Smart Factory - Omo

System simuluje práci továrny na nábytek.

Továrnu jsme postavili pomocí vzoru **Builder**, který pomáhá vytvářet složité objekty krok za krokem. Máme 2 konfigurace továrny, jedna z nich dovoluje postavit továrnu z json souboru. Továrna má výrobní linky na kterých probíhá práce strojů a lidí. V naší továrně pracuje jedna linka (je možnost přidat více linek), která se během simulace 2-krát přeskládá. Linku postavíme tak, že nejdřív vytvoříme stroje a dělníky (instance abstraktní třídy LineItem), které uložíme do listu availableLineItems. Když se zavolá metoda setLineItems, linka se postaví z již uložených strojů a dělníků podle předdefinované sekvence lineItemů. Jestliže se nastane případ, že chybí nějaký stroj čí dělník, do logu se vypíše varování a ukončí se běh aplikace. Mezi produkty továrny patří židle, stůl a skříň.

Každý typ stroje má svou spotřebu a náklady na provoz, ale také opotřebení které se s časem zvyšuje. Pokud se stroj rozbije, automaticky se spustí událost “porucha” a stroj bude přidán do fronty rozbitých strojů, čekajících na opraváře. Pro znovupoužití opravářů je realizován vzor **Object Pool**.

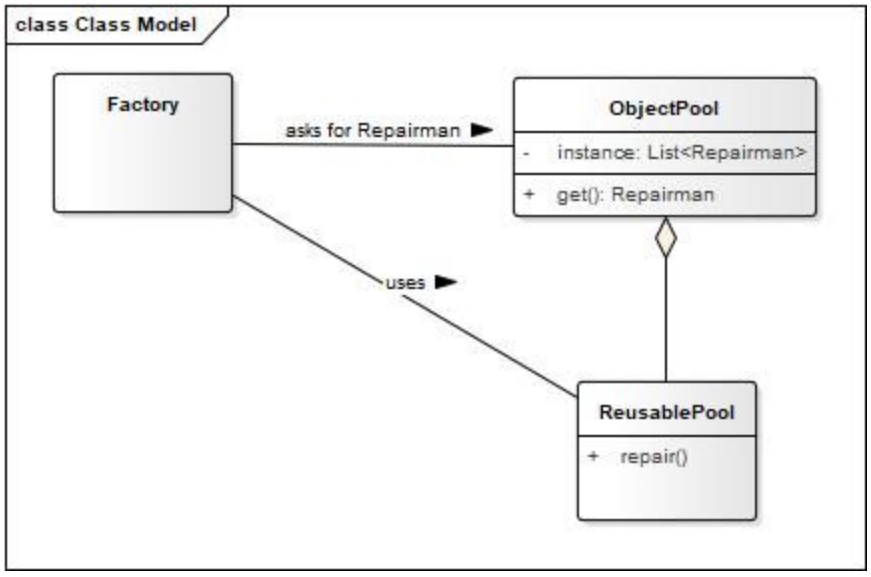
Stroje, lidi a produkty navštěvuje ředitel a inspektor. U těchto tříd je důležité mít dostupnou vždy jen jednu a tu samou instanci, proto používáme vzor **Singleton.** Během návštěvy se názvy provedených akcí zapisují do logu.

Jako *zajímavý softwarový design* jsme použili design pattern **Memento** na ukládání stavů jednotlivých strojů. Pomocí něj můžeme vytáhnout stav stroje v libovolném taktu.

Pro implementaci továrny budeme používat 7 design patternů:

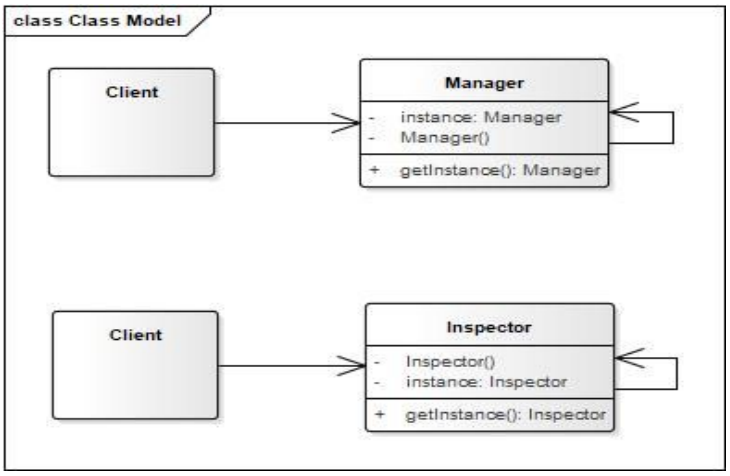
1. Object pool

Tento pattern jsme použili ve třídě RepairStatus. Třída poskytuje instance třídy Repairman a obsahuje 2 pole: available, kde jsou už nainstancovaní opraváři, a working repairmen. Když bude zavolána metoda getRepairman(), třída vezme opraváře z pole available (pokud pole není prázdné) a vloží ho do pole working.



