Čau, takže tady máš návod jak použít síťovou vrstvu. Není to úplně triviální seskupení tříd a interfaců, takže ten návod se ti bude hodit. Mimochodem pochopit princip fungování by ti měli pomoct sekvenční diagramy, které tam taky jsou.

K dispozici jsem ti dal celej eap soubor, takže si můžeš prohlídnout, jaký všechny třídy tam budou, ale pro tvoje potřeby, by tě měli zajímat jenom následující části: třídy v balíčku *Protocols* (a to ty třídy který končí slovem Protocol), komplet obsah balíčku *Business* a třída *ConnectionProvider*.

A kdyby se ti zdálo, že těch tříd je tam málo, tak tě můžu ujistit, že tam ještě přibydou :-D …bez obav, to už budou jenom moje interní třídy.

# Koncept

Jak by měla síťová vrstva pracovat:

Ve třídě aplikace (myslím tím Application) by se měla nacházet instance ConnectionProvider (o její inicializaci se postarám já, až rozchodím tu šifrovanou komunikaci, ty ber v potaz jenom to, že tam bude a budeš jí moci použít). Tahle třída by ti měla přes metodu *getNetworkInterface*() poskytovat instanci třídy, která zajišťuje síťovou komunikaci (přenos dat). Pokud budeš chtít komunikovat se serverem, budeš zároveň potřebovat třídu protokolu (jsou to třídy, který končí slovem Protocol v balíčku Protocols - ty, ti poskytují rozhraní, se kterým pracuješ) a ještě nějakou další třídu, která bude celou tu komunikaci provozovat (tu už si musíš napsat sám, ale pro účely toho souboru eap jsem si ty třídy dovolil vytvořit, jsou to třídy v balíčku Business).

Teď ti popíšu, jak to bude probíhat například při kontrole parkování. Nejprve se ti otevře ta aktivita, potom si v ní vytvoříš třídu *SMSParking* (balíček Business). Když pak budeš chtít odeslat dotaz, zda to auto může parkovat, zavoláš si nějakou tvoji metodu na tom *SMSParking* a ta si z ConnectionProvideru vytáhne *NetworkInterface* a vytvoří *třídu protokolu* *SMSParkingProtocol*, té předá v konstruktoru ten objekt *NetworkInterface* a za pomoci metody *setSMSParkingProtocolListener(this)* se ta třída *SMSParking* nastaví jako posluchač událostí, které přichází z třídy protokolu (to znamená, že ta třída, která je posluchačem musí implementovat interface *ISMSParkingProtocolListener* - balíček Business/Listeners). Pak už můžeš poslat dotaz na server. Na třídě protokolu zavoláš metodu *checkParking*(), který předáš SPZ a čekáš na odpověď (ta metoda je neblokující, takže čekáš, aniž by ti to blokovalo vlákno). Při tom čekání se může stát několik věcí. Zaprvé ti třída protokolu může vyhodit událost ConnectionTerminated (nečekaně skrze metodu *onConnectionTerminated*() :-D) to znamená, že síť teď není dostupná (třeba jsi mimo oblast pokrytí signálem). Pokud se ti to stane ještě před tím, než dostaneš událost MessageSent, znamená to, že došlo k chybě při odesílání a zpráva na server nedorazila (takže musíš dotaz na parkování odeslat znova). Pokud ConnectionTerminated dostaneš až po MessageSent, znamená to, že zpráva sice odeslána byla, ale spojení se přerušilo při příjmu odpovědi ze serveru, takže si stejně dotaz budeš muset zopakovat. Za druhé se ti může stát, že vše proběhne v pořádku a ty dostaneš zprávu ReceivedSMSParkingInfo. V parametru metody *onReceivedSMSParkingInfo*() pak dostaneš objekt s informacemi o parkování, který můžeš zpracovat. Když pak budeš chtít ověřit další auto, tak zase zavoláš metodu *checkParking().* Důležité taky je, že když aktivita končí, nebo když už se serverem prostě komunikovat nepotřebuješ, musíš na třídě protokolu zavolat metodu *disconnectProtocol*(), to zajistí odpojení všech posluchačů událostí.

***Poznámka 1:*** komunikovat se serverem může v jeden čas, jen jeden protokol (resp. třída protokolu). To znamená, že když by si chtěl před přenosem PL nahrát geolokační data, musíš nejprve vytvořit a použít třídu protokolu *GeoDataProtocol* (s tím, že jí předáš *NetworkInterface*), pak jí odpojit a teprve teď vytvořit a použít *TicketSyncProtocol* (pokud bys vytvořil obě třídy současně, později vytvořená třída by přepsala napojení na *NetworkInterface* a první vytvořená třída by nefungovala).

