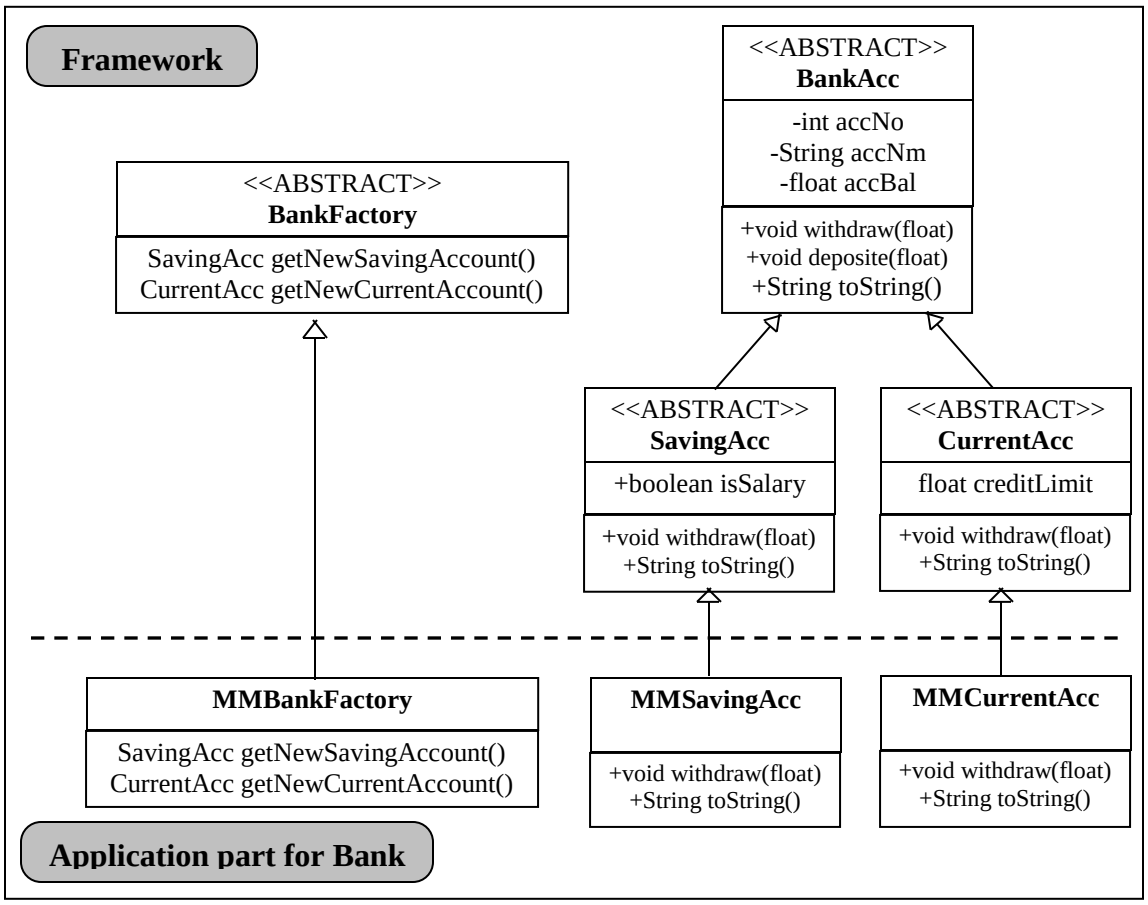
Java Case Study – I
Framework for Bank

Design a simple framework for Bank Application to represent Saving Accounts and Current Accounts.

Use the framework to design application for *MoneyMoney Bank*.

Objectives

- To understand the concept of framework in application development.
- Areas of application for Abstract classes, abstract methods etc.
- Polymorphism and its uses,
- Final fields and Lazy Initialization
- Getter and Setter methods
- Lazy Binding of methods



Design following class structures for framework…

1. Abstract BankAcc: An **abstract** class to represent a bank account.

Fields	Access	Type	Property
accNo	private	int	Read Only
accNm	private	String	Read-Write
accBal	private	float	Read Only

Constructors	Access	Parameters
	Public	AccNo, accNm, accBal

Methods	Access	Return Type	Parameters	Particulars
withdraw	public	void	float	
deposite	public	void	float	
toString	public	String		Overridden

2. Abstract SavingAcc extends BankAcc: An **abstract** class to represent specific case of Saving Account. It **extends** BankAcc. The overridden withdraw method of the class must not allow withdrawal below minimum balance in the account.

Fields	Access	Type	Property	Particulars
isSalaried	private	boolean	Read Only	
MINBAL	private	float	Read Only	static final

Constructors	Access	Parameters
	Public	AccNo, accNm, accBal, isSalaried

Methods	Access	Return Type	Parameters	Particulars
withdraw	public	void	float	Overridden
toString	public	String		Overridden

3. Abstract CurrentAcc extends BankAcc: An **abstract** class to represent specific case of Current Account. It **extends** BankAcc. The overridden withdrawal method of the class must not allow withdrawal below sum of balance and the credit limit for the account.

Fields	Access	Type	Property	Particulars
creditLimit	private	float	Read Only	final

Constructors	Access	Parameters	Particulars
	Public	AccNo, accNm, accBal, creditLimit	Lazy initialization for creditLimit.

Methods	Access	Return Type	Parameters	Particulars
withdraw	public	void	float	Overridden
toString	public	String		Overridden

4. Abstract BankFactory: An **abstract** class having necessary **factory** methods to instantiate new Saving or Current types of accounts.

Methods	Access	Return Type	Parameters
getNewSavingAcc	Public	SavingAcc	AccNo, accNm, accBal, isSalaried
getNewCurrentAcc	Public	CurrentAcc	AccNo, accNm, accBal, creditLimit

Design following class structures for application part…

5. Concrete MMSavingAcc: A concrete class representing bank specific Saving Account. It extends SavingAcc.

Fields	Access	Type	Property	Particulars
MINBAL	private	float	Read Only	static final

Constructors	Access	Parameters
	Public	AccNo, accNm, accBal, isSalaried

Methods	Access	Return Type	Parameters	Particulars
withdraw	public	void	float	Overridden
toString	public	String		Overridden

6. Concrete MMCurrentAcc: A concrete class representing bank specific Current Account. It extends CurrentAcc.

Constructors	Access	Parameters	Particulars
	Public	AccNo, accNm, accBal, creditLimit	Lazy initialization for creditLimit.

Methods	Access	Return Type	Parameters	Particulars
withdraw	public	void	float	Overridden
toString	public	String		Overridden

7. Concrete MMBankFactory: A concrete class having complete implementation of necessary factory methods to instantiate MMSavingAcc and MMCurrentAcc. It extends BankFactory.

Methods	Access	Return Type	Parameters
getNewSavingAcc	public	MMSavingAcc	AccNo, accNm, accBal, isSalaried
getNewCurrentAcc	public	MMCurrentAcc	AccNo, accNm, accBal, creditLimit

8. The Entry point for application part: Design an entry point for the application to test working of a framework.
    a. Assign instance of MMBankFactory to BankFactory reference.
    b. Instantiate MMSavingAcc and refer it through reference SavingAcc.
    c. Instantiate MMCurrentAcc and refer it through reference CurrentAcc.
    d. Invoke withdraw() method.
    e. Invoke toString() method.