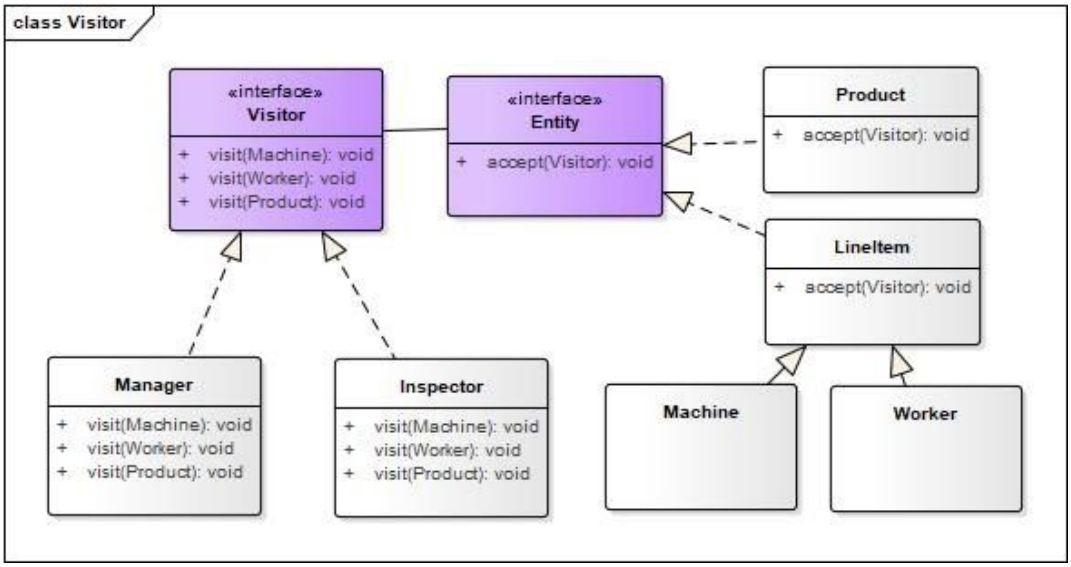
2. Singleton

Tento pattern jsme použili v několika třídách: RepairStatus, Queue, Manager, Inspector. U těchto tříd je důležité mít dostupnou vždy jen jednu a tu samou instanci.



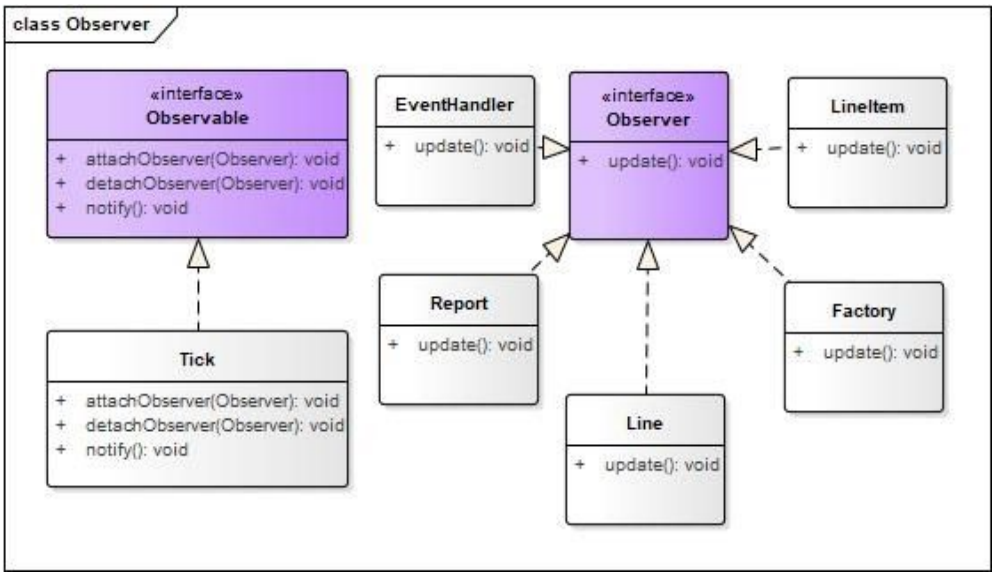
3. Visitor

Pomocí metod visit() a accept() Manažer a Inspector vykonávají různé akce na strojích.
Pokud stroj dosáhne určité míry poškození bude poslán na opravu aby se snížil čas opravy



