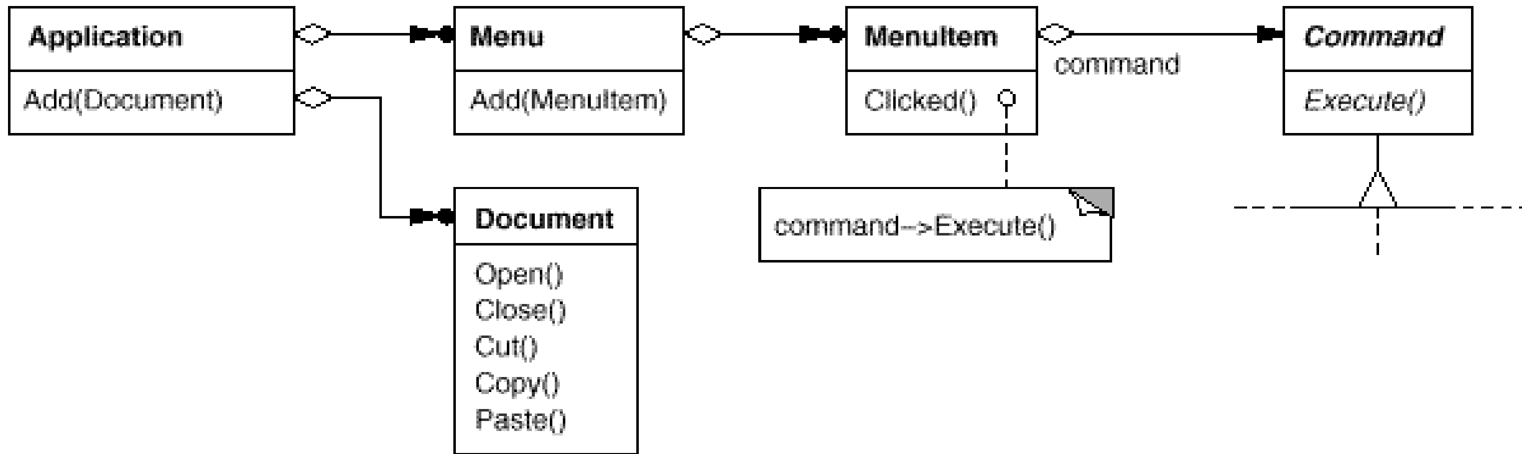
Command

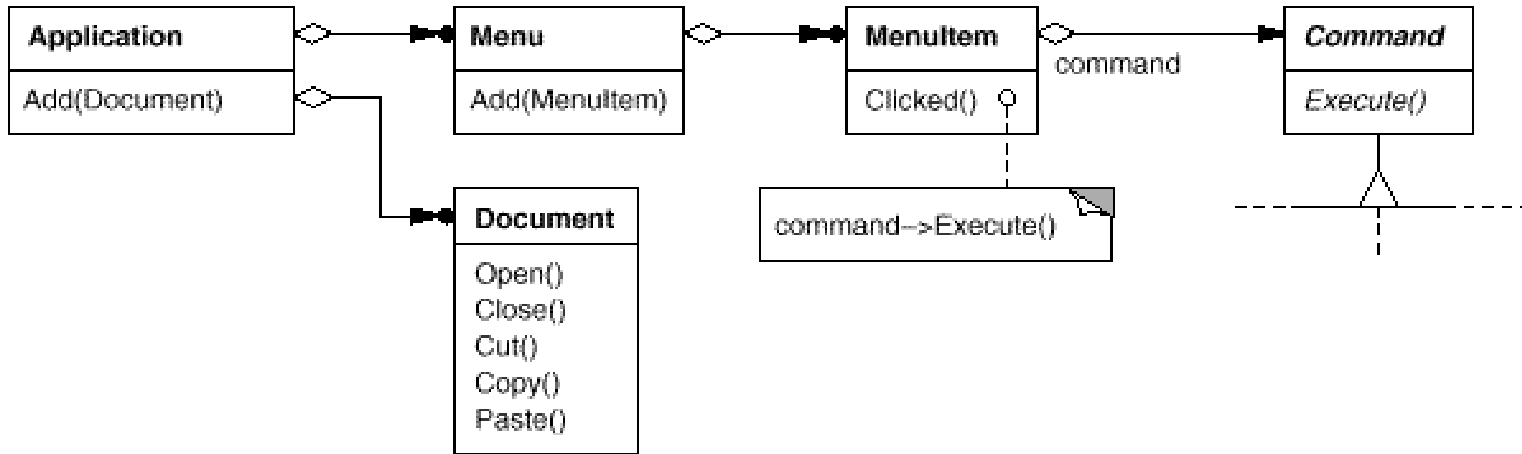
Command

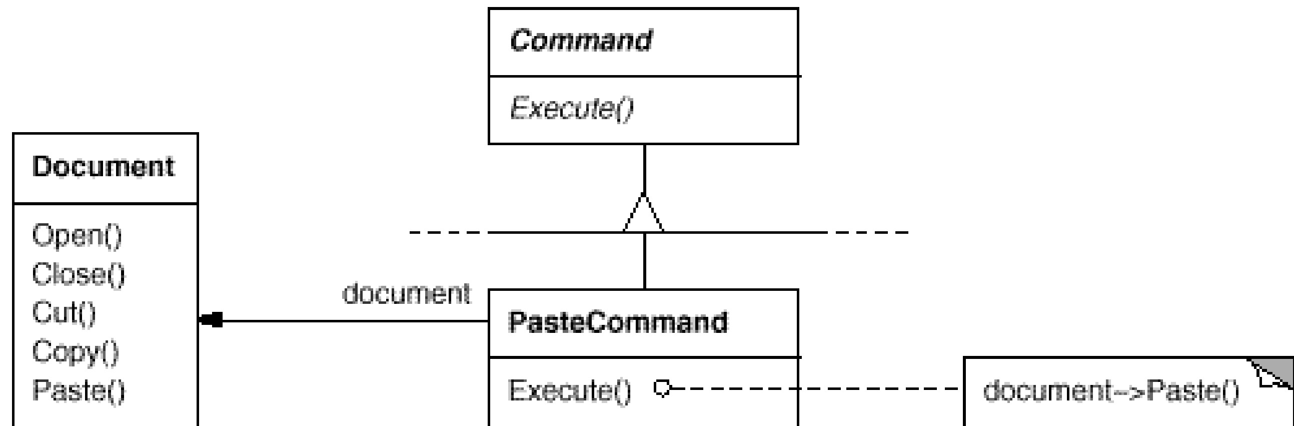
- **Známý jako**
  - Action, Transaction
- **Účel**
  - zapouzdřit požadavek do objektu
  - oddělit požadavek od jeho vykonání
  - možnost vyvolat metodu bez znalosti cílového objektu nebo konkrétní metody
- **Motivace**
  - grafické toolkity (menu...), undo, logování, nahrávání maker, wizardy…

