Objektově orientované programování

Návrh programu II

2023/24

Osnova přednášky

- Dědičnost – shrnutí
- Návrh programu s využitím dědičnosti

Dědičnost – shrnutí

Klíčové otázky

- **CO** je dědičnost?
- **PROČ** použít dědičnost?
- **KDY** použít dědičnost?
- **JAK** správně použít dědičnost?

CO je dědičnost?

- Koncept, který poskytuje substituční princip.
- Popis skupiny entit se společným chováním na základě vztahu „generalizace - specializace“.
- Efektivní rozšíření a změna existujícího (otestovaného) kódu.

PROČ použít dědičnost?

- Bez dědičnosti by se některé úlohy řešily obtížně…
- …zejména v případě, kdy potřebujeme zastoupit předka potomkem, který má specifické chování.
- Objekt pak hraje v různých kontextech různě specializované role.

KDY použít dědičnost?

- Potřebujeme polymorfní přiřazení.
- Potřebujeme objekty organizovat v polymorfních datových strukturách.
- Potřebujeme pracovat s „podobnými“ entitami (které mají společné chování).
- Prostě potřebujeme *polymorfismus*.

JAK použít dědičnost?

- Opatrně!!!
- Pokud možno, jen jednoduchou dědičnost.
  - Vícenásobnou dědičnost jen jako interface (čistě abstraktní třídy).
- Preferovat rozšíření před změnou.
  - Když změna, měl by se zvážit polymorfismus (pozdní vazba).
- Změnu používat s ohledem na zachování zapouzdření.
  - Tedy minimalizovat využití detailů implementace předka (protected).

Návrh programu
s využitím dědičnosti

Návrh programu s dědičností

- V konkrétním případě zvážit, kdy použít skládání a kdy dědičnost.
  - Někdy lze použít obě alternativy.
- Navrhnout dědičnou hierarchii tříd.
- Navrhnout, kde je potřeba pracovat s polymorfními datovými strukturami (a jak).
- Zvážit použití vícenásobné dědičnosti.

Zadání

- Stavíme objektový základ elektronického obchodu specializovaného na počítače a mobilní zařízení (telefony, tablety, notebooky,…). Nakupovat mohou registrovaní uživatelé a firmy, ale také jednorázově osoby bez registrace. Produkty se vybírají z katalogu. Detaily produktu z katalogu je možno získat v textové podobě. Objednávka obsahuje produkty a počet kusů. Objednávku lze souhrnně získat v textové podobě ve formě informací o zákazníkovi a seznamu položek s cenou a počtem kusů.

Identifikace tříd

- Stavíme objektový základ elektronického obchodu specializovaného na počítače a mobilní zařízení (telefony, tablety, notebooky,…). Nakupovat mohou registrovaní uživatelé a firmy, ale také jednorázově osoby bez registrace. Produkty se vybírají z katalogu. Detaily produktu z katalogu je možno získat v textové podobě. Objednávka obsahuje produkty a počet kusů. Objednávku lze souhrnně získat v textové podobě ve formě informací o zákazníkovi a seznamu položek s cenou a počtem kusů.

Koncept objektového návrhu

- Hlavními entitami jsou *Zákazník*, *Produkt*, *Katalog produktů*, *Objednávka*, *Položka objednávky*.

- Některé entity mají své speciální případy.

- Některé entity vyžadují použití struktur (pro společné uchování entit „podobného“ typu).

