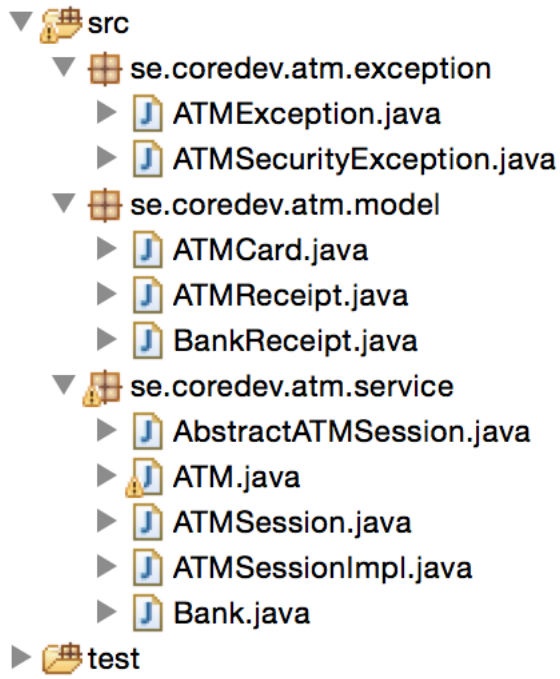
Övning - Bankomat - Steg 1
Denna övning går ut på att ni ska bygga en webbaserad bankomat. Ni kommer att utgå från dessa klasser som beskrivs nedan. Etter det kommer ni att exponera funktionaliteten som ett web-api m.h.a. Servlet/Filter/Listener (detta görs i Steg 2 - se nedan). Detta steg bygger alltså logiken för en (enkel) bankomat.

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 subklassas 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
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
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;
	}
}
```

Steg 2
Den funktionalitet som ATM/ATMSession erbjuder ska gå att nå via HTTP. Börja med att fundera på hur URL:erna kommer att se ut, vilken/vilka Content-Types som får användas samt vilka Status-Codes som ska returneras för de olika operationerna. Notera att ni kan behöva ändra ATM/ATMSession för att detta ska vara möjligt.

Extra 1
Du använder dig av ServletFilter för att kontrollera om en operation är giltig.

Extra 2
Lägg till funktionalitet för att sätta in pengar på ett visst konto.