Lab-07
Use of abstract & final

Objectives:
Understand the abstract method & class and final method and class.

Theory:
Abstract Methods
There are cases when we have to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.
In order to ensure that a subclass does, indeed, override all necessary methods, Java provides abstract methods. Certain methods can be made mandatory to be overridden by subclasses by specifying the abstract type modifier.
To declare an abstract method, use this general form:
abstract type name(parameter-list);

Abstract Class
Any class that contains one or more abstract methods must also be declared abstract. To declare a class abstract, ***abstract*** keyword is used in front of the class keyword at the beginning of the class declaration.
abstract class class_name {
abstract type method();
}

There can be no objects of an abstract class.

Final Method
Final can be use to prevent overriding. To disallow a method from being overridden, specify ***final*** as a modifier at the start of its declaration. Methods declared as final cannot be overridden.
To declare final method, use this general form:
final type name(parameter-list){}

Final Class
Final can be use to prevent inheritance. In order to prevent a class from being inherited, the class is made final. Declaring a class as final implicitly declares all of its methods as final, too.
To do this, precede the class declaration with ***final***.

final class class_name{ }
It is illegal to declare a class as both abstract and final.

Lab Task:
```
// A Simple demonstration of abstract.
abstract class A {
abstract void callme();
```

```
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}
}
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}
}
class AbstractDemo {
public static void main(String args[]) {
B b = new B();
b.callme();
b.callmetoo();
}
}
```

Using *final* to Prevent Overriding

```
class A {
final void meth() {
System.out.println("This is a final method.");
}
}
class B extends A {
void meth() { // ERROR! Can't override.
System.out.println("Illegal!");
}
}
```

Using *final* to Prevent Inheritance

```
final class A {
void meth() {
System.out.println("This is by default final method.");
}
}
class B extends A {
void meth() { // ERROR! Can't override.
System.out.println("Illegal!");
}
}
```

Lab Assignment:

1. **Consider developing a simple bank application as per given details.**

The project contains different classes. One class is **Account** which is abstract. The Account class should implement following details;

- List of attributes:

protected String id;
protected double balance;

- Implement the following constructor:

public Account(String id, double balance) // this constructor will be used from a sub-class's constructor.

- List of methods:

public String getID() // Returns the account id.
public double getBalance() // Returns the current balance.
Public *abstract* booleanwithdraw(double amount)
Public *abstract* void deposit(double amount)

Second class is **SavingsAccount** which extends from Account. The SavingsAccount class should implement following details;

- Implement the following constructor:

Public SavingsAccount(String id, double initialDeposit): // the initial deposit passed will be at least $10.

- List of methods:

public void deposit(double amount): // The provided amount is added to the account
public boolean withdraw(double amount):
Implement the withdraw method to take out the provided amount from the account balance. Incorporate the transaction fee $2 per withdrawal. A withdrawal that potentially lowers the balance below $10 is not allowed. The balance remains unchanged but the method returns false. If the withdrawal succeeds, the method returns true.

Conclusion: