Programming II

Design of Program II

Lecture Outline

- Inheritance – summary
- Design of program with inheritance

Inheritance – summary

Key Questions

- **WHAT** is inheritance?
- **WHY** to use inheritance?
- **WHEN** to use inheritance?
- **HOW** to use inheritance correctly?

WHAT is inheritance?

- A concept which provides substitution principle.
- Description of entities with common behavior by the relationship "generalization - specialization."
- Effective extension and modification of existing (tested) code.

WHY to use inheritance?

- A solution of some tasks would be difficult without inheritance...
- ...especially when we need to replace an ancestor by a descendant with a specific behavior.
- The object plays different specific roles in different contexts.

WHEN to use inheritance?

- We need polymorphic attachment (assignment).
- We need to organize objects in a polymorphic data structures.
- We need to work with "similar" entities (which have a common behavior).
- We just need *polymorphism*.

HOW to use inheritance?

- Very carefully!!!
- If possible, a simple inheritance.
  - Multiple inheritance just like the interface (pure abstract class).
- Prefer extension before the change.
  - The extension and changes should use polymorphism (late binding).
- Changes should be used regarding to preserve encapsulation.
  - Minimizing the use of implementation details of ancestors (protected).

Design of program with inheritance

Design with inheritance

- In the specific case, consider when to use compositions and when to use inheritance.
  - Sometimes we can use both alternatives.
- Design a class hierarchy based on inheritance.
- Define where to work with polymorphic data structures.
- Consider using multiple inheritance.

Assignment

- We are building e-shop which is specialized on computers and mobile devices (phones, tablets, laptops). Products can be bought by registered users, company users and users without registration. Products are selected from the catalog. Details of the product can be obtained in textual form. The order contains products and number of order items. The order can aggregate information about customer and order items (products with price and quantity) in textual form.

Identification of Classes

- We are building e-shop which is specialized on computers and mobile devices (phones, tablets, laptops). Products can be bought by registered users, company users and users without registration. Products are selected from the catalog. Details of the product can be obtained in textual form. The order contains products and number of order items. The order can aggregate information about customer and order items (products with price and quantity) in textual form.

Concept of Object-oriented Design

- The main entities are *Customer*, *Product*, *Product Catalog*, *Order*, *Order Item*.
- Some entities have their special cases.
- Some entities require the use of structures (to preserve entities of similar type).

