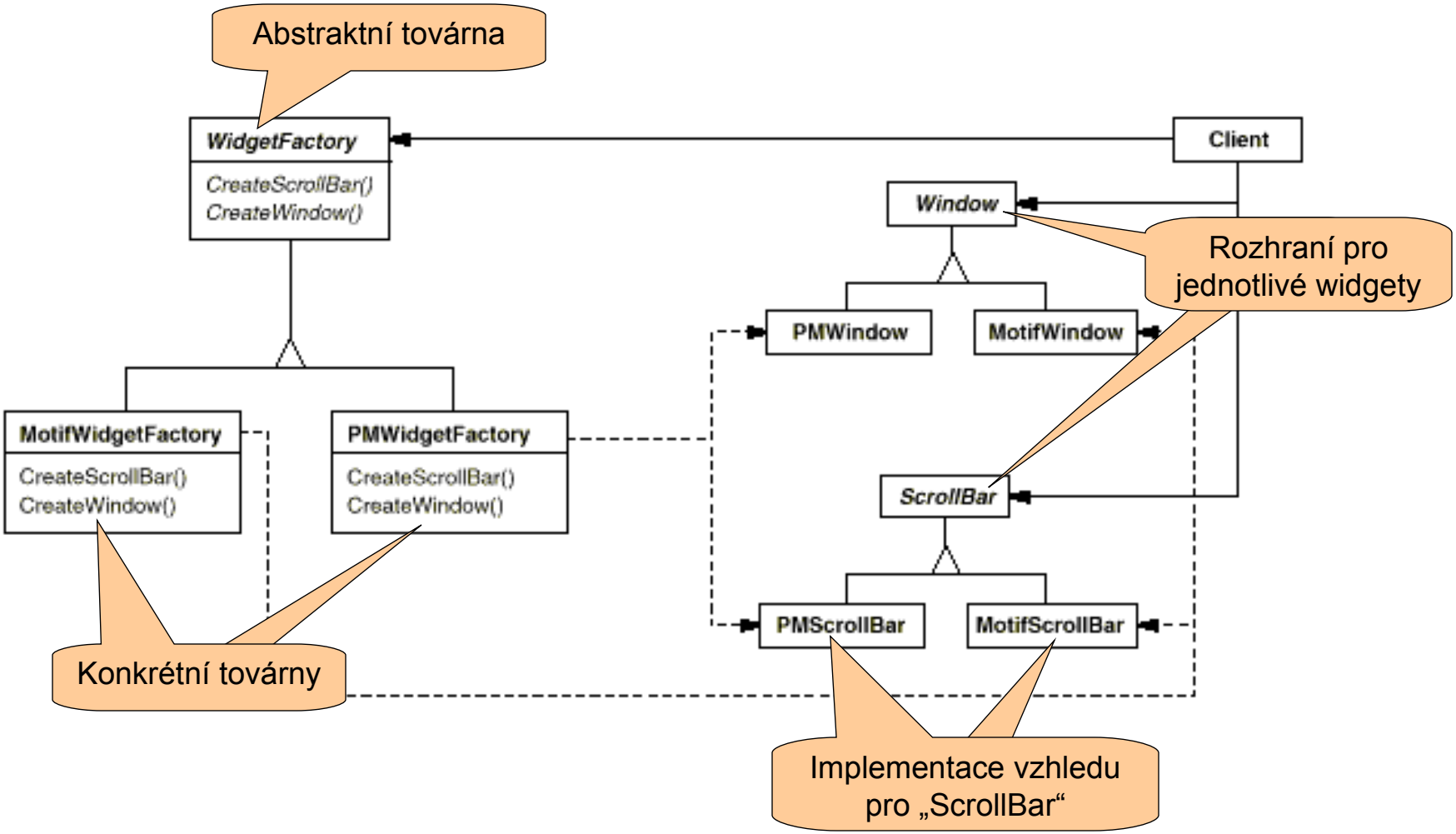
Abstract Factory

Abstract Factory – úvod

- **Problém**
  - potřebujeme vytvářet objekty ze skupiny souvisejících tříd
  - skupin je více, chceme je snadno vyměňovat
  - můžeme používat (v jednom kontextu) pouze jednu skupinu – konzistence
- **Příklad: widgety**
  - toolkit pro vytváření GUI s různými “look and feel“
  - WidgetFactory – vytváří okna, tlačítka, …
  - konkrétní skupina do které objekt patří není důležitá (PMButton vs. MotifButton)
  - pozn: PM = Presentation Manager – GUI v systému OS/2
- **Jak na to?**
  - izolovat aplikaci od konkrétních typů - zbavit se operátoru `new`
  - vytvářet objekty pouze pomocí **továrny** → možnost kontroly z jednoho místa
  - využít polymorfismus – přístup k továrně přes **abstraktní** rozhraní
- **Známý také jako Kit**