Speciální případy

- Produkt
  - Notebook
  - Mobilní telefon
  - Tablet

- Zákazník
  - Registrovaný x neregistrovaný
  - Uživatel x firma

Struktury

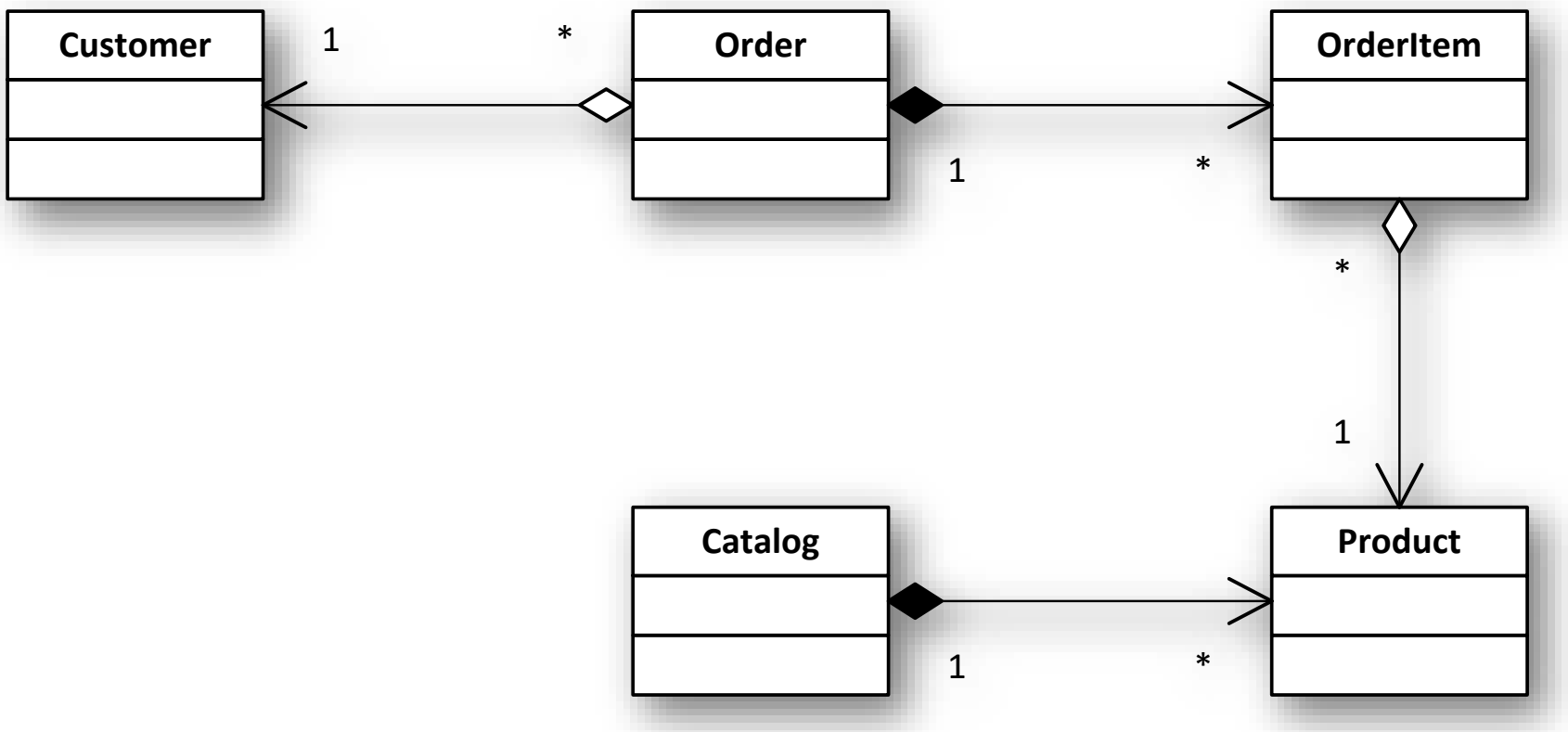
- Katalog produktů
  - Produkty v katalogu jsou různého typu

- Objednávka s jednotlivými položkami
  - Položky objednávky jsou stejného typu

Jak skládat objekty?

- Katalog **MÁ** produkty.
- Objednávka **MÁ** zákazníka a položky objednávky.
  - Nebo zákazník má objednávku?
- Položka objednávky **MÁ** produkt.

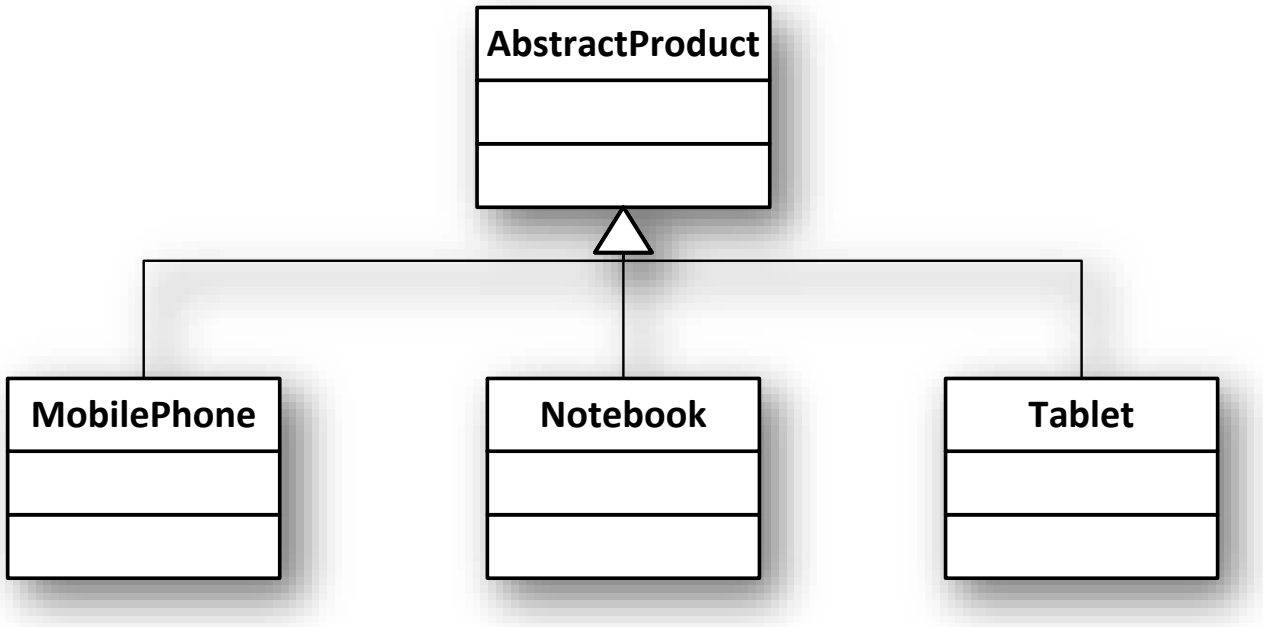
Objednávka (UML statický diagram tříd)



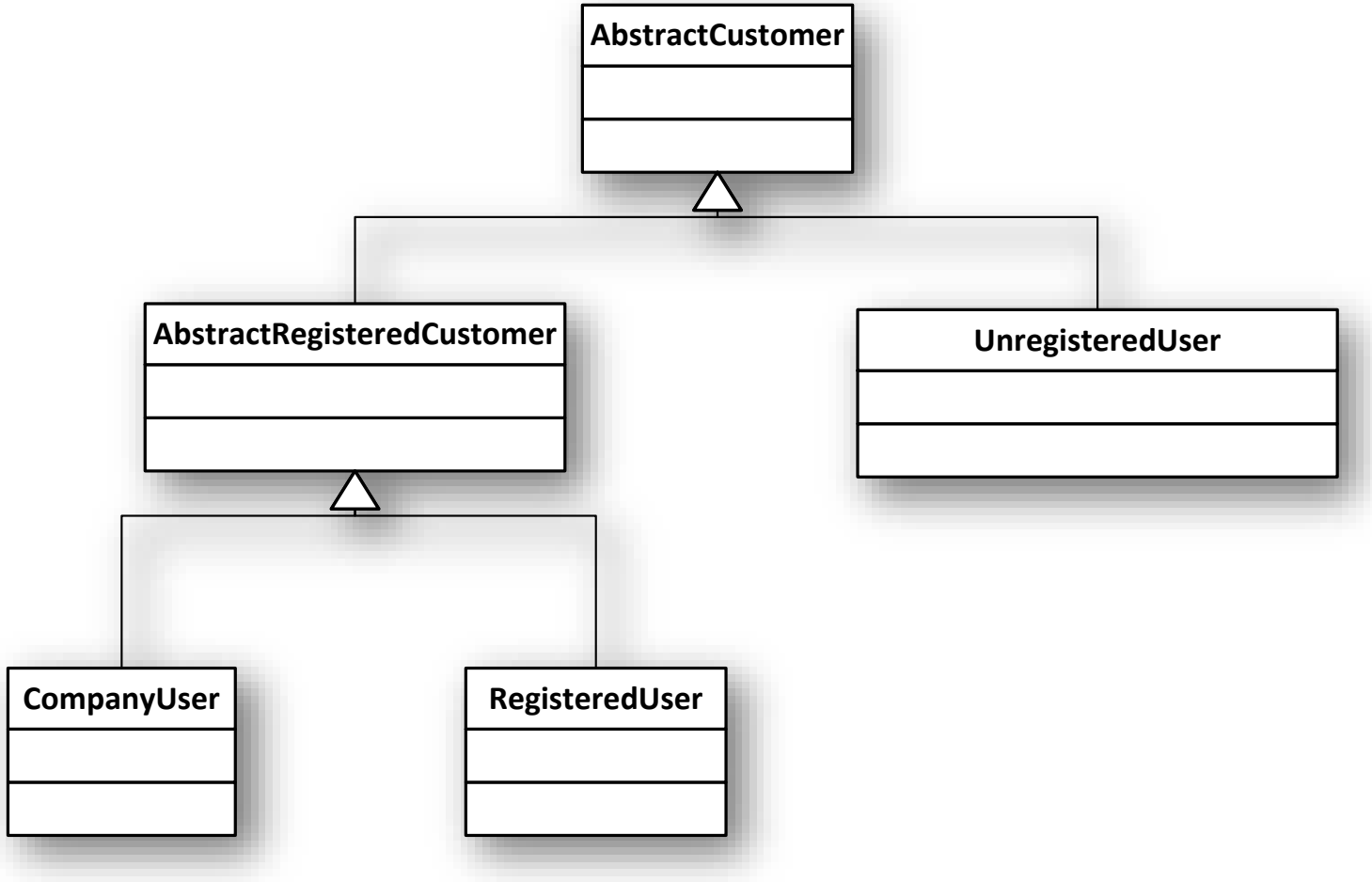
Jak na speciální případy?

- Mobilní telefon, počítač, tablet **JE** produkt.
- Registrovaný i neregistrovaný uživatel **JE** zákazník.
- Uživatel i firma **JSOU** zákazníci.

Produkty



Zákazníci


AbstractCustomer
AbstractRegisteredCustomer
UnregisteredUser
CompanyUser
RegisteredUser

Polymorfní přiřazení a struktury

- Katalog?
  - Tři různé nepolymorfní katalogy pro různé produkty…
  - Proč?
  - Můžeme přetypovat?
- **Objednávka může mít různé zákazníky.**
- **Položka objednávky může mít různý produkt.**

Výstup do textu

- Stavíme objektový základ elektronického obchodu specializovaného na počítače a mobilní zařízení (telefony, tablety, notebooky,…). Nakupovat mohou registrovaní uživatelé a firmy, ale také jednorázově osoby bez registrace. Produkty se vybírají z katalogu. Detaily produktu z katalogu je možno získat v textové podobě. Objednávka obsahuje produkty a počet kusů. Objednávku lze souhrnně získat v textové podobě ve formě informací o zákazníkovi a seznamu položek s cenou a počtem kusů.