Special Cases

- Product
  - Notebook
  - Mobile phone
  - Tablet

- Customer
  - Registered x Unregistered
  - User x Company

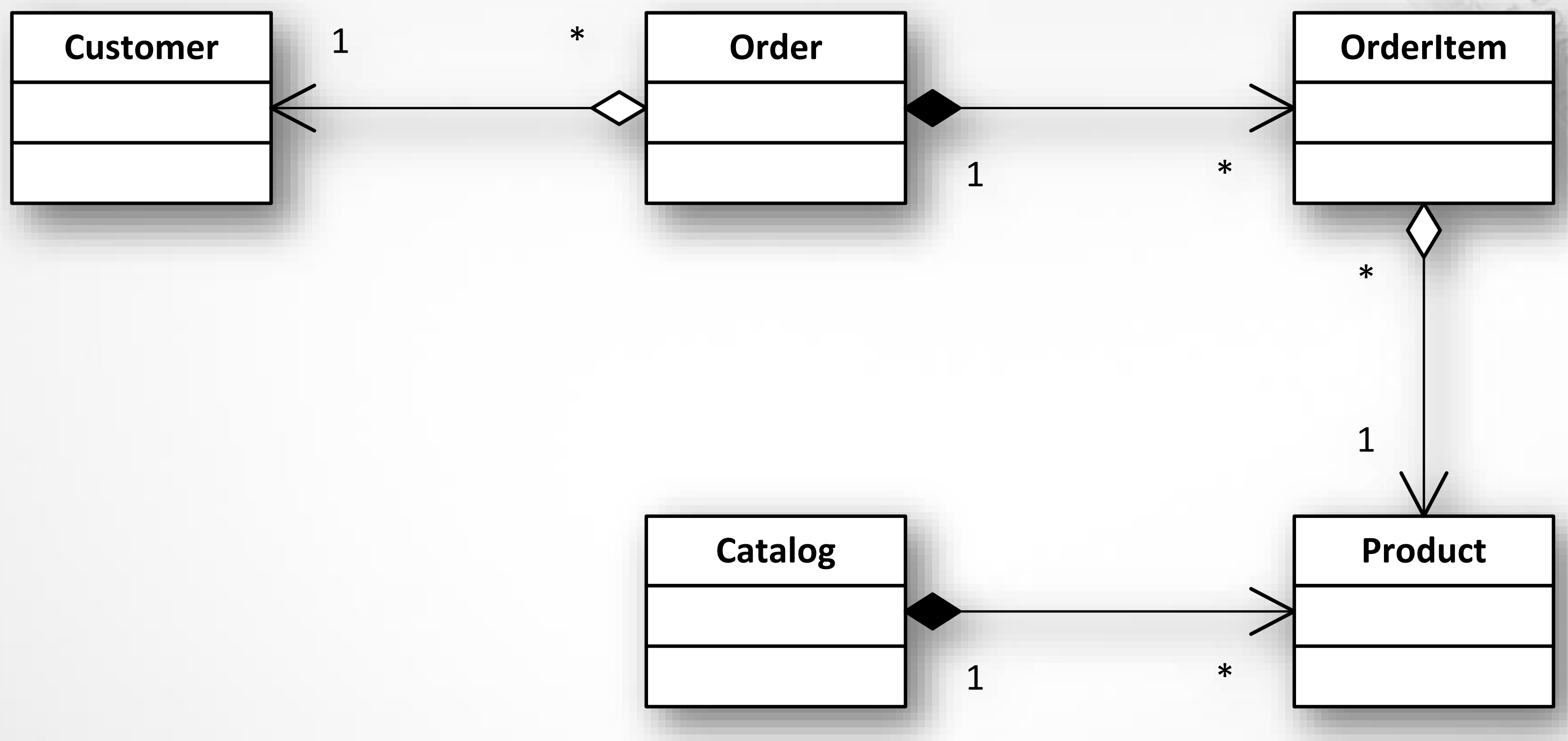
Structures

- Product catalog
  - Products of different types
- Order with items
  - Order items of the same type

Object Compositions

- The catalog **HAS** products.
- The order **HAS** a customer and order items.
  - Alternatively, the customer HAS orders?
- The order item **HAS** a product.

