**Architecture and Design**

**laboratory work 7**

**EXERCISE 01 – Creating UML diagrams**

**IMPORTANT: PERFORM ALL TASKS ACCORDING TO YOUR OWN TOPIC.**

# **Overview**

In this project, you will continue with the design of your project. You will include a variety of UML diagrams that will graphically depict your design.

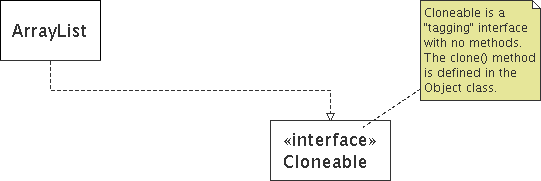
To successfully complete this exercise, you must:

1. Create a Class Diagram (at least 6 classes, which cover all types of relationships) that models your object-oriented system.
   * 1. Define your classes and for each class draw three-sector squares.

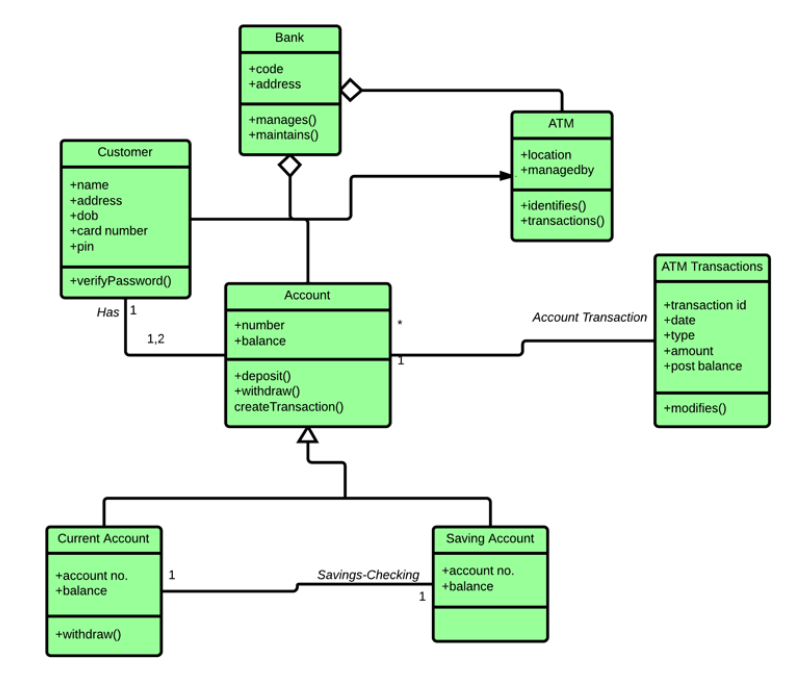
* The first sector is for a name of a class.
* The second is for attributes, which are data containing values that describe each instance of a class. Also here you have to write data type. (It must look like this: **-name: String**)
* And the third sector is for methods, which allow you to specify any behavioral features of a class. (It must look like: **-setName()**).
* Also you have to pay attention on visibility, if your variables or methods are private, you should put –(minus) before; if they are public, there goes +(plus); if one of them is protected there goes #(hashtag); also there is ~, this sets for package or default, and underline static attributes.
  + 1. Next, you have to define relationships between classes.
* The first type of relationship is Inheritance, which covers by this arrow , for using it you should fine subclasses of superclass. Also can be used for interface implementation.
* Another type of relationship is Association, which sets by this arrow . If two classes in a model need to communicate with each other, there must be a link between them, and that can be represented by an association (connector). We can indicate the multiplicity of an association by adding multiplicity adornments to the line denoting the association.  (For example,
* owners feed pets, pets please owners (association)
* a tail is a part of both dogs and cats (aggregation / composition)
* a cat is a kind of pet (inheritance)).
* The next type of relationship is Aggregation, where you have to use this type of arrow  . It’s a special type of Association denoting a "consists-of" hierarchy. (For example, car-wheel).
* The next type of relationship is Composition, where you have to use this type of arrow

. A special type of aggregation where parts are destroyed when the whole is destroyed. (For example, house-room).

* + 1. Next, you have to define multiplicity between your classes. How many objects of each class take part in the relationships and multiplicity can be expressed as:
* 0…1
* n (specific number)
* 0…\* (zero to many)
* 1…\*(one to many)
* m…n (specific number range)
  + 1. Comments can be represented as a folded note, attached to the appropriate class/method/etc by a dashed line



* + 1. Save your diagram and upload it on DL.