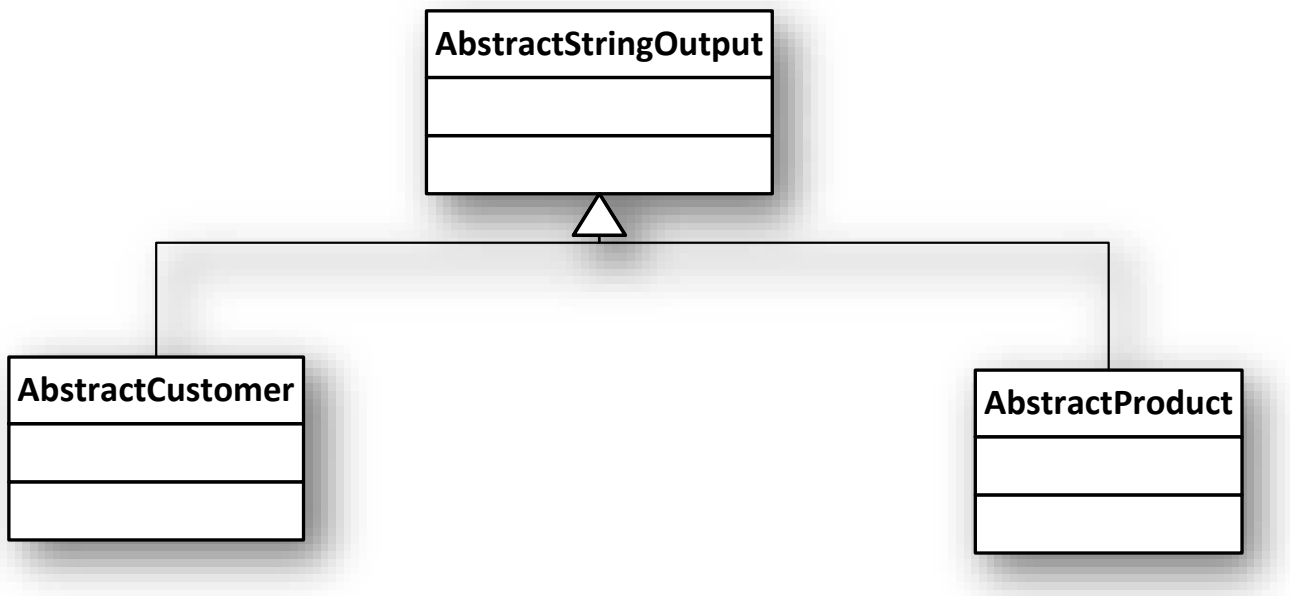
Jak s textovým výstupem?

- Produkt se musí dát vypsat na objednávku.
- Zákazník se musí dát vypsat na objednávku.
- Jak to udělat?
  - Objednávka „neví“ jakého typu je zákazník.
  - Položka objednávky „neví“, jakého typu je produkt.

Jaké máme možnosti?

- Každá třída může mít metodu „*ToString*“.
  - Má umět objekt poskytovat data ve formě souhrnného textu?
- Můžeme připravit abstraktní třídu „*AbstractStringOutput*“ s čistě virtuální metodou „*ToString*“.
  - Každá třída, která potřebuje poskytovat data ve formě textu, z ní může dědit.
  - Můžeme použít jak jednoduchou, tak vícenásobnou dědičnost.

Společný předek

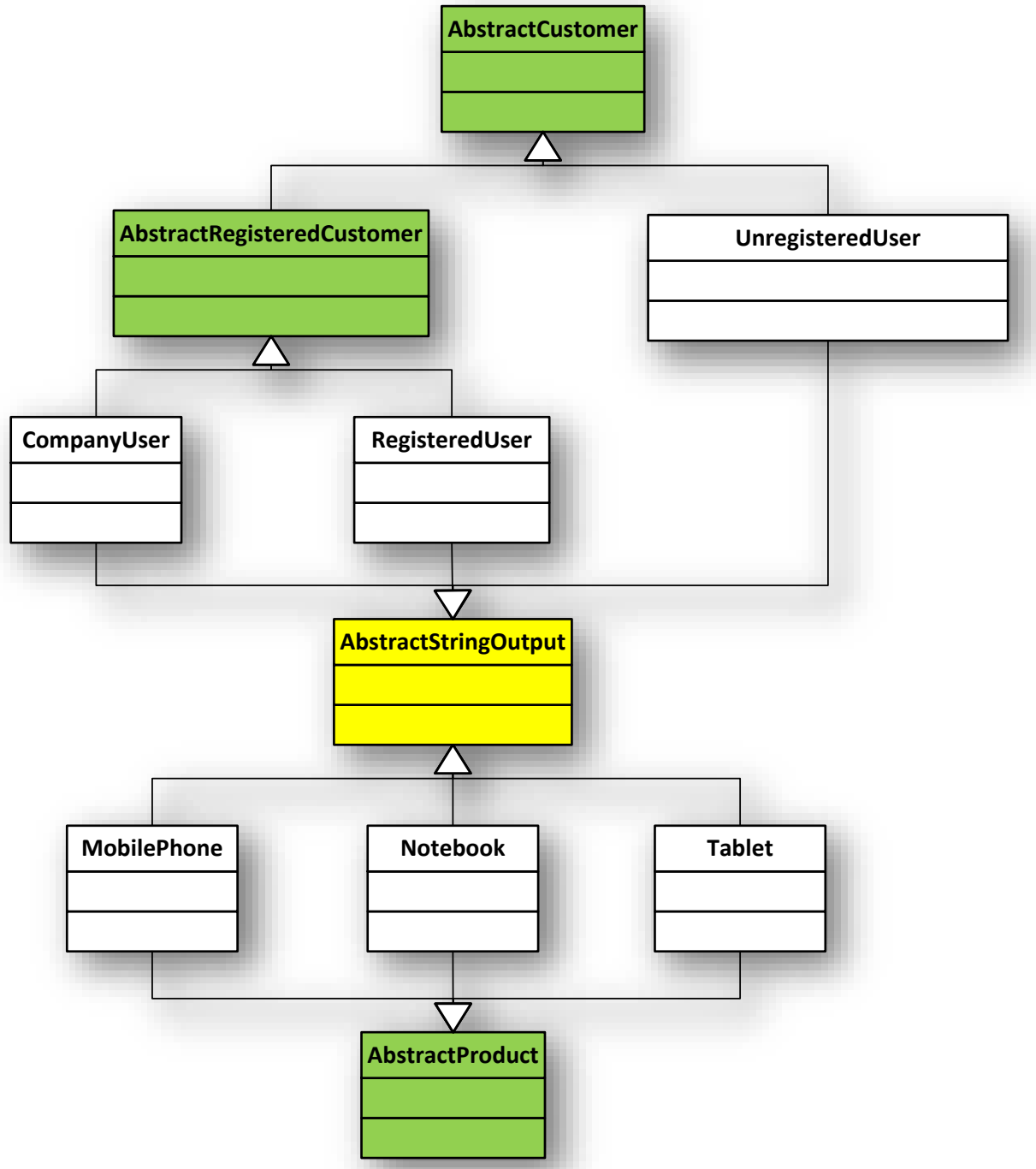


Použití

```
AbstractCustomer * c;
AbstractStringOutput * o;

CompanyUser * u = new CompanyUser(…);

o = u;
o->???;

c = u;
c->???;
```

Je to správně?

- Je *AbstractCustomer* a *AbstractProduct* speciálním případem *AbstractStringOutput*?
  - Spíše ne.
- A kdo je tedy speciálním případem *AbstractStringOutput*?
  - Konkrétní třídy


AbstractCustomer
AbstractRegisteredCustomer
UnregisteredUser
CompanyUser
RegisteredUser
AbstractStringOutput
MobilePhone
Notebook
Tablet
AbstractProduct

V čem je rozdíl?

```
AbstractCustomer * c;
AbstractStringOutput * o;

CompanyUser * u = new CompanyUser(…);

o = u;
o->???;

c = u;
c->???;
```

Abstraktní třída x Interface

- „Interface“ je koncept, který v C++ chybí.
  - Dědičnost – dědí a rozšiřuje se chování
  - Interface – chování se musí implementovat
- Můžeme použít čistě abstraktní třídu (nemá implementaci).
- Použitím interface třída nic nedědí, pouze dostává předpis, co musí implementovat.

V čem je rozdíl?

- Abstraktní třída v C++ má vždy implementaci (alespoň výchozí konstruktor a destruktor).
- Interface nemá žádnou implementaci.
- V C++ se tedy vždy jedná o dědičnost, ale využitím čistě abstraktních tříd dosáhneme stejného efektu jako s použitím interface.
- Doporučení – interface by měla implementovat konkrétní třída (a každá po svém).

Úkoly na cvičení

- Implementujte dědičné hierarchie zákazníků a produktů podle přednášky. Jak pro zákazníky, tak pro produkty navrhněte data a chování, které potomci rozšiřují.
- Navrhněte a implementujte třídy pro objednávku a položku objednávky.
- Napište program, ve kterém vytvoříte několik různých objednávek (s různými typy zákazníků a produktů).
- Vypište obsah vytvořených objednávek na obrazovku.