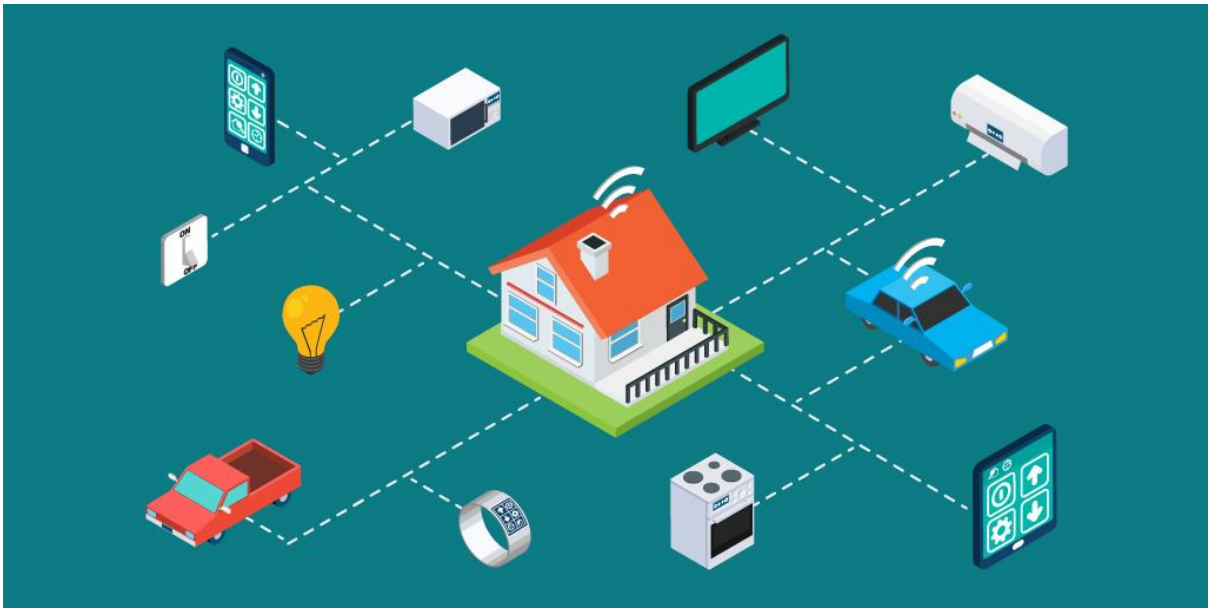

Projekt: Smart Home

Abstrakt

Vytvořit aplikaci pro virtuální simulaci inteligentního domu, kde simulujeme chod domácnosti, používáme jednotlivá zařízení domu a vyhodnocujeme využití, spotřebu, volný a pracovní čas jednotlivých osob.

Funkční požadavky

1. Entity se kterými pracujeme je dům, okno (+ venkovní žaluzie), patro v domu, senzor, zařízení (=spotřebič), osoba, auto, kolo, domácí zvíře jiného než hospodářského typu, plus libovolné další entity
2. Jednotlivá zařízení v domu mají API na ovládání. Zařízení mají stav, který lze měnit pomocí API na jeho ovládání. Akce z API jsou použitelné podle stavu zařízení. Vybraná zařízení mohou mít i obsah - lednice má jídlo, CD přehrávač má CD.
3. Spotřebiče mají svojí spotřebu v aktivním stavu, idle stavu, vypnutém stavu
4. Jednotlivá zařízení mají API na sběr dat o tomto zařízení. O zařízeních sbíráme data jako spotřeba elektřiny, plynu, vody a funkčnost (klesá lineárně s časem)
5. Jednotlivé osoby a zvířata mohou provádět aktivity(akce), které mají nějaký efekt na zařízení nebo jinou osobu. Např. *Plynovy_kotel_1[oteverny_plyn] + Otec.zavritPlyn(plynovy_kotel_1) -> Plynovy_kotel_1[zavreny_plyn].*
6. Jednotlivá zařízení a osoby se v každém okamžiku vyskytují v jedné místnosti (pokud nesportují) a náhodně generují eventy (eventem může být důležitá informace a nebo alert)
7. Eventy jsou přebírány a odbavovány vhodnou osobou (osobami) nebo zařízením (zařízeními). Např.:
    - *čidlo na vítr (vítr) => vytažení venkovních žaluzií*
    - *jistič (výpadek elektřiny) => vypnutí všech nedůležitých spotřebičů (v provozu zůstávají pouze ty nutné)*
    - *čidlo na vlhkost (prasklá trubka na vodu) => máma -> zavolání hasičů, táta -> uzavření vody;dcera -> vylovení křečka*
    - *Miminko potřebuje přebalit => táta se skrývá, máma -> přebalení*
    - *Zařízení přestalo fungovat => …*
    - *V lednici došlo jídlo => ...*
8. Vygenerování reportů:

    - *HouseConfigurationReport*: veškerá konfigurační data domu zachovávající hieararchii - dům -> patro -> místnost -> okno -> žaluzie atd. Plus jací jsou obyvatelé domu.
    - *EventReport*: report eventů, kde grupujeme eventy podle typu, zdroje eventů a jejich cíle (jaká entita event odbavila)
    - *ActivityAndUsageReport*: Report akcí (aktivit) jednotlivých osob a zvířat, kolikrát které osoby použily které zařízení.
    - *ConsumptionReport*: Kolik jednotlivé spotřebiče spotřebovaly eletřiny, plynu, vody. Včetně finančního vyčíslení.
    - *SavingsReport*: Za každou místnost report, kdy v ní běžely spotřebiče (televize, topení atd.) nikdo v místnosti nebyl, výpis zařízení, které nebyly vůbec použity.
9. Při rozbití zařízení je nutné prozkoumat dokumentaci k zařízení (najít záruční list, projít manuál na opravu atd.) a situaci vyřešit nápravnou akcí (např. Oprava svépomocí, koupě nového atd.). Manuály zabírají mnoho místa a trvá dlouho než je najdete. *Hint: dokumentace je přístupná jako proměnná přímo v zařízení, nicméně se dotahuje až, když je potřeba.*
10. Rodina je aktivní a volný čas tráví zhruba v poměru (50% používání spotřebičů v domě a 50% sport kdy používá sportovní náčiní kolo nebo lyže). Když není volné zařízení nebo sportovní náčiní, tak osoba čeká.

Nefunkční požadavky

- Není požadována autentizace ani autorizace
- Aplikace může běžet pouze v jedné JVM
- Vygenerování reportů zpětně v libovolné verzi. Verze je stav objektů v libovolném minulém čase, efektivní paměťová složitost - při update stavu domu neklonujeme celý objektový graf
- Aplikace by měla být psána jako framework, který mohou používat další vývojáři - nikoliv jako jednorázově a jednoúčelově napsaná aplikace - tedy tak, abyste vaši aplikaci mohli dát další osobě, která si přes veřejné API provede implementaci pro svůj dům. Použití je pro implementátora jednoduché a implementátor zároveň nemá přístup k interním funkcím frameworku.
- Reporty jsou generovány do textového souboru
- Konfigurace domu, zařízení a obyvatel domu může být nahrávána přímo z třídy nebo externího souboru (preferován je json)

Určitě vhodné design patterny

- State machine
- Iterator
- Factory/Factory method
- Decorator/Composite
- Singleton
- Visitor/Observer/Listener
- Chain of responsibility
- Partially persistent data structure
- Object Pool
- Lazy Initialization

Požadované výstupy

- Design ve formě class diagramů a stručného popisu jak chcete úlohu realizovat
- Veřejné API - Javadoc vygenerovaný pro funkce, kterými uživatel pracuje s vaším software
- Alespoň 20% tříd veřejného API pokryto unit testy

- Dvě různé konfigurace domu a pro ně vygenerovány reporty za různá období. Minimální konfigurace alespoň jednoho domu je: 6 osob, 3 zvířata, 8 typů spotřebičů, 20 ks spotřebičů, 6 místností, jedny lyže, dvě kola.

Hodnocení

- Čistota softwarového designu aplikace a veřejného API (40%)
- Množství naimplementovaných funkčních požadavků (40%)
- Vlastní inovace - inovativní funkční požadavek, který jste přidali (20%)

Logistika

- Úloha se vypracovává ve dvou - třech studentech (ze stejného cvičení), přičemž je požadováno, aby každá osoba rozuměla i částem, které nepsala. Semestrální úloha může být znovu otevřena u zkoušky.
- Minimálně tři týdny před finálním odevzdáním je potřeba poslat cvičícímu design vaší aplikace. Pokud by byl design špatný, došlo k odchýlení od designu nebo naopak budete chtít poradit, tak bude konzultace provedena na cvičení, případně mimo cvičení po dohodě s cvičícím.