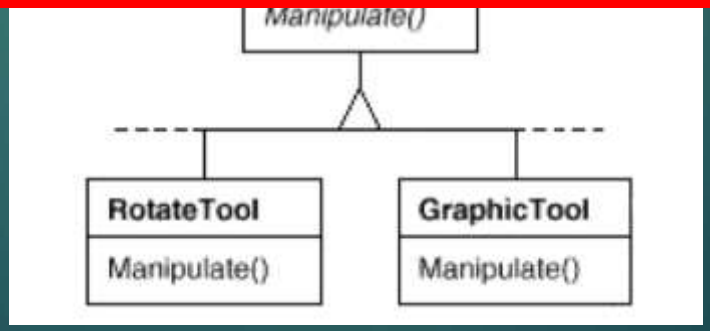
Prototype

Situace

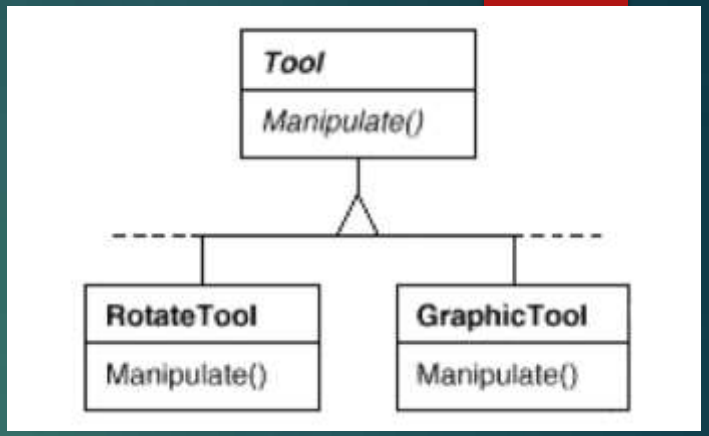
- chceme vytvořit grafický editor pro různé neznámé objekty
  - může to být cokoliv – text, obrázky, „čáry“
  -

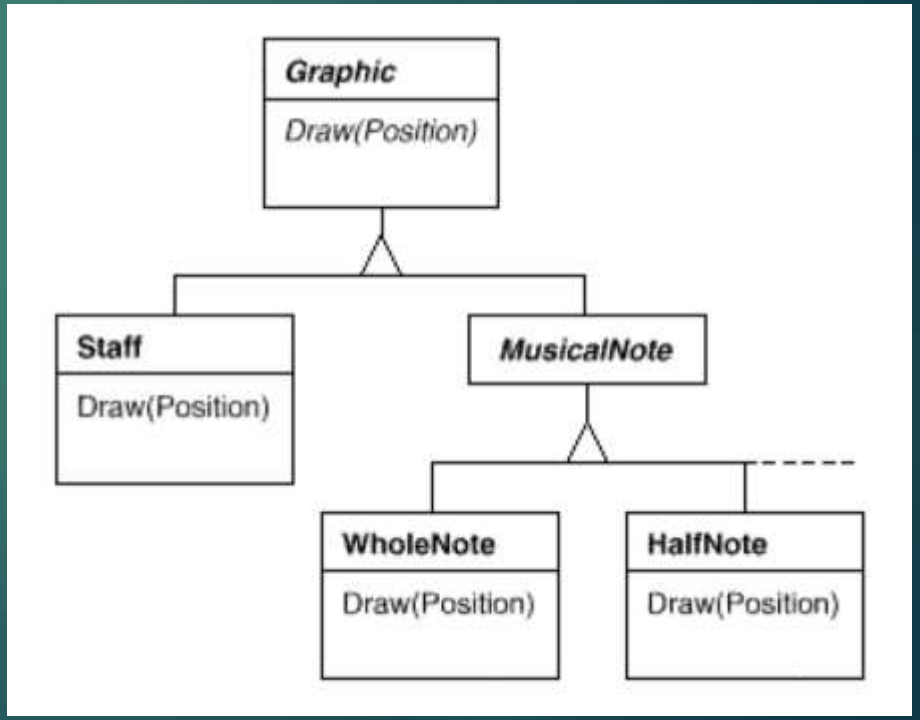
Ne světlé písmo na tmavém pozadí !!!



Situace

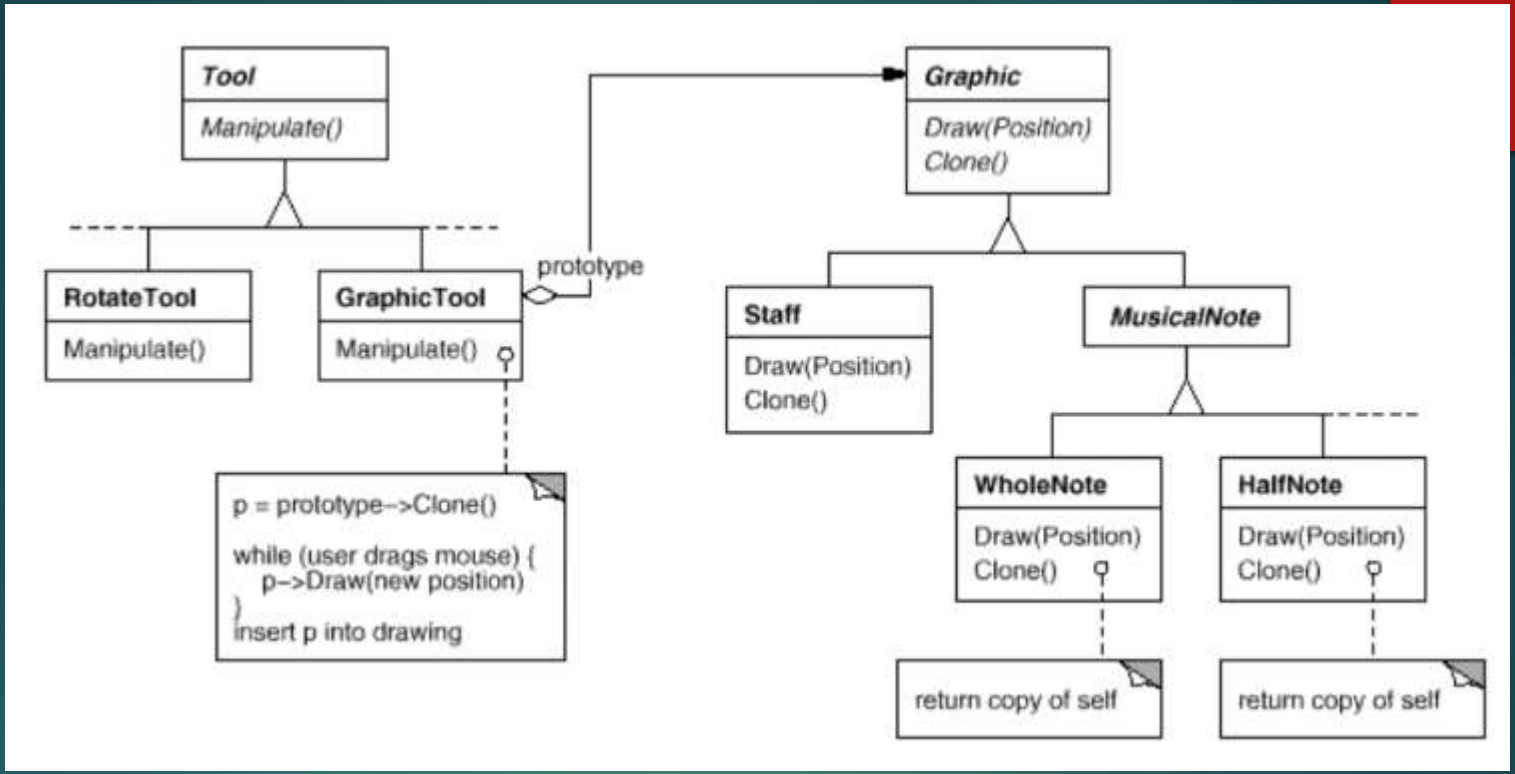
- GraphicTool potřebuje způsob, jak vytvořit objekty, s kterými má pracovat





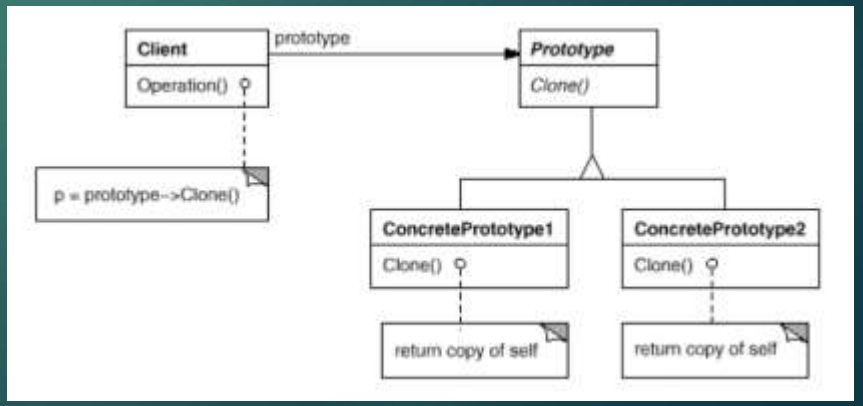
Řešení

- nebudeme vytvářet třídu NěcoGraficTool pro každý druh prvků (WholeNoteGT, HalfNoteGT,..)
- vytvoříme jenom jednu třídu a jednotlivé instance inicializujeme instancí prvku (WholeNote, HalfNote,...), který budeme kopírovat
- taková instance se jmenuje prototype



Účastníci:

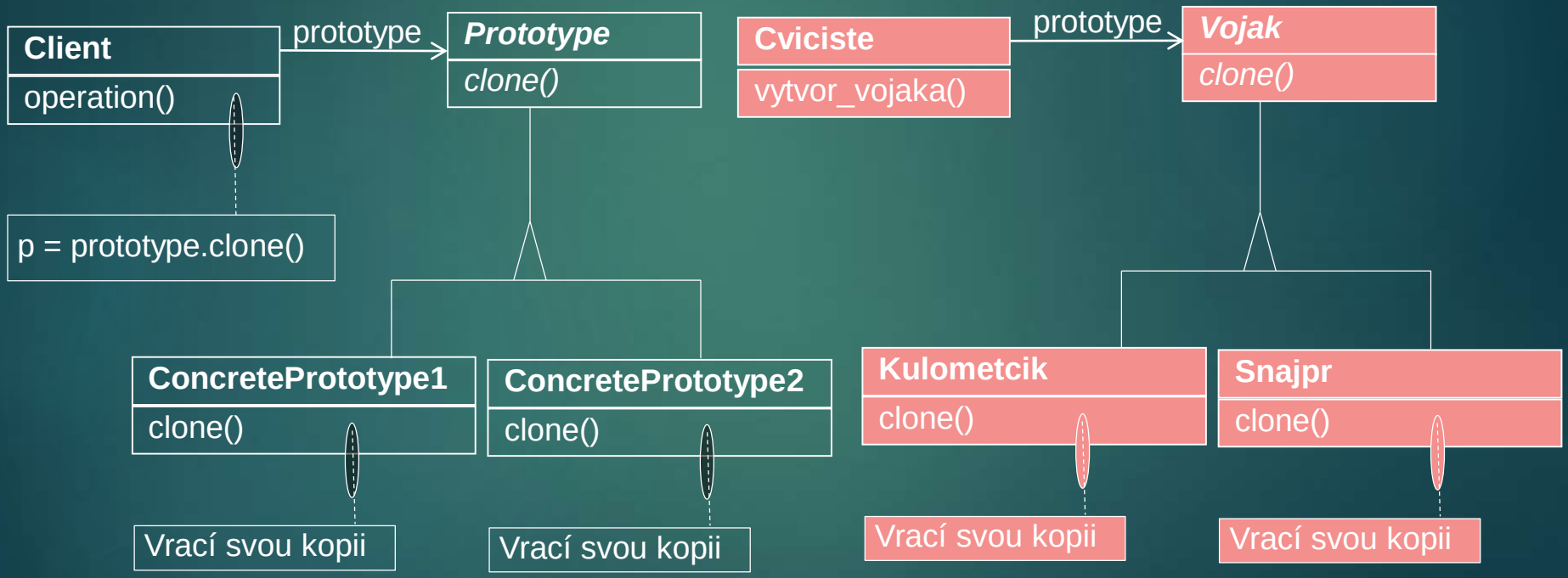
- Prototype (abstraktní třída)
- ConcretePrototype (podtřídy)
- Client



Popis Prototype

- „tvořivý“ (creational) návrhový vzor
- systém (Client) je nezávislý na produktech (objektech) s kterými pracuje
  - nezná vnitřní strukturu objektů, jenom veřejný interface
- systém neví, kterou konkrétní třídu používá
  - zná jenom (abstraktního) předka této třídy
- podstatou je schopnost prototypu vytvářet svou kopii
  - `prototype->clone();`

Popis Prototype



Popis Prototype

```
class Cviciste {
        public Vojak vytvor_vojaka() {
                if (energieCviciste ≥ 50)
                        return kulometcik.clone();
                else
                        return snajpr.clone();
        }
        private Vojak kulometcik = new Kulometcik(); // prototyp
        private Vojak snajpr = new Snajpr(); // prototyp
}
class Vojak {
        public abstract Vojak clone();
}
class Kulometcik extends Vojak {
        @Override
        public Vojak clone() {
                // vytvori kopii sebe sama
        }
}
class Snajpr extends Vojak {
        @Override
        public Vojak clone() {
                // vytvori kopii sebe sama
        }
}
```

Popis Prototype

- Poznámky:
  - pokud klonování je používáno v programu i jinde pro jiné účely, máme Prototype implementovaný „zadarmo“
- Někdy se používá
  - katalog prototypů (**prototype manager**)
    - prototypy se registrují v katalogu (pod nějakým identifikátorem)
    - klienti používají pro klonování prototypy z katalogu (a vytvářejí nové instance)
  - metoda clone() nemusí provádět „deep copy“
    - ve některých případech stačí „shallow copy“ nebo nějaká jejich kombinace
      - musíme si rozmyslet, co mohou klony sdílet a co ne
- Nevýhody:
  - obtížné implementovat clone() metody do již existující hierarchie tříd
  - obtížné implementovat clone() metodu pro třídy s nekopírovatelnými elementy nebo s cyklickými referencemi