Command – „učebnicový příklad“

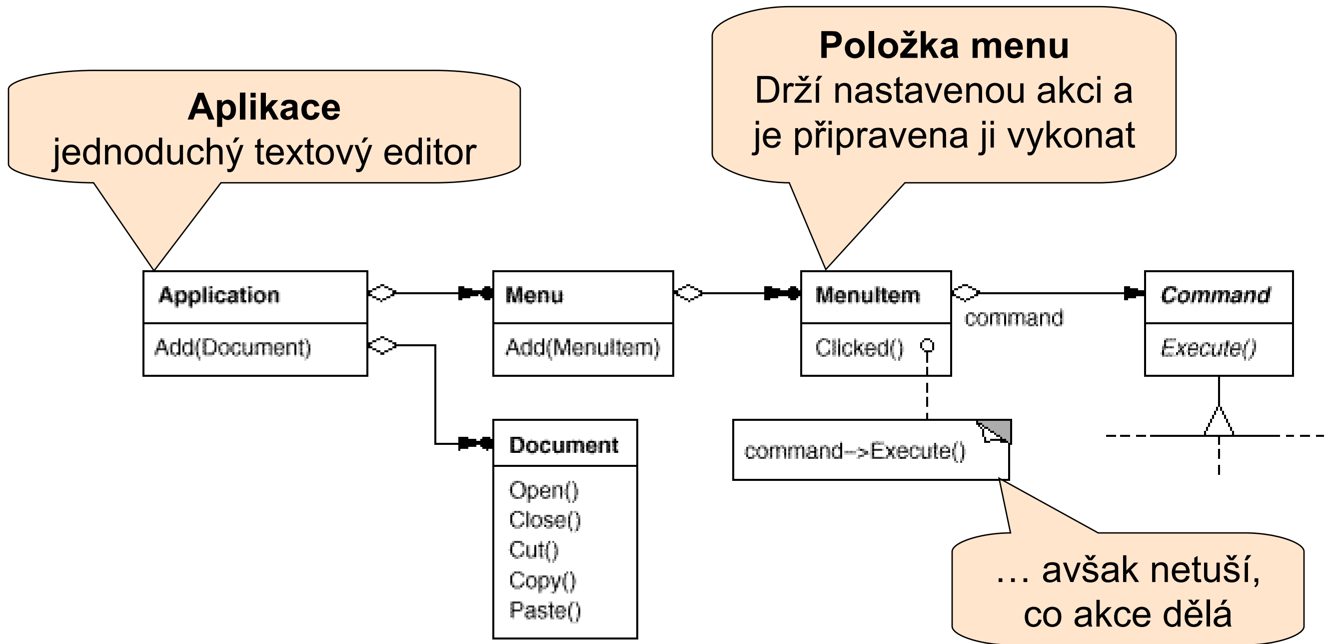


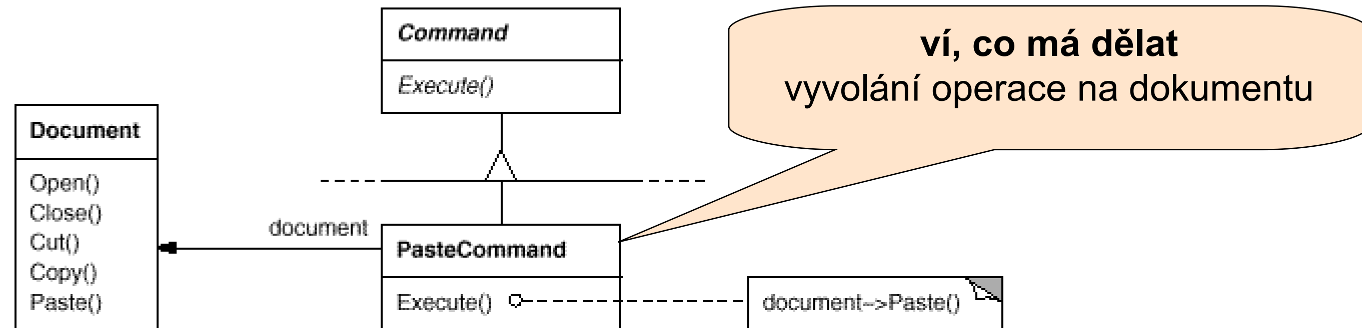
Command – „učebnicový příklad“





Command – „učebnicový příklad“





Command – „učebnicový příklad“

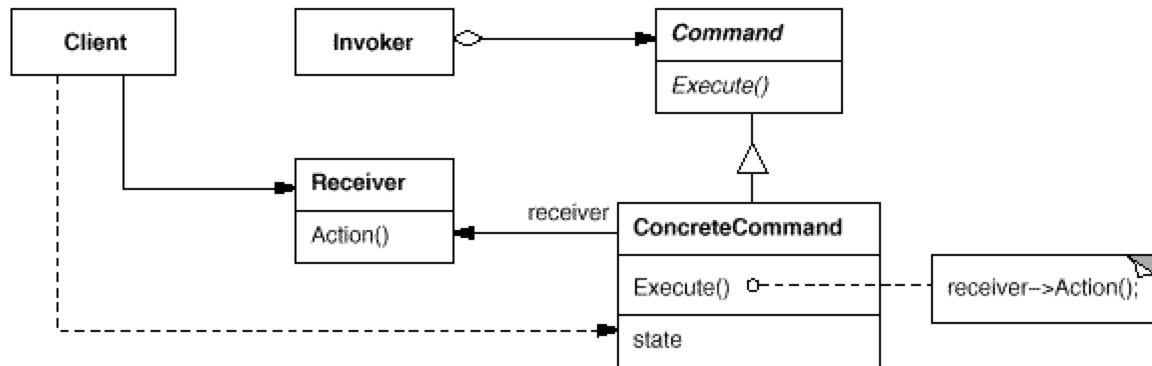
```
public abstract class Command
{
   public abstract void Execute();
}


public class PasteCommand : Command
{
    private Document _document;

    public PasteCommand(Document doc)
    {
        _document = doc;
    }
    public override void Execute()
    {
          _document.Paste();
    }
}
```

