccount class:

- Private: accountNumber, balance
- Setter validates balance (no negative value)
- Getter returns balance.

Concepts: Validation logic

```
BankAccount.java
1- class BankAccount {
2-     private int accountNumber;
3-     private double balance;
4-     public void setAccountNumber(int acc) {
5-         this.accountNumber = acc;
6-     }
7-     public void setBalance(double bal) {
8-         if (bal >= 0)
9-             this.balance = bal;
10-        else
11-            System.out.println("Balance cannot be negative!");
12-    }
13-    public double getBalance() {
14-        return balance;
15-    }
16-    public static void main(String[] args) {
17-        BankAccount b = new BankAccount();
18-        b.setAccountNumber(12345);
19-        b.setBalance(5000);
20-        b.setBalance(-200); // invalid
21-        System.out.println("Balance: " + b.getBalance());
22-    }
23-}
```

Output

```
Balance cannot be negative!
Balance: 5000.0
=== Code Execution Successful ===
```

Assignment 3: Employee Salary

Create Employee class:

- Private: empId, salary
- Setter restricts salary > 0
- Method displaySalary()

Concepts: Controlled access

```
Employee.java
1- class Employee {
2-     private int empId;
3-     private double salary;
4-     public void setEmpId(int id) {
5-         this.empId = id;
6-     }
7-     public void setSalary(double sal) {
8-         if (sal > 0)
9-             this.salary = sal;
10-        else
11-            System.out.println("Salary must be positive!");
12-    }
13-    public void displaySalary() {
14-        System.out.println("Employee ID: " + empId);
15-        System.out.println("Salary: " + salary);
16-    }
17-    public static void main(String[] args) {
18-        Employee e = new Employee();
19-        e.setEmpId(111);
20-        e.setSalary(30000);
21-        e.displaySalary();
22-    }
23-}
```

Output

```
Employee ID: 111
Salary: 30000.0
=== Code Execution Successful ===
```

Assignment 4: Product Price

Create Product class:

- Private: price
- Setter allows price only between 100 – 100000

Concepts: Business rule encapsulation



The screenshot shows a Java IDE with a file named `Product.java`. The code defines a `Product` class with a private `price` attribute. It has a `setPrice` method that validates the price (must be between 100 and 100,000) and a `getPrice` method. A `main` method creates a `Product` object, sets the price to 500, and prints it. The output window shows "Price: 500.0" and "=== Code Execution Successful ===".

```
1- class Product {
2-     private double price;
3-     public void setPrice(double price) {
4-         if (price >= 100 && price <= 100000)
5-             this.price = price;
6-         else
7-             System.out.println("Price must be between 100 and 100000!");
8-     }
9-     public double getPrice() {
10-         return price;
11-     }
12-     public static void main(String[] args) {
13-         Product p = new Product();
14-         p.setPrice(500);
15-         System.out.println("Price: " + p.getPrice());
16-     }
17- }
```

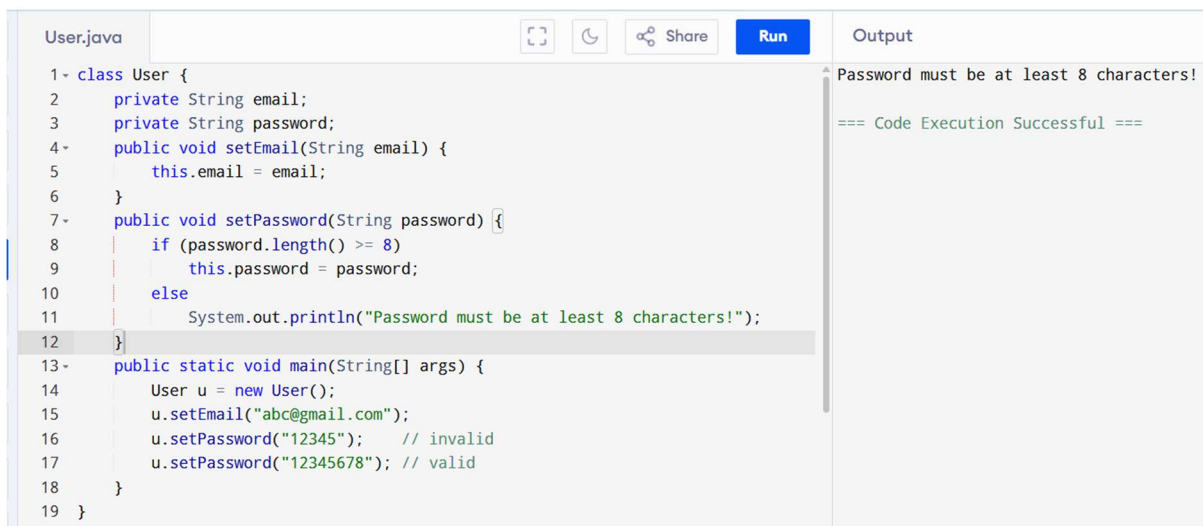
Output: Price: 500.0
=== Code Execution Successful ===

Assignment 5: Login Credentials

Create User class:

- Private: email, password
- Password setter validates length ≥ 8

Concepts: Security with encapsulation



The screenshot shows a Java IDE with a file named `User.java`. The code defines a `User` class with private `email` and `password` attributes. It has `setEmail` and `setPassword` methods. The `setPassword` method validates that the password length is at least 8 characters. A `main` method creates a `User` object, sets the email to "abc@gmail.com", and sets the password to "12345" (invalid) and "12345678" (valid). The output window shows "Password must be at least 8 characters!" and "=== Code Execution Successful ===".

```
1- class User {
2-     private String email;
3-     private String password;
4-     public void setEmail(String email) {
5-         this.email = email;
6-     }
7-     public void setPassword(String password) {
8-         if (password.length() >= 8)
9-             this.password = password;
10-        else
11-            System.out.println("Password must be at least 8 characters!");
12-    }
13-     public static void main(String[] args) {
14-         User u = new User();
15-         u.setEmail("abc@gmail.com");
16-         u.setPassword("12345"); // invalid
17-         u.setPassword("12345678"); // valid
18-     }
19- }
```

Output: Password must be at least 8 characters!
=== Code Execution Successful ===

Assignment 6: Customer Profile

Create Customer class:

- Private: name, age
- Setter restricts age ≥ 18
- Getter returns formatted data

Concepts: Validation inside setters

```
Customer.java
1- class Customer {
2     private String name;
3     private int age;
4- public void setName(String name) {
5         this.name = name; }
6- public void setAge(int age) {
7     if (age >= 18)
8         this.age = age;
9     else
10        System.out.println("Age must be 18 or above!");
11 }
12- public String getProfile() {
13     return "Name: " + name + ", Age: " + age;
14 }
15- public static void main(String[] args) {
16     Customer c = new Customer();
17     c.setName("Likhitha");
18     c.setAge(17);
19     System.out.println(c.getProfile()); }}
```

Output

```
Age must be 18 or above!
Name: Likhitha, Age: 0

=== Code Execution Successful ===
```

Assignment 7: Mobile Phone

Create Mobile class:

- Private: brand, price
- Getter returns price with GST

Concepts: Logic in getter

```
Mobile.java
1- class Mobile {
2     private String brand;
3     private double price;
4- public void setBrand(String brand) {
5         this.brand = brand;
6     }
7- public void setPrice(double price) {
8     this.price = price;
9     }
10- public double getPriceWithGST() {
11     return price + (price * 0.18); // 18% GST
12 }
13- public static void main(String[] args) {
14     Mobile m = new Mobile();
15     m.setBrand("Samsung");
16     m.setPrice(10000);
17     System.out.println("Price with GST: " + m.getPriceWithGST());
18 }
19 }
```

Output

```
Price with GST: 11800.0

=== Code Execution Successful ===
```

Assignment 8: ATM System

Create ATMAccount class:

- Private: pin, balance
- setPin() validates 4 digits

- withdraw(amount) checks balance

Concepts: Secure operations

```

ATMAccount.java
1- class ATMAccount {
2    private int pin;
3    private double balance;
4- public void setPin(int pin) {
5        if (String.valueOf(pin).length() == 4)
6            this.pin = pin;
7        else
8            System.out.println("PIN must be 4 digits!");
9    }
10- public void deposit(double amount) {
11        balance += amount;
12    }
13- public void withdraw(double amount) {
14        if (amount <= balance)
15            balance -= amount;
16        else
17            System.out.println("Insufficient balance!");
18    }

```

```

ATMAccount.java
19- public double getBalance() {
20    return balance;
21 }
22
23- public static void main(String[] args) {
24    ATMAccount a = new ATMAccount();
25    a.setPin(1234);
26    a.deposit(5000);
27    a.withdraw(2000);
28
29    System.out.println("Balance: " + a.getBalance());
30 }
31 }

```

Output

```

Balance: 3000.0
=== Code Execution Successful ===

```

Assignment 9: College Admission

Create Admission class:

- Private: marks
- Setter assigns grade internally

Concepts: Internal processing

```
Admission.java
1- class Admission {
2-     private int marks;
3-     private String grade;
4-     public void setMarks(int marks) {
5-         this.marks = marks;
6-         if (marks >= 90) grade = "A";
7-         else if (marks >= 75) grade = "B";
8-         else if (marks >= 50) grade = "C";
9-         else grade = "D";
10-    }
11-    public String getGrade() {
12-        return grade;
13-    }
14-    public static void main(String[] args) {
15-        Admission a = new Admission();
16-        a.setMarks(88);
17-        System.out.println("Grade: " + a.getGrade());
18-    }
19-}
```

Output

Grade: B

=== Code Execution Successful ===

Assignment 10: Insurance Policy

Create Policy class:

- Private: policyId, premium
- Premium calculated internally based on age

Concepts: Business rule hiding

```
Policy.java
1- class Policy {
2-     private int policyId;
3-     private double premium;
4-     public void setPolicy(int id, int age) {
5-         this.policyId = id;
6-         if (age < 30)
7-             premium = 3000;
8-         else if (age < 45)
9-             premium = 4500;
10-        else
11-            premium = 6000; }
12-    public double getPremium() {
13-        return premium;
14-    }
15-    public static void main(String[] args) {
16-        Policy p = new Policy();
17-        p.setPolicy(101, 25);
18-        System.out.println("Premium: " + p.getPremium());
19-    }
}
```

Output

Premium: 3000.0

=== Code Execution Successful ===