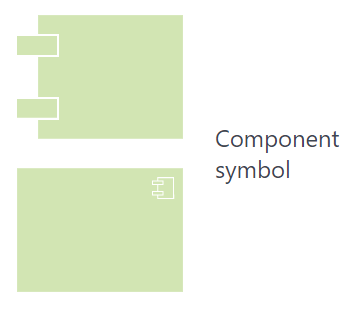


Pic. 1 – An example of Class Diagram

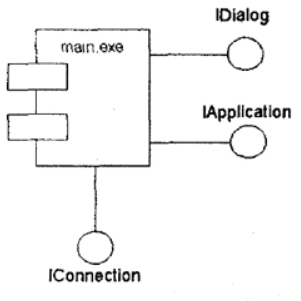
1. Create a Component Diagram that shows the relationship between different components (files, libraries, modules, executables, packages, etc.) in a system.
   * 1. Firstly, a component is shown as a rectangle with
2. A keyword <<component>>
3. Optionally, in the right hand corner a component icon can be displayed

* A component icon is a rectangle with two smaller rectangles jutting out from the left-hand side
* This symbol is a visual stereotype

c. The component name

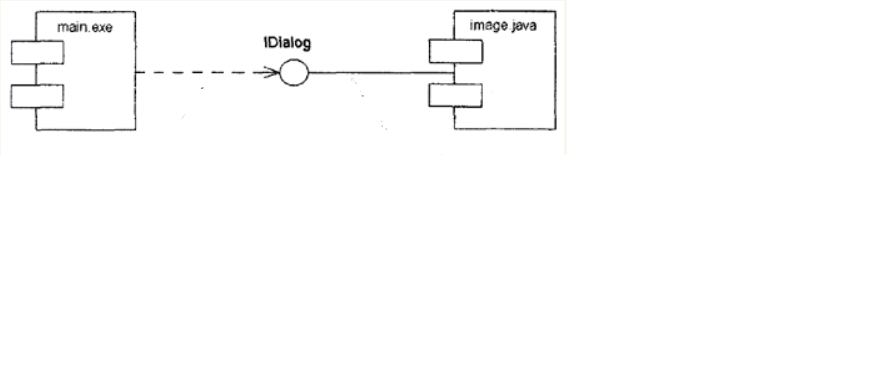


* + 1. An interface represents a declaration of a set of operations and obligations. Graphically, the interface is depicted by a circle that connects to the component by cutting the line without arrows. The name of the interface is written next to the circle.

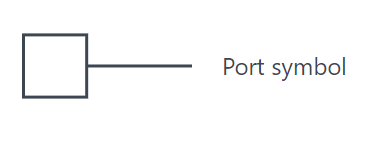


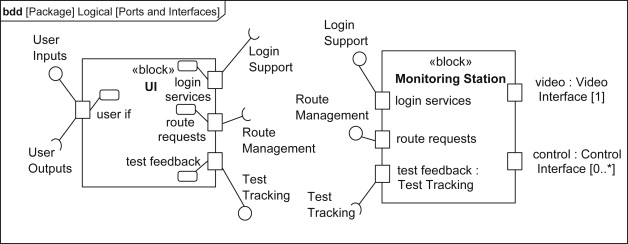
|  |  |
| --- | --- |
| Component Diagram Provided Interfaces | **Provided interfaces:** A straight line from the component box with an attached circle. These symbols represent the interfaces where a component produces information used by the required interface of another component. |
| Component Diagram Required Interfaces | **Required interfaces:**A straight line from the component box with an attached half circle (also represented as a dashed arrow with an open arrow). These symbols represent the interfaces where a component requires information in order to perform its proper function. |

* + 1. A usage dependency is relationship which one element requires another element for its full implementation. The usage dependency on the component diagram is depicted by a dashed line with an arrow directed from the client (dependent element) to the source (independent element).



the component “main.exe” depends on the imported IDialog interface, which, in turn, is implemented by the component named “image.java”.

* + 1. Ports. Port represents an interaction point between a component and its environment
    2. Save your diagram and upload it on DL.



Pic. 2 – An example of Component Diagram

A required interface on a port specifies one or more operations required by behaviors of the block (or its parts). A [provided interface](https://www.sciencedirect.com/topics/computer-science/provided-interface) on a port specifies one or more operations that a block (or one or more of its parts) must provide. A part that has a port with a required interface needs to be connected to another part that provides the services it needs, typically via a port with a provided interface.

Pic 2 shows the set of ports that define interface points on the blocks UI and Monitoring Station. UI has five ports of defined interfaces: two that provide services, three that require services. The port test feedback provides the services defined by the provided interface Test Tracking, whereas in Monitoring Station there is a required interface, which requires test feedback. The port login services requires the services defined by the required interface Login Support, while in Monitoring Station the same port provides the services defined by the provided interface Login Support. The port user if offers services defined in User Inputs and requires services defined by User Outputs. All of UI’s ports are behavior ports as indicated by the behavior port symbol.