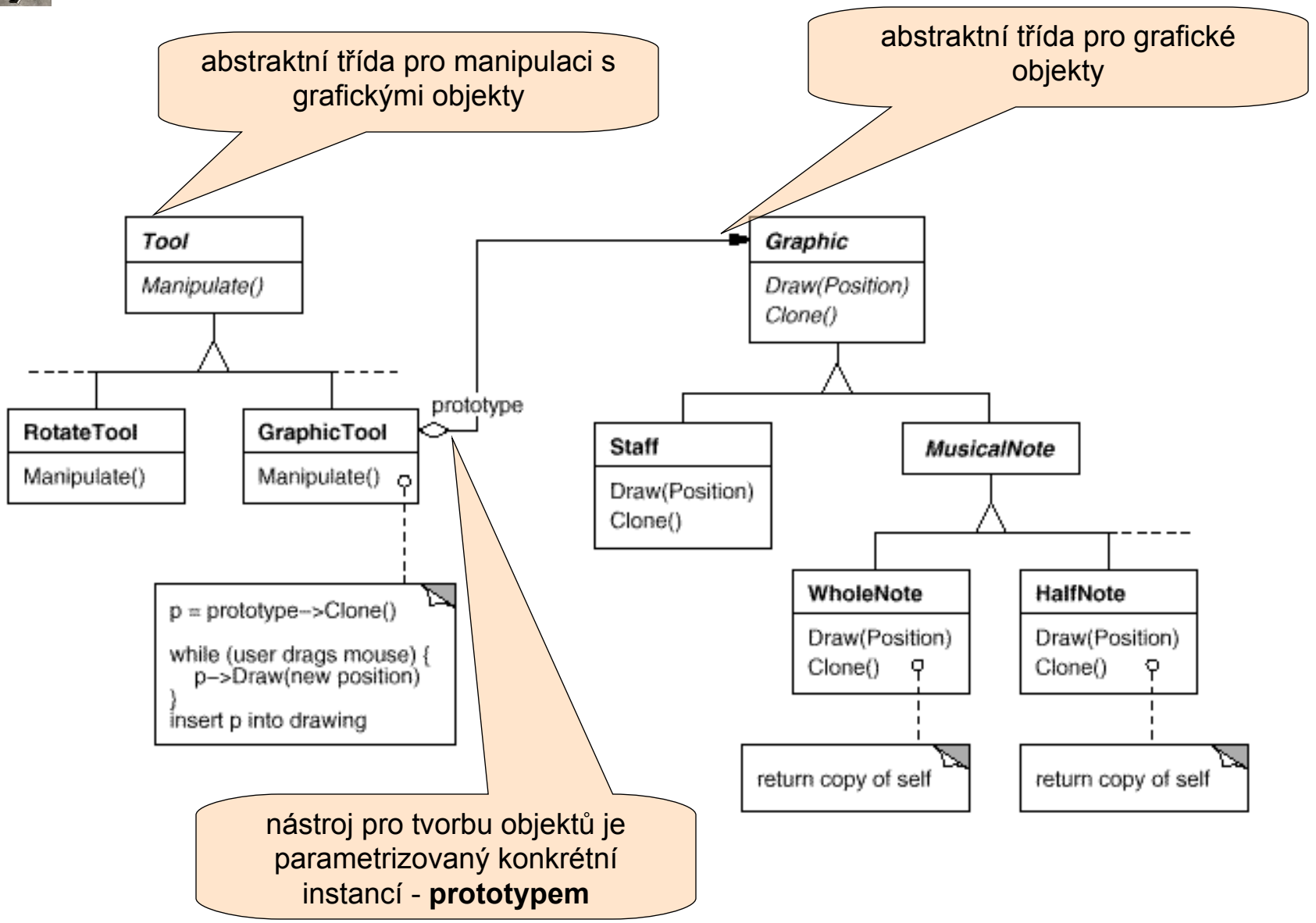
Prototyp

Prototyp

- **Účel**
  - kategorie: vytvářecí pro objekty *(class creational)*
  - specifikuje vytvářené objekty pomocí prototypické instance
  - nový objekt vytváří kopírováním prototypu
- **Příklad**
  - obecný systém pro tvorbu grafických editorů – abstraktní třídy `Graphic`, `Tool` a její podtřída `GraphicTool`, která vytváří grafické komponenty typu `Graphic`
  - úpravou chceme získat editor hudebních partitur (noty, pomlčky, osnovy jako podtřídy `Graphic`)
  - problém – jak vytvářet objekty konkrétních tříd pro noty, pomlčky a osnovy obecnou třídou `GraphicTool`?
  - řešení: `GraphicTool` vytvoří novou třídu `Graphic` pomocí „klonování“ (kopírování) instance podtřídy `Graphic` – **prototyp**
  - `GraphicTool` je parametrizovaný tímto **prototypem,** který se umí sám naklonovat = musí implementovat metodu `Clone()`

Prototyp – příklad



Prototyp – použití

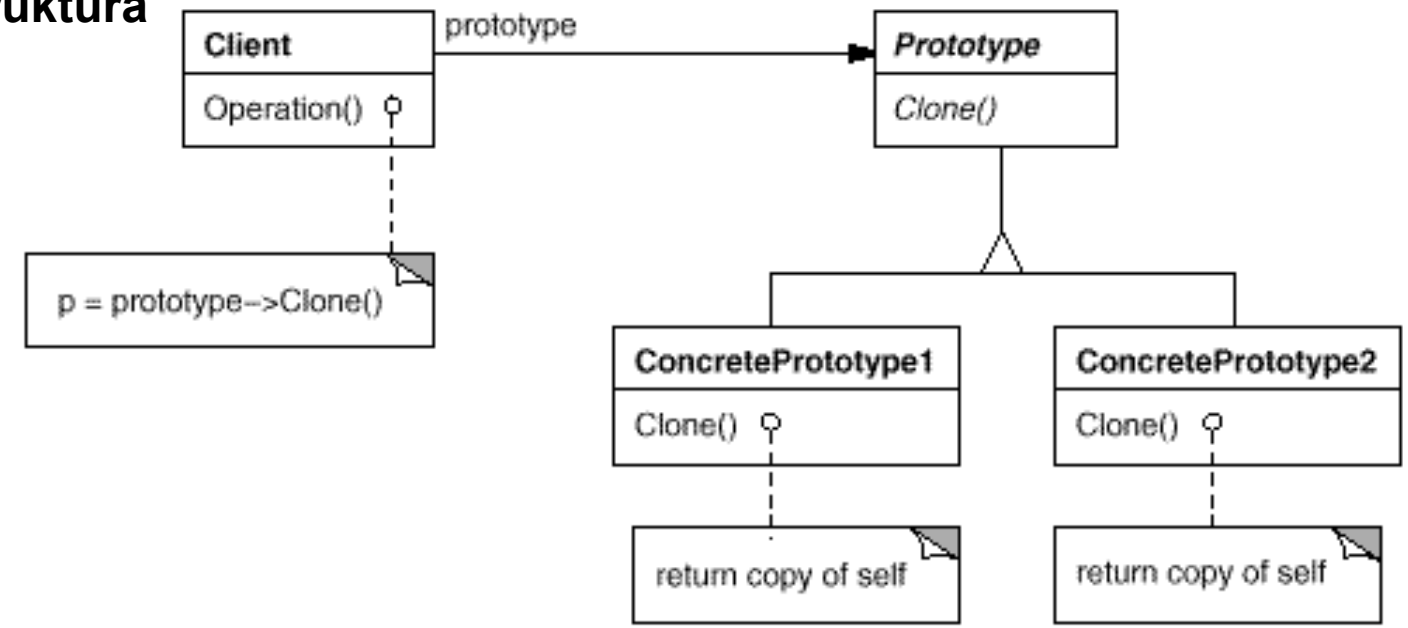
- **Použití**

Systém má být nezávislý na vytváření, skládání a zobrazovaní produktů a

- pokud jsou **třídy**, kterých instance se mají vytvořit, **specifikované za běhu**, např. pomocí dynamického načítání, nebo
- pokud se chceme vyhnout budování **hierarchie tříd továren**, která je souběžná s hierarchií tříd produktů, nebo
- pokud se **instance** třídy může nacházet jen **v několika málo stavech**,
- je **výhodnější** nainstalovat odpovídající počet prototypů a ty **klonovat**, jako za každým **vytvářet instance** tříd s příslušným stavem

Prototyp – struktura

- **Struktura**



- **Účastníci**
  - `Prototype` (`Graphic`)
    - deklaruje rozhraní pro vlastní klonování
  - `ConcretePrototype` (`Staff`, `WholeNote`, `HalfNote`)
    - implementuje operace pro vlastní klonování
  - `Client` (`GraphicTool`)
    - vytváří nový objekt požádáním prototypu aby se naklonoval

Prototyp – důsledky

- Skrývá
  - konkrétní **produktové třídy** před klientem a tím snižuje počet názvů, o kterých klienti vědí
- Umožňuje klientovi:
  - přidávat a odebírat produkty za běhu
  - specifikovat nové objekty pomocí změn hodnot existujících objektů
  - specifikace nových objektů změnou struktury
  - vytváření menšího počtu podtříd
  - dynamická konfigurace aplikace pomocí tříd
- Nevýhoda
  - každá podtřída musí implementovat `Clone()`, může být někdy obtížné (cyklické odkazy, třídy bez copy-konstruktorů)

Prototyp – implementace

- Třeba brát v úvahu:
  - použití správců prototypů
    - počet prototypů v systému není pevný,
      můžou se vytvářet a odstraňovat
    - správce prototypů – asociační sklad;
      vrací, registruje prototyp odpovídající zadanému klíči
  - implementace operace `Clone`
    - obtížné (kruhové odkazy)
    - většina jazyků poskytuje podporu pro kopírovaní objektů
      - např. C++ copy konstruktor
    - „mělká vs. hluboká“ kopie (třída obsahuje ukazatel)
    - `save` a `load` objektů – jednoduchá implementace `Clone`

Prototyp – implementace

- inicializace klonů
  - někteří klienti jsou s objektem úplně provázaní,
    některé potřebují objekty inicializovat podle svých požadavků
  - inicializace nemůže probíhat obecně přes `Clone()`,
    různý počet inicializačních parametrů
  - řešení
    - operace pro nastavení stavu objektu
    - metoda `Initialize()`,
      při „hluboké“ kopii operací `Clone()` může být nutné odstranit kopie

Prototyp – příklad

- nadefinujme podtřídu `MazePrototypeFactory` třídy `MazeFactory`, kterou nainicializujeme pomocí prototypů objektů, které má vytvářet
- ušetřili jsme vytváření jejích podtříd pro každý druh stěn nebo místností, které chceme vytvářet
- podtřída rozšiřuje rozhraní pomocí konstruktoru, který má **prototypy jako argumenty**

```
class MazePrototypeFactory : public MazeFactory {
public:
  MazePrototypeFactory(Maze*, Wall*, Room*,
Door*);
  virtual Maze* MakeMaze() const;
  virtual Room* MakeRoom(int) const;
  virtual Wall* MakeWall() const;
  virtual Door* MakeDoor(Room*, Room*) const;
private:
  Maze* prototypeMaze_;
  Room* prototypeRoom_;
  Wall* prototypeWall_;
  Door* prototypeDoor_;
};
```

```
MazePrototypeFactory::MazePrototypeFactory (
  Maze* m, Wall* w, Room* r, Door* d) {
    prototypeMaze_ = m;
    prototypeWall_ = w;
    prototypeRoom_ = r;
    prototypeDoor_ = d;
}
```

Prototyp – příklad

- jednoduché metody pro tvorbu (stěn, místností, dveří)
  - klonují prototyp a potom ho inicializují

```
Wall* MazePrototypeFactory::MakeWall () const {
  return prototypeWall_->Clone();
}

Door* MazePrototypeFactory::MakeDoor (Room* r1, Room *r2) const {
  Door* door = prototypeDoor_->Clone();
  door->Initialize(r1, r2);
  return door;
}
```

- bludiště vytvoříme pomocí `MazePrototypeFactory` inicializovanou prototypy základních komponent bludiště

```
MazeGame game;
MazePrototypeFactory simpleMazeFactory(
  new Maze, new Wall, new Room, new Door
);
Maze* maze = game.CreateMaze(simpleMazeFactory);
```

Prototyp – příklad

- změna typu bludiště – inicializovat `MazePrototypeFactory` pomocí jiné sady prototypů

```
MazePrototypeFactory bombedMazeFactory(
  new Maze, new BombedWall, new RoomWithABomb, new Door);
```

- objekt, který chceme použít jako prototyp, např. instance `Door`, musí podporovat operaci `Clone`, případně operaci `Initialize`

```
class Door : public MapSite {
public:
  Door();
  Door(const Door&);

  virtual void Initialize(Room*, Room*);
  virtual Door* Clone() const; virtual void Enter();
  Room* OtherSideFrom(Room*);
private:
  Room* room1_;
  Room* room2_;
};
```

```
Door::Door (const Door& other) {
  room1_ = other.room1_;
  room2_ = other.room2_;
}

void Door::Initialize (Room* r1, Room* r2) {
  room1_ = r1;
  room2_ = r2;
}

Door* Door::Clone () const {
  return new Door(*this);
}
```

Prototyp – příklad

- podtřída `BombedWall` musí operaci `Clone` přetížit a implementovat příslušný kopírovací konstruktor

```
class BombedWall : public Wall {
public:
  BombedWall();
  BombedWall(const BombedWall&);

  virtual Wall* Clone() const;
  bool HasBomb();
private:
  bool bomb_;
};
```

```
BombedWall::BombedWall (const BombedWall& other) : Wall(other) {
  bomb_ = other.bomb_;
}

Wall* BombedWall::Clone () const {
  return new BombedWall(*this);
}
```

- Klienti klonující prototyp neví o jeho konkrétní podtřídě
  - např.: `BombedWall::Clone()` vrací `Wall*`,
  - ale implementace vrací ukazatel na novou instanci podtřídy (`BombedWall*`)

Prototyp – související NV

- *Abstract Factory*
  - některými způsoby si konkurují
  - společné použití: abstraktní továrna může ukládat sadu prototypů, ze kterých se mají klonovat a vracet produktové objekty

- Návrhy používající vzory *Composite* a *Decorator*, můžou často a s výhodou používat i vzor prototyp