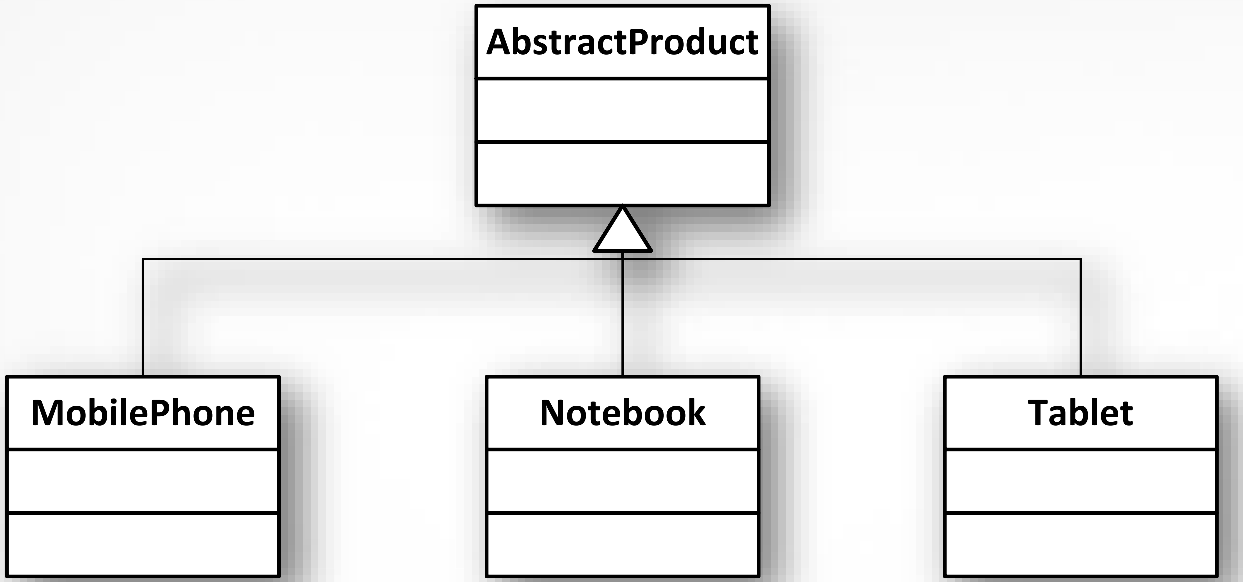
Order



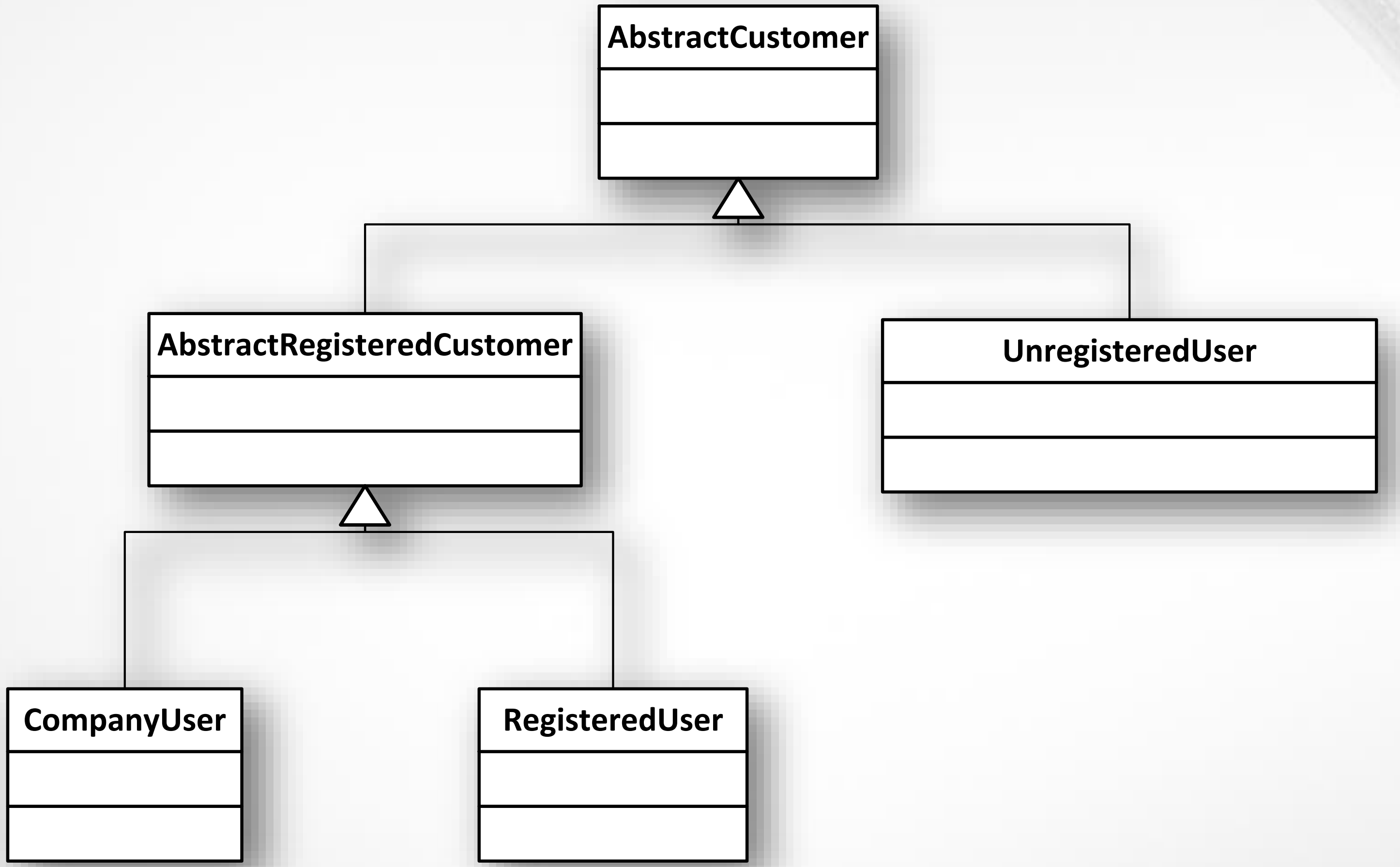
Special cases?

- Mobile phone, notebook, tablet **IS** a product.
- Registered and unregistered user **IS** a customer.
- User and company **ARE** customers.