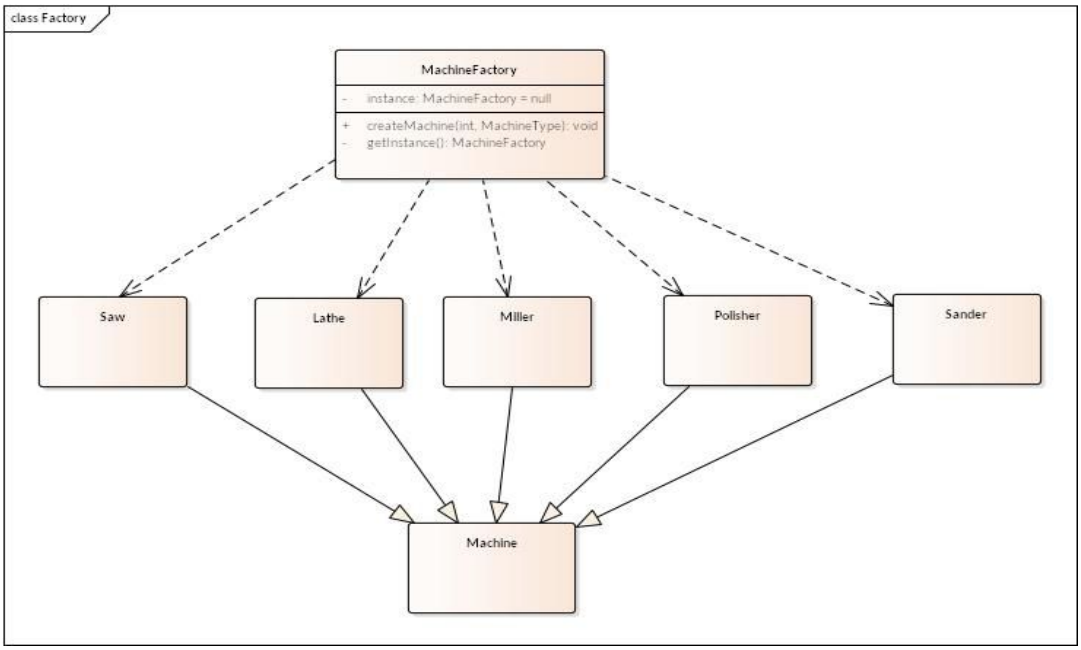
4. Observer

Tick je observable a má metodu notify(). Pomoci observeru kontrolujeme, například, stav stroje v každém taktu a změní-li stav stroje na rozbitý, pošle se event breakdown



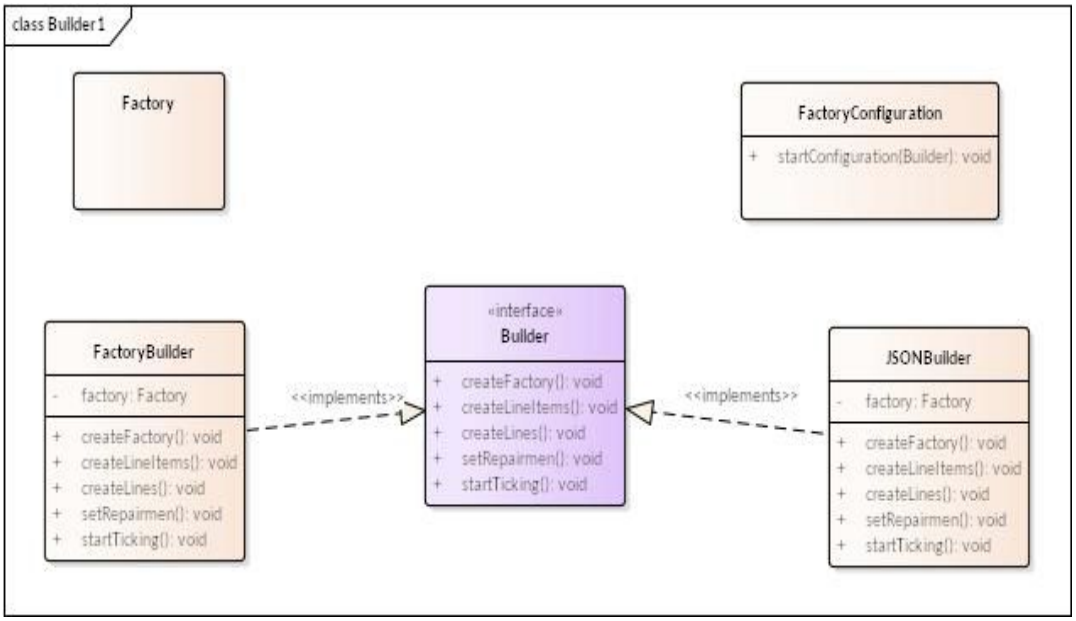
5. Factory

MachineFactory vytváří jednotlivé stroje.



6. Builder

Builder postaví továrnu buď podle defaultní konfigurace a nebo podle konfigurace z JSONu. Nakonec spustí simulaci.



7. Chain of responsibility

Tento design pattern napomáhá nám uchovat pořadí jednotlivých strojů. V abstraktní třídě LineItem máme referenci na další stroj (nextLineItem). Na stroji, který je první na řadě, začínáme rekurzivně volat work.