

Beskrivning

Denna laboration kommer att validera era kunskaper inom objektorientering, enhetstester och mocking. Laborationen ska utföras i grupper om fyra eller fem. Alla har fått en plats i en grupp. Hur din grupp ser ut kommer att presenteras. Laborationen går ut på att skriva ett gränssnitt mot en bankomat.

Denna laboration är obligatorisk och kan som högst ge nivå **godkänd**.

Alla i gruppen ska bidra till koden och kunna förklara den.

Uppgiften ska vara klar senast onsdagen den **11 februari 2015**. Då kommer koden att redovisas.

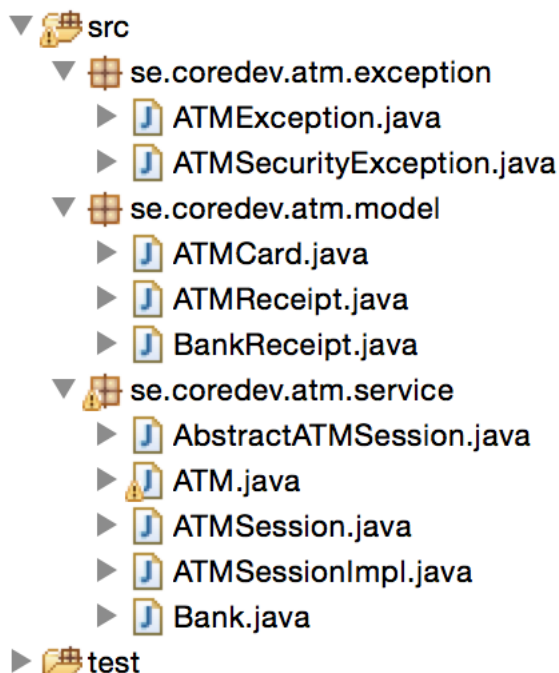
Generella krav

All kod ska vara formaterad på passande sätt. Alla variabler ska vara `private` och `final` om de inte behöver vara annat. Du använder de klasser och interface som beskrivs nedan.

Det ska finnas tester som verifierar:

- Att ett `ATMException` kastas om beloppet är mindre än 100, över 10000 eller inte ett jämnt hundratal
- Att ett `ATMSecurityException` kastas om fel pinkod matas in
- Att ett `ATMException` kastas om ett bankomatkort som inte är anslutet till någon av bankomatens banker användas
- Att ett `ATMException` kastas om du en andra gång anropar någon av de metoder som avslutar en session på din subklass till `AbstractATMSession`
- Alla publika metoder i din subklass till `AbstractATMSession` är testade
- Att ett `IllegalArgumentException` kastas om en tom lista med banker skickas till `ATM`-klassen vid `new`
- Att ett `ATMException` kastas om ett högre belopp än det finns täckning för anges till `withdrawAmount`

Så här skulle ert projekt kunna vara strukturerat (**byt ut paketnamnen så att de passar er**)



Generellt om klasserna och interfacen som måste vara med

ATMSecurityException - är ett unchecked exception som du själv skapar

ATMException - är ett unchecked exception som du själv skapar

// Denna klass representerar en bankomat.

// Den har till uppgift att hålla anslutna banker och returnera en ATMSession som i sin

// tur kan användas för att kommunicera med en bank

```
public final class ATM
{
    private final Map<String, Bank> banks;

    public ATM(List<Bank> banks)
    {
        // To be Implemented
    }

    // Returnerar en implementation av ATMSession om pikoden är korrekt. Annars kastas ett
    // ATMSecurityException
    public ATMSession verifyPin(int pin, ATMCard card)
    {
        // To be Implemented
    }

    // Returnerar banken som angivet bankomatkort är kopplat till. Hittas ingen bank kastas ett
    // ATMException
    private Bank getBank(ATMCard card)
    {
        // To be Implemented
    }
}
```

// Detta interface ska implementeras som stateless, dvs. subklasser ska inte hålla något state för den
// som anropar. Däremot behöver den hålla koll på om en metod som avslutar/invaliderar en session
// har anropats.

// Den returnerar istället ett transactionId som senare kan användas för att begära ett kvitto på uttaget

```
public interface ATMSession
{
    // Denna metod ska kasta ett exception om beloppet som anges är:
    // < 100 eller > 10000 eller inte ett jämnt hundratal (exempelvis kastas ett exception om
    // beloppet är 110)
    // Denna metod avslutar en session
    // Om man anropar denna metod två gånger efter varandra ska ett ATMException kastas
    // Denna metod kastar ett ATMException om ett högre belopp än det finns täckning för anges
    long withdrawAmount(int amount);

    // Begär ett kvitto från banken för angivet transactionId
    ATMReceipt requestReceipt(long transactionId);

    // Denna metod returnerar aktuellt belopp på kontot som är knutet till denna session
    // Denna metod avslutar en session
    // Om man anropar denna metod två gånger efter varandra ska ett ATMException kastas
    long checkBalance();
}
```

```
// Abstrakt implementation av ATMSession.
// Denna klass kommer att subclassas för att tillhandahålla en konkret implementation
public abstract class AbstractATMSession implements ATMSession
{
    protected final ATMCARD atmCard;
    protected final Bank bank;

    public AbstractATMSession(ATMCARD atmCard, Bank bank)
    {
        this.atmCard = atmCard;
        this.bank = bank;
    }
}
```

```
// Detta interface ska mockas. Detta interface representerar en bank
public interface Bank
{
    String getBankId();
    long getBalance(String accountHolderId);
    long withdrawAmount(int amount);
    BankReceipt requestReceipt(long transactionId);
}
```

```
// Denna klass innehåller information om ett bankomatkort och ska inte mockas utan användas som
// den är
```

```
public final class ATMCARD
{
    private final String accountHolderId;
    private final String bankId;
    private final int pin;

    public ATMCARD(String accountHolderId, String bankId, int pin)
    {
        this.accountHolderId = accountHolderId;
        this.bankId = bankId;
        this.pin = pin;
    }

    public String getAccountHolderId()
    {
        return accountHolderId;
    }

    public String getBankId()
    {
        return bankId;
    }

    public boolean verifyPin(int pin)
    {
        return this.pin == pin;
    }
}
```

// Denna klass innehåller information om ett uttag och ska **inte** mockas utan användas som den är

```
public final class ATMReceipt
{
    private final long transactionId;
    private final int amount;
    private final Date date;

    public ATMReceipt(long transactionId, int amount)
    {
        this.transactionId = transactionId;
        this.amount = amount;
        this.date = new Date(); // Date in java.util
    }

    public long getTransactionId()
    {
        return transactionId;
    }

    public int getAmount()
    {
        return amount;
    }

    public Date getDate()
    {
        return date;
    }
}
```

// Denna klass innehåller information om ett uttag sett från **bankens sida** och ska **inte** mockas utan
// användas som den är

```
public final class BankReceipt
{
    private final String bankId;
    private final long transactionId;
    private final int amount;
    private final Date date;

    public BankReceipt(String bankId, long transactionId, int amount, Date date)
    {
        this.bankId = bankId;
        this.transactionId = transactionId;
        this.amount = amount;
        this.date = date;
    }

    public String getBankId()
    {
        return bankId;
    }

    public long getTransactionId()
    {
        return transactionId;
    }

    public int getAmount()
    {
        return amount;
    }

    public Date getDate()
    {
        return date;
    }
}
```