Abstract Factory – příklad



Abstract Factory – účastníci

- **Abstraktní produkty (`Window`, `ScrollBar`)**
  - rozhraní objektů vytvářených v továrně
  - viditelné pro aplikaci
- **Abstraktní továrna (`WidgetFactory`)**
  - rozhraní pro vytváření objektů
  - vrací „abstraktní“ objekty
  - viditelné pro aplikaci
- **Klient**
  - používá jenom rozhraní abstraktních produktů a továrny
- **Konkrétní produkty (`MotifWindow`, `MotifScrollBar`)**
  - skutečné objekty, neviditelné pro aplikaci
  - implementují rozhraní jednotlivých abstraktních produktů
- **Konkrétní továrny (`MotifWidgetFactory`, `PMWidgetFactory`)**
  - vytváří skutečné objekty, neviditelné pro aplikaci
  - implementují rozhraní abstraktní továrny

Abstract Factory – Factory Method implementace

- **Snadná změna skupiny tříd**
  - stačí použít jinou konkrétní továrnu
  - jen změna inicializace abstraktní továrny
- **Továrna jako singleton**
  - nemá smysl více než jedna instance každé konkrétní továrny
  - typicky chceme pouze jednu továrnu v celé aplikaci (kvůli konzistenci)
- **Přidávání skupin**
  - pro každou novou skupinu je nutné implementovat všechny metody
  - varianta: továrna není abstraktní – poskytuje defaultní implementace
- **Přidávání druhů objektů**
  - nutná změna rozhraní a všech konkrétních továren
  - řešení: rozšiřitelná továrna

Abstract Factory – rozšiřitelná implementace

- **Vlastnosti**
  - stačí jedna metoda - její parametr identifikuje objekt, který má být vytvořen
  - všechny objekty musí mít společného předka
  - eliminuje nutnost změny rozhraní továren při přidání nového druhu objektu
  - nevýhoda: ztrácíme část typové kontroly, klient musí provádět přetypování
  - jak volit typové identifikátory?

```
class GUIFactory {
public:
    virtual Widget* Create(WidgetId id);
};

Widget* GUIFactory::Create(WidgetId id) {
         switch (id) {
         case BUTTON:
              return new Button();
         case TEXTBOX:
              return new TextBox();
         default:
              return 0;
         }
}
```

```
class NewFactory: public GUIFactory {
public:
     virtual Widget* Create(WidgetId id);
};

Widget* NewFactory::Create(WidgetId id) {
         switch (id) {
         case BUTTON:
              return new NewButton();
         case SLIDER:
              return new Slider();
         default:
              return GUIFactory::Create(id);
         }
}
```

Abstract Factory – dynamická implementace

- **Vlastnosti**
  - továrna spravuje mapování: identifikátor typu → vytvářející funkce
  - konkrétní typy vytvářených objektů se registrují jednotlivě za běhu
  - stačí jedna univerzální implementace továrny
  - má širší použití – systémy kde neznáme přesné typy při kompilaci

```
class Factory {
          typedef AbstractProduct* (*)() ProductCreator;
          AssocMap<IdType,ProductCreator> mapping;
public:
          bool Register(IdType id,ProductCreator creator) {
               AssocMap::value_type toInsert(id, creator);
               return mapping.insert(toInsert).second;
          }
          bool Unregister(IdType id) {
               return mapping.erase(id) == 1;
          }
          AbstractProduct* CreateObject(IdType id) {
               AssocMap::const_iterator it = mapping.find(id);
               if (it != mapping.end())
                    return (*it->second)();
               return 0;
          }
};
```

Abstract Factory – implementace pomocí prototypů

- **Pouze jedna továrna obsahující prototypy**
  - pro změnu typů vytvářených objektů stačí změnit prototypy v továrně
  - objekty vytvářeny klonováním prototypů speciální virtuální metodou
  - pomocí prototypů lze implementovat všechny tři uvedené typy továren

```
class CloneFactory {
        Button *buttonPrototype;
        Window *windowPrototype;
public:
        CloneFactory(Button *buttonProt,Window *windowProt)
            : buttonPrototype(buttonProt), windowPrototype(windowProt) {}
        ~Factory() {
            delete buttonPrototype;
            delete windowPrototype;
        }
        Button* CreateButton() {
            return buttonPrototype->Clone();
        }
        Window* CreateWindow() {
            return windowPrototype->Clone();
        }
};
```