Situace, kdy se hodí (1/5)

- Situace:
  konstrukce objektu vyžaduje netriviální práci (z hlediska velikosti kódu)
  - např. při použití návrhových vzorů Decorator nebo Composite
- Problém: chceme-li více totožných objektů
- Některé možnosti řešení:
  - Factory Method / Abstract Factory
    - máme metodu, která provede konstrukční kód
  - Prototype
    - implementujeme metody Clone()
    - výhody oproti továrnám:
      - konstrukce objektu je na jednom místě, není nutné upravovat ještě továrnu
      - vytvořit nové typy objektů lze v run-time (vytvořit novou továrnu v run-time obecně nelze)
  - Builder
    - Výhoda: flexibilnější
    - Nevýhoda: Vytvoření pomocí builderu je "ukecanější" (nutno specifikovat parametry, v případě factory/prototype necháváme specifikaci parametrů na nich)

Situace, kdy se hodí (2/5)

- Situace:
  chceme/máme mnoho druhů instancí nějaké třídy
  - např. mnoho potomků třídy nebo mnoho druhů různých parametrizací této třídy
- Příklad situace:
  - např. vytváříme hru, v níž je velké množství zbraní, které jsou různě účinné a mají různý dosah
- Některé možnosti řešení:
  - druh instance = nová dědící třída
    - nevýhody
      - nepraktické, může-li se počet druhů instancí měnit
      - nepoužitelné, mohou-li se druhy instance generovat v run-time
  - druh instance = sada parametrů hlavní třídy (nejsou zde žádné dědící třídy)
    - výhoda: můžeme se zbavit toho velkého množství tříd, stačí nám jen jedna
    - nevýhoda: špatně se s tím pracuje
  - Prototype
    - pro každý druh třídy vytvoříme prototyp
    - výhody:
      - flexibilní řešení
      - můžeme se zbavit toho velkého množství tříd, stačí nám jen jedna
        - unikátní vlastnost návrhového vzoru Prototype

Situace, kdy se hodí (3/5)

- Situace:
  „třídy se vytvářejí dynamicky v run-time“
- Příklad situace:
  - např. procedurálně generujeme různé druhy jednotek do nějaké hry
- Problém: chceme vytvářet instance nějaké této třídy
- Řešení:
  - Prototype
    - každou instanci vytvoříme jen jednou a uložíme si ji jako prototyp
    - toto je něco, na co jiné návrhové vzory pro vytváření instancí neposkytují řešení
    - Příklad

Situace, kdy se hodí (4/5)

- Situace:
  „je nutné duplikovat hierarchii tříd jak u těchto tříd, tak u jejich továren“
- Příklad situace:
  - např. Animal má potomky Dog a Cat, AnimalFactory má potomky DogFactory a CatFactory
- Problém: špatně se modifikuje hierarchie tříd
- Řešení:
  - Prototype
    - máme prototypy, nepotřebujeme továrny

Situace, kdy se hodí (5/5)

- Situace:
  „konstrukce objektu je netriviální (z hlediska časové efektivity)“
- Příklad situace:
  - např. při konstrukci objektu je nutná práce se souborem
- Problém: chceme se vyhnout každé další konstrukci objektu „od nuly“
- Řešení:
  - Prototype
    - první konstrukci provedeme obvykle, každou další již pomocí klonování prototypu
      - předpokládáme, že při dalších konstrukcích nebude třeba se souborem pracovat (např. je již načtený a jeho obsah je součástí prototypu)

Obecné vztahy k jiným návrhovým vzorům

- Abstract Factory / Factory Method
  - Prototype a Abstract Factory a Factory Method slouží podobným účelům
    - Prototype ale může někdy nabídnout více (viz předchozí slajdy)
- Composite, Decorator
  - Prototype může sloužit pro uchování vytvořených kompozitů
- Singleton
  - Prototype Manager může být řešen pomocí Singletonu

Známá použití

- A long, long time ago in a galaxy far, far away… Impérium použilo návrhový vzor Prototype pro výrobu Storm Trooperů

- Existují programovací jazyky, kde se objekty nevytvářejí obvyklým způsobem ale pomocí klonování prototypů
  - Self
    - živý jazyk (zatím poslední verze 2017)
    - objektově-orientovaný, dynamicky typovaný, JIT
      - některé JIT techniky byly zde nasazeny poprvé
    - prototypovaný přístup k objektům
      - vytvoříme nějaké objekty systémem „přidat element“, poté objekty klonujeme
      - žádné třídy
  - Omega