Drží referenci na cílový objekt.

Implementuje konkrétní akci

Command – obecná struktura

- **Struktura**



- **Účastníci**
  - *`Command`*
    - abstraktní reprezentace akce
  - `ConcreteCommand (PasteCommand, OpenCommand)`
    - definuje, jak akci vykonat
  - `Client (Application)`
    - vytváří `ConcreteCommand` a nastavuje mu příjemce
    - nastavuje účastníkovi `Invoker` konkrétní instanci `ConcreteCommand`
  - `Invoker (MenuItem)`
    - vyvolává požadavky
  - `Receiver (Document, Application, …)`
    - implementuje operace nutné pro vyřízení požadavku

Command – sekvenční diagram

- **Interakce mezi objekty**
  - ilustrace oddělení objektu vyvolávajícího akci (invoker) od příjemce (receiver)

aReceiver aClient aCommand anInvoker

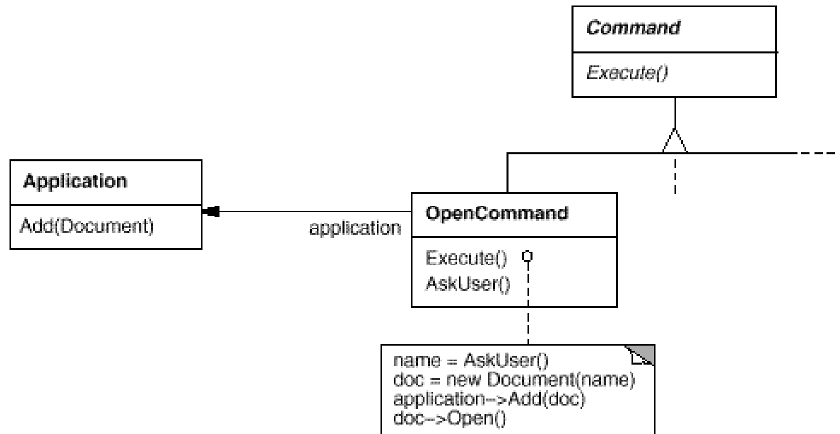
new Command(aReceiver)

StoreCommand(aCommand)

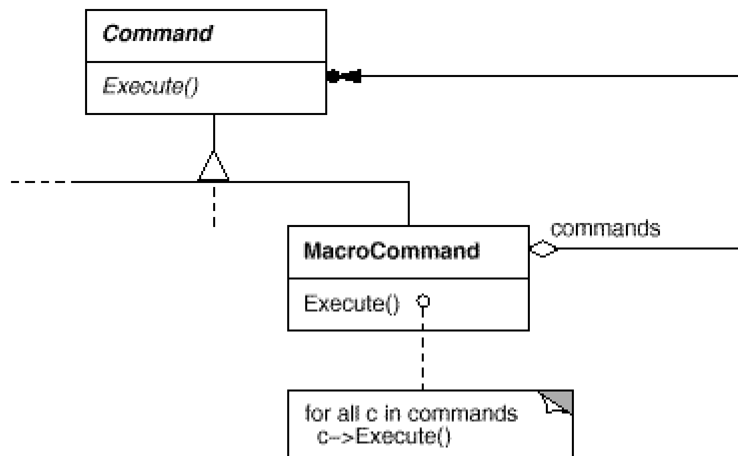
Execute()

Action()

Command – „učebnicový příklad“ - rozšíření



- **Složitější logika v rámci Execute**
  - volání více metod receiveru
  - dialogové okno...



