Tentamen i Objektorienterad modellering och design

Lösningar

1. **a.**

```
public abstract class Member {
    protected String name;
    public abstract int amount();
        public String payment() {
            return name + "den avgift du ska betala är" + amount();
        }
    }

public class Senior extends Member {
    public int amount() {
        return 3000;
    }
}
```

Klasserna Junior och Passive analoga med Senior.

b.

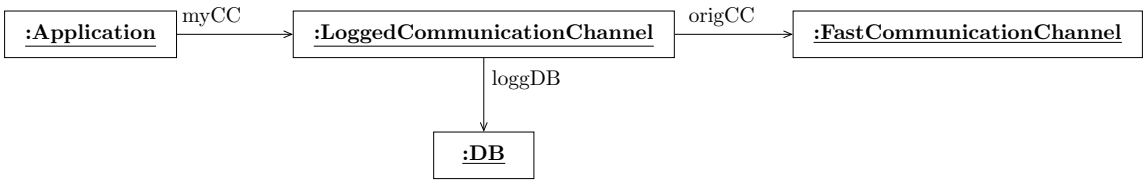
```
abstract public class Member {
    private String name;
    TypeOfMembership membership;

    public Member(String namn, TypeOfMembership medlemskap) {
        this.name = namn;
        this.membership = medlemskap;
    }

    public void setMedlemskap(TypeOfMembership ms) {
        membership = ms;
    }

    public String payment() {
        return name + "den avgift du ska betala är" +
            membership.amount();
    }
}

public interface TypeOfMembership {
    int amount();
}

public class SeniorStrategy implements TypeOfMembership {
    public int amount() {
        return 3000;
    }
}
```

2.
```
public class Account extends Observable {
    protected int saldo;
    public void deposit(int amount) {
        saldo += amount;
        setChanged();
        notifyObservers();
    }
    public void withdraw(int amount) {
        saldo -= amount;
        setChanged();
        notifyObservers();
    }
    public int getSaldo() {
        return saldo;
    }
}

public class Alarm implements Observer {
    protected int limit;
    AccountManager theAccount;
    public Alarm(AccountManager theAccount, int limit) {
        this.limit = limit;
        this.theAccount = theAccount;
        theAccount.addObserver(this);
    }

    @Override
    public void update(Observable o, Object arg) {
        if (theAccount.getSaldo() < limit) {
            alert();
        }
    }

    public void alert() {
        // omissions
    }
}
```

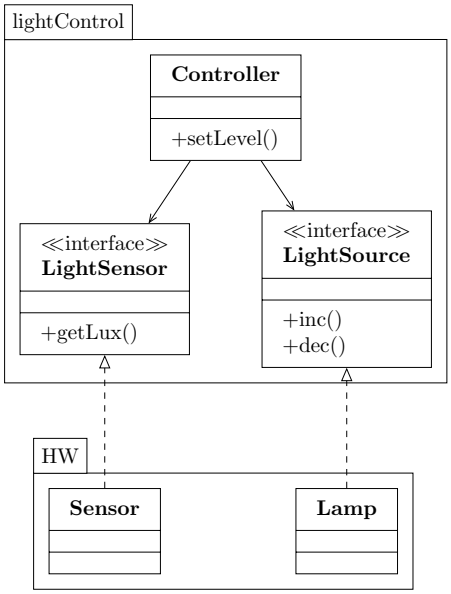
3.
```
public class Application {
    CommunicationChannel myCC = new FastCommunicationChannel();
    public static void main(String[] args) {
        // omissions
    }
    private void send(String text) {
        myCC.send(text);
    }
    private String receive() {
        return myCC.receive();
    }
    public void startLogg(DB loggDB) {
        myCC = new LoggedCommunicationChannel(loggDB, myCC);
    }
    public void stopLogg() {
        if (myCC instanceof LoggedCommunicationChannel) {
            myCC = ((LoggedCommunicationChannel)myCC).getOrigCC();
        }
    }
}
public interface CommunicationChannel {
    public void send(String text);
    public String receive();
}
public class LoggedCommunicationChannel implements CommunicationChannel {
    CommunicationChannel origCC;
    DB loggDB;
    public LoggedCommunicationChannel(DB db, CommunicationChannel origCC) {
        this.origCC = origCC;
        this.loggDB = db;
    }
    public CommunicationChannel getOrigCC() {
        return origCC;
    }
```

```
    public void send(String text) {
        loggDB.appendSent(text);
        origCC.send(text);
    }
    public String receive() {
        String text;
        text = origCC.receive();
        loggDB.appendReceived(text);
        return text;
    }
}
```

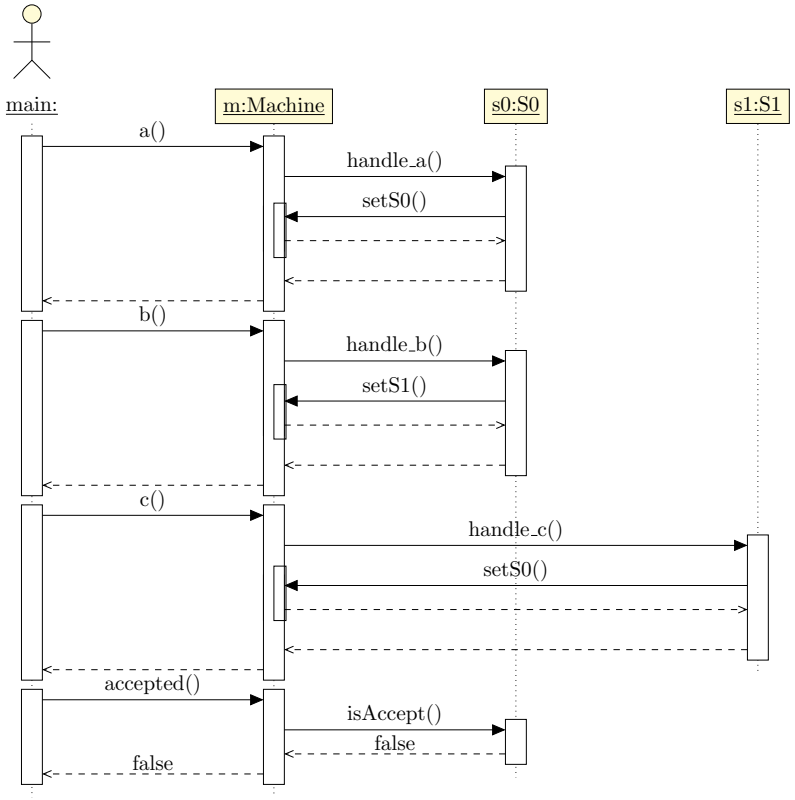
Objektdiagram (kallades instansmodell i uppgiften):



4. Klassdiagram:



5.


main:
m:Machine
s0:S0
s1:S1
a()
handle_a()
setS0()
b()
handle_b()
setS1()
c()
handle_c()
setS0()
accepted()
isAccept()
false
false