Products



Customers


AbstractCustomer
AbstractRegisteredCustomer
UnregisteredUser
CompanyUser
RegisteredUser

Polymorphic attachment and structures

- Catalog?
  - Three different non-polymorphic catalogs with different products?
  - Why?
  - **Polymorphic catalog**
- **The order may have different customers.**
- **Order items may have a different product.**

Text output

- We are building e-shop which is specialized on computers and mobile devices (phones, tablets, laptops). Products can be bought by registered users, company users and users without registration. Products are selected from the catalog. Details of the product can be obtained in textual form. The order contains products and number of order items. The order can aggregate information about customer and order items (products with price and quantity) in textual form.

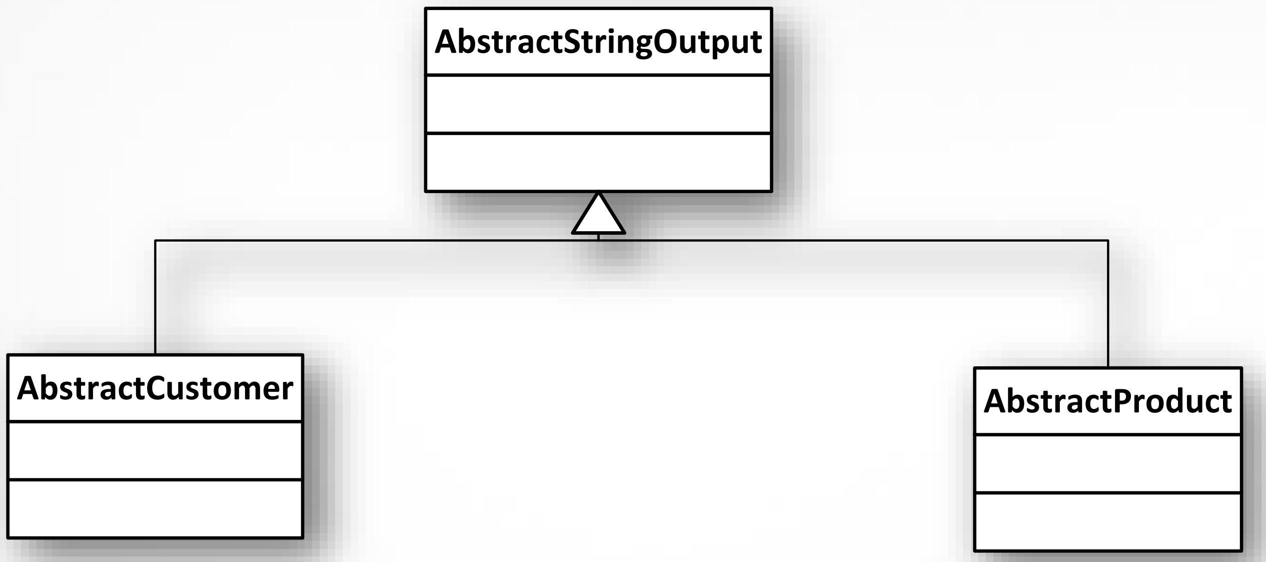
Text Output?

- The product details must be included on the order.
- Customer info must be included on the to order.
- How to do it?
  - The order does not "know" the type of the customer.
  - Item orders "do not know" type of the product.

Possibilities?

- Each class may have a method "*ToString*".
- We can declare an abstract class "*AbstractStringOutput*" with a pure virtual method "*ToString*".
  - Any class that needs to provide data in the text form may inherit.
  - We can use both single and multiple inheritance.