- **Kompozice**
  - posloupnosti volání (makra)
  - Composite pattern
  - Composite objekt nemá receiver

Command – undo

```
public abstract class Command
{
    public abstract void Execute();
    public abstract void Undo();
}

public class PasteCommand : Command
{
    private Document _document;
    string _oldText;

    public PasteCommand(Document doc)
    {
        _document = doc;
    }
    public override void Execute()
    {
         _oldText = _document.Text;
            _document.Paste();
    }
    public override void Undo()
    {
        _document.Text = _oldText;
    }
}
```

Uloží stav

Obnoví stav z uložených hodnot

Command – undo, redo

- **Je třeba ukládat stav receiveru**
  - reference na samotný objekt receiveru
  - argumenty volání Command objektu
  - původní hodnoty receiveru
- **Command objekty občas musí být kopírovány**
  - pokud se dále může měnit jejich stav
  - Prototype pattern
- **Pozor na chyby při undo/redo**
  - zajistit konzistentní stav uložených command objektů (Memento)

Command – logování, transakce

```
public abstract class Command
{
    public abstract void Execute();
    public abstract void Undo();
    public abstract void Store();
    public abstract void Load();
}

public class PasteCommand : Command
{
    private Document _document;
    string _oldText;

    ...
}
```

Uložení, načtení z disku.
Možnost zpětné aplikace provedených změn

Command – detaily

```
private void menuItem_Click(MenuItem sender)
{
    Command clickedItemCommand = sender.Command;

    clickedItemCommand.Execute();

    _undoManager.AddCommand(clickedItemCommand.Clone());
    _macroRecorder.AddCommand(clickedItemCommand.Clone());

    clickedItemCommand.Store();
}

...

_undoManager.Undo();
_undoManager.Redo();

Command recordedMacro = _macroRecorder.GetRecorderMacro();
```

Prototype pattern
Je třeba kopírovat pokud se mění stav.

Builder pattern

Command – odlišné použití

```
private void menuItem_Click(MenuItem sender)
{
     Command clickedItemCommand = sender.Command;
     clickedItemCommand.Execute();
     ...
}
```

Command – odlišné použití

```
private void menuItem_Click(MenuItem sender)
{
    Command clickedItemCommand = sender.Command;
    clickedItemCommand.Execute();
    ...
}
```

Commant pattern často uváděn takto.
Umožňuje předávat aktuální parametry

vs.

```
private void menuItemPaste_Click(MenuItem sender)
{
    Command pasteCommand =
            new PasteCommand(_document, _currentPasteStyle);
    pasteCommand.Execute();
    ...
}
```

Command – implementace pomocí šablon

```
template <class Receiver>
class SimpleCommand : public Command {
public:
    typedef void (Receiver::* Action)();

    SimpleCommand(Receiver* r, Action a): _receiver(r), _action(a) { }

    virtual void Execute() {
        (_receiver->*_action)();
    }
private:
    Action _action;
    Receiver* _receiver;
};

MyClass* receiver = new MyClass;
// ...
Command* aCommand = new SimpleCommand<MyClass>(receiver, &MyClass::Action);
// ...
aCommand->Execute();
```

Command – známé použití

- **Undo**
- **Transakční chování**
- **Makra**
- **GUI toolkity**
  - Java Swing – interface Action, metoda actionPerformed (~Execute)
- **ThreadPool**
  - odkládání požadavků do fronty - zpracování, až na ně přijde řada
- **Předávání požadavků po síti**
- **Wizardy**
  - celý wizard jako jeden Command objekt
- **Paralelní výpočty**

Command – související vzory

- **Composite**
  - vytváření maker (MacroCommand)
- **Memento**
  - uchovávání stavu objektů pro případné vrácení akce (undo)
- **Prototype**
  - pokud potřebujeme do historie akcí pro Undo ukládat kopie akcí