***Poznámka 2:*** Já pro zjednodušení uvažuju, že si vytvoříš nějaké pomocné třídy jako *SMSParking* a ty pak budeš z aktivity volat, ale ve skutečnosti pokud budeš chtít, můžeš si tu implementaci vytvořit rovnou v aktivitě (například by to mohlo být právě u kontroly parkování). Nesmíš ale zapomenout aktivitě přidat interface posluchače tj. I“názevTřídyProtokolu“Listener a pak aktivitu v protokolu zaregistrovat jako posluchače.

***Poznámka 3:*** Jak už z textu výše vyplynulo, existují 3 události. MessageSent se volá poté, co jsou data odeslána na server. ConnectionTerminated se volá poté co při odesílání nebo přijímání dat dojde ke ztrátě spojení. A pak je zde událost příjmu odpovědi ze serveru. Ta ale nemusí být přítomna vždy, protože například na odeslání geolokačních dat, server nijak neodpovídá.

# Vysvětlení tříd v balíčku Protocol:

Tohle jsou třídy, naproti kterým budeš programovat, samotný třídy přijímají instance specifických typů, převádí je na byty a odesílají na síť, to, jak to dělají je moje práce. Tyhle třídy mají dost metod, ale tebe by měli zajímat jen ty, přes který komunikuješ se serverem - budu jim říkat *výkonné metody* (u *GeoDataProtocol* je to metoda *sendGeoData*(...), u *TicketSyncProtocol* je to *sendTicket*(...) atd.) a metody set"názevTřídy"Listener(...), který přijímají posluchače událostí z těchto tříd (posluchač by měla být ta třída, která komunikuje s třídou protokolu. To aby věděla, co jí třída protokolu odpovídá).

## Jak fungují výkonné metody:

Protože jsem chtěl, aby se ti programovalo co nejjednodušeji, všechny tyhle metody fungují asynchroně (pěkná pakárna ty asynchroní kraviny :-( ). To znamená, že pokud tu metodu zavoláš a předáš jí například objekt parkovacího lístku, tak ta metoda nečeká až se data odešlou přes síť, ale rovnou se ti vrátí a samotný data se odešlou postupně v jiném vláknu. Touhle dlouhou větou jen říkám, že *všechny výkonné metody jsou neblokující.* Ale pozor! Ačkoliv je metoda neblokující, pokud bys ihned po prvním zavolání zavolal výkonnou metodu znovu, metoda zatuhne, protože musí počkat, až se odešle předchozí zpráva. To znamená, že nemůžeš všechny PL odeslat jen tak v nějakém cyklu. To, že můžeš odeslat další PL ti řekne zavolání metody *onMessageSent*(), kterou implementuje třída, která ovládá třídu protokolu a která je zavolaná třídou protokolu jako callback.

# Poznámky

***Poznámka 4:*** Při zkoumání, jak bude fungovat přihlašování na server, jsem zjistil, že aby byla síťová vrstva alespoň trochu jednodušší, bude potřeba, aby se policajt před použitím programu přihlásil. To ti píšu proto, že podle přihlašovacího scénáře v tom eap souboru na svn to není potřeba. Teď bude platit, že diagram „Začátek služby“ bude povinný. To znamená, že v diagramu „Práce v terénu“, nebude moci nastat případ, kdy bude policista nepřihlášen. V tom eapu to pak opravím.

***Poznámka 5:*** Moc se nepouštěj do programování přihlašování k serveru. Já sice už vím, jak by to mělo probíhat, ale je možný, že až budu programovat tu šifrovanou komunikaci, budu to muset trochu poupravit. Jinak plán je takový, že po stisknutí tlačítka přihlásit se vytvoří připojení na server (tady zatím trochu váhám jak na to) a po zadání správných údajů bude možné se serverem komunikovat i pomocí jiného protokolu než jen přihlašovacím. Jinak přihlašování používá úplně stejný postup komunikace se serverem, jako jsem ti už popsal výše.

***Poznámka 6:*** Jak už sem se zmiňoval v poznámce 1, před odesláním PL na server bys měl odeslat nejdřív geolokační data (aby si je ty policajti nemohli syslit). Jinak já jsem nevěděl, jak budou ty geolokační data uložený, takže požaduju jenom třídu List. Nějak tam ty data nasyp a já se podle toho pak zařídím.

***Poznámka 7:*** V balíčku Business/DataClasses jsem vytvořil pár pomocných tříd a výčtů který jsou potřeba v callback metodách.

***Poznámka 8:*** Strukturu balíčků, jakou jsem tam vytvořil, nemusíš dodržet, jenom jsem to potřeboval nějak oddělit, takže to je, jak to je.

***Poznámka 9:*** Přestože je tobě tahle informace asi k ničemu, upravil jsem i tu excelovskou tabulku s posílanými zprávami, ale zatím jsem ty změny nezapracoval, takže na svn je pořád ta stará verze.