Common Ancestor

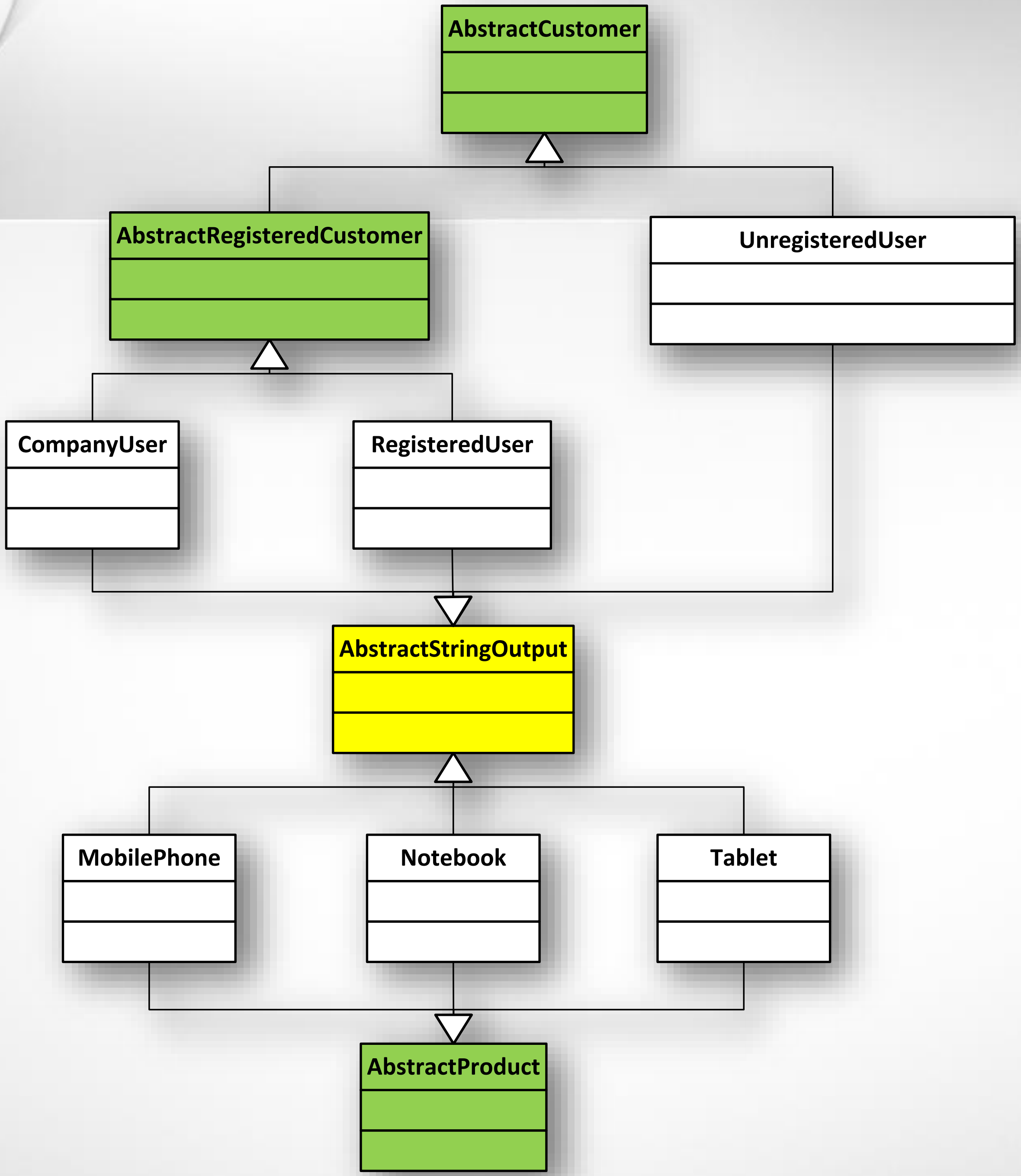


Using

```
AbstractCustomer * c;
AbstractStringOutput * o;

CompanyUser * u = new CompanyUser(…);

o = u;
o->???;

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

Is it correct?

- Be *AbstractCustomer* and *AbstractProduct* a special case of *AbstractStringOutput*?
  - Probably no.
- Who is the special case of *AbstractStringOutput*?
  - Concrete classes


AbstractCustomer
AbstractRegisteredCustomer
UnregisteredUser
CompanyUser
RegisteredUser
AbstractStringOutput
MobilePhone
Notebook
Tablet
AbstractProduct

What is the difference?

```
AbstractCustomer * c;
AbstractStringOutput * o;

CompanyUser * u = new CompanyUser(…);

o = u;
o->???;

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

Abstract Class x Interface

- "Interface" is a concept that is missing in C ++.
  - Inheritance - inherits behavior
  - Interface - implements behavior
- We can use pure abstract class (with no implementation).
- By using the interface, a class does not inherit anything. It is a declaration what must be implemented.

What is the difference?

- The abstract class in C ++ always has an implementation (at least constructor and destructor).
- Interface has no implementation.
- In C ++, we have only the inheritance but using pure abstract classes we achieve the same effect as using the interface.
- Recommendation - interface should be implemented by a specific (concrete) class.