Abstract Factory – bludiště

- **Program stavící bludiště dostane `MazeFactory` jako parametr**
- **`MazeFactory` je abstraktní a zároveň konkrétní továrnou**

```
class MazeFactory {
public:
        // factory methods:
        virtual Maze* MakeMaze() const {
            return new Maze;
        }
        virtual Room* MakeRoom(int n) const {
            return new Room(n);
        }
        virtual Wall* MakeWall() const {
            return new Wall;
        }
        virtual Door* MakeDoor(Room* r1, Room* r2)
const {
            return new Door(r1, r2);
        }
};
```

Abstract Factory - bludiště

- **Potomci implementují specializace jednotlivých komponent**
- **Tována zajišťuje konzistenci komponent**
  - `RoomWithABomb` má kolem sebe `BombedWalls`

```
class BombedMazeFactory: public MazeFactory {
public:
        virtual Wall* MakeWall() const {
            return new BombedWall;
        }
        virtual Room* MakeRoom(int n) const {
            return new RoomWithABomb(n);
        }
};
```

```
class EnchantedMazeFactory: public MazeFactory {
public:
        virtual Room* MakeRoom(int n) const {
            return new EnchantedRoom(n, CastSpell());
        }
        virtual Door* MakeDoor(Room* r1, Room* r2) const {
            return new DoorNeedingSpell(r1, r2);
        }
protected:
    Spell* CastSpell() const;
};
```

Abstract Factory – bludiště

objekty vytváří továrna

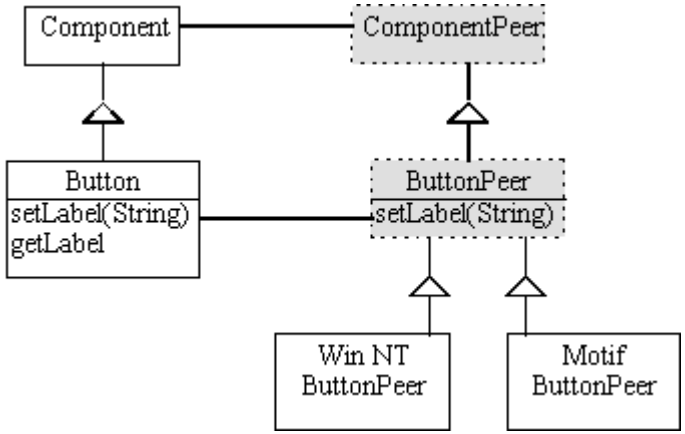
```
Maze* CreateMaze(MazeFactory *f) {
    Maze* aMaze = f->MakeMaze();
    Room* r1 = f->MakeRoom(1);
        Room* r2 = f->MakeRoom(2);
    Door* d1 = f->MakeDoor(r1, r2);
    aMaze->AddRoom(r1);
    aMaze->AddRoom(r2);
    r1->SetSide(North, f->MakeWall());
    r1->SetSide(East, d1);
    r1->SetSide(South, f->MakeWall());
    r1->SetSide(West, f->MakeWall());
    .....
    return aMaze;
}
```

názvy objektů „natvrdo“

```
Maze* MazeGame::CreateMaze() {
    Maze* aMaze = new Maze;
    Room* r1 = new Room(1);
    Room* r2 = new Room(2);
    Door* d1 = new Door(r1, r2);
    aMaze->AddRoom(r1);
    aMaze->AddRoom(r2);
    r1->SetSide(North, new Wall);
    r1->SetSide(East, d1);
    r1->SetSide(South, new Wall);
    r1->SetSide(West, new Wall);
    .....
    return aMaze;
}
```

Abstract Factory – použití v AWT

- **AWT**
  - Abstract Window Toolkit
  - každá AWT komponenta (Component) obsahuje odpovídající objekt daného systému (ComponentPeer)
- **java.awt.Toolkit**
  - AbstractFactory
  - potomci vrací příslušné objekty daného systému (MS Windows, X Window)
  - createMenu(), createButton(), …
  - getDefaultToolkit(), getToolkit()




```
public static Toolkit getDefaultToolkit() {
    if (toolkit == null) {
        String nm = System.getProperty(
            "awt.toolkit",
            "sun.awt.motif.MToolkit");
        toolkit = (Toolkit) Class.forName(nm).
            newInstance();
    }
    return toolkit;
}
```

Abstract Factory – shrnutí

- **Definice abstraktní továrny**
  - rozhraní pro vytváření rodin souvisejících nebo závislých objektů
- **Jakou implementaci?**
  - je třeba volit mezi flexibilitou a “type safety“
  - někdy omezené možnosti – například nemůžeme zasahovat do kódu vytvářených objektů
  - záleží na konkrétním použití
- **Související vzory**
  - Factory Method - abstraktní továrna často používá tovární metody
  - Prototype - abstraktní továrna může být implemetovaná i pomocí prototypů
  - Singleton - je vhodné, aby továrna byla singleton