ASTANA IT UNIVERSITY	**Cours:** Object-Oriented Programming **Deadline:** February 07, 2024 till (23:59)

Instructor: Toleu Altynbek

ASSIGNMENT 5

CREATION OF CLASSES, DESCRIPTION OF PROPERTIES, WORKING WITH OBJECTS

A class is a description of objects that are similar in characteristics (properties) and behavior (methods). In programming, a class is a template (type) for creating objects.

This class is defined (i.e. differs from other classes) by a set of properties that determine the state of objects created on the basis of this class, as well as by an interface - a set of methods (procedures and functions) that allow the user to perform certain actions on objects, i.e. e. determining the behavior of objects.

Creating an object-oriented application consists of two stages: creating classes and using classes.

1. Creating a class in JAVA is done using the *class* keyword, for example:

```
// class declaration Fruit

class Fruit {

}
```

To describe a property, a private variable of a certain type is created (encapsulated) inside the class.

```
// class declaration Fruit

class Fruit {

// private variable-property “color”
private String color;

}
```

The principle of encapsulation is that data is hidden from the user within a class. Access to a property is provided (at the discretion of the class developer) using the methods for reading the property value (get) and writing the property value (set)

```
// class declaration Fruit

class Fruit {

// private variable-property "color"

private String color;

// public function method that provides reading of the "color" property

public String getColor() {

         return color;
      }

// public method-procedure that records the "color" property

public void setColor(String newColor) {

         color = newColor;
      }

}
```

There are special languages (notations) for graphically displaying object-oriented structures that allow you to visually model various aspects of the system being designed. One of the most common is the Unified Modeling Language (UML – Unified Modeling Language). In UML, a class is represented as a rectangle divided into three parts. The name of the class is displayed in the upper part, a set of fields (properties) in the middle, and a set of methods in the lower part. For properties and values types can be specified; for methods, sets of parameters and types of return values can be specified. The implementation (content) of the methods is not displayed.

The Fruit class in UML can be represented as follows:

Fruit
color: String

```
void setColor(String newColor)
String getColor()
```

Fig 1. Description of the Fruit class in UML

2. Using classes.

The use of classes consists of creating objects and calling (calling) methods of the created objects.

```
public class TestFruit {
 public static void main(String args[]) {

             // creating Fruit class objects
             Fruit apple = new Fruit();
             Fruit lemon = new Fruit();

            // writing property values
            apple.setColor("green");
            lemon.setColor("yellow");

            // reading property values

            System.out.println(apple.getColor());

            System.out.println(lemon.getColor());

  }
}
```

```
// class declaration Fruit

class Fruit {

   // private variable-property "color"

   private String color;

// public method that provides reading of the "color" property

public String getColor() {

    return color;
 }

// public method that provides a record of the "color" property

public void setColor(String newColor) {

    color = newColor;
 }
}
```

Relationships between classes (composition)

Objects of different classes can be interconnected with each other, for example, an Employee works in an Organization, an Airport offers Flights, a Fruit grows on a Tree, etc.

In object-oriented design, such relationships (associations) between classes are described using a composition relationship. The participants of the relationship are two interrelated classes and an association. The association can be unidirectional or bidirectional. For each end of the association, the “arity” is determined, i.e. a limit on the number of objects of a given class that can interact with a specified number of objects of another class.

1. Unidirectional association means a one-way relationship between two classes, when one class contains one or many references to objects of the second class. As a result, we get a construction that means that an object of the first class “knows” about connections with objects of another class.

In UML, a unidirectional association is represented by a solid arrow from the source class to the associated class with which it interacts. Multiplicity is indicated near the ends of the arrow as <min. value> [... <max. value>]. If as a max. values are indicated by the symbol * - this means that there can be any number of objects.

Example:

Let us display the structure of the relationship: Fruit grows on a Tree.

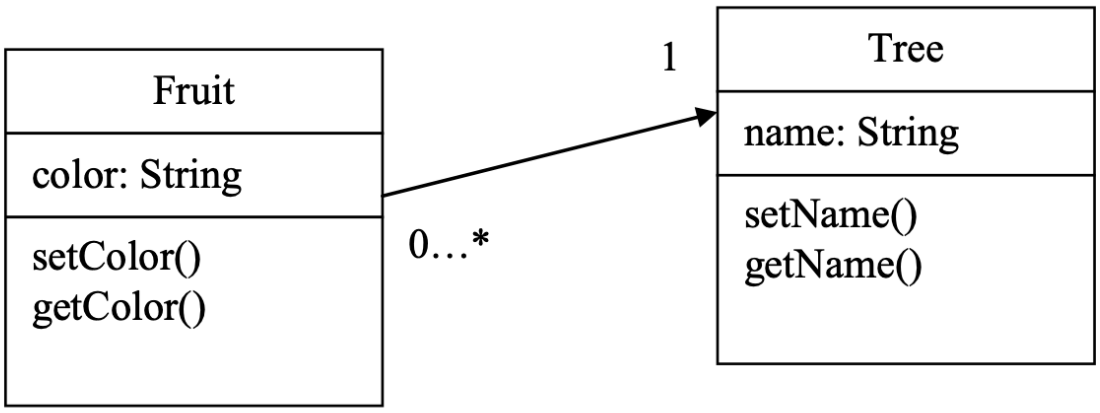

Fig 2. Unidirectional communication between the Fruit and Tree classes in UML

This structure says that each fruit “knows” which tree it grows on, while each fruit object can grow on only one tree, and a tree can contain from 0 to an infinite number (*) of fruits.

2. Bidirectional association, in contrast to unidirectional, means a two-way relationship between classes, when the first class contains references to objects of the second class, and the second class contains references to objects of the second. As a result, each of the classes participating in the association “knows” about connections with another class.

A bidirectional association in UML is shown as a solid line. Let's display the relationship Fruit grows on a Tree in the form of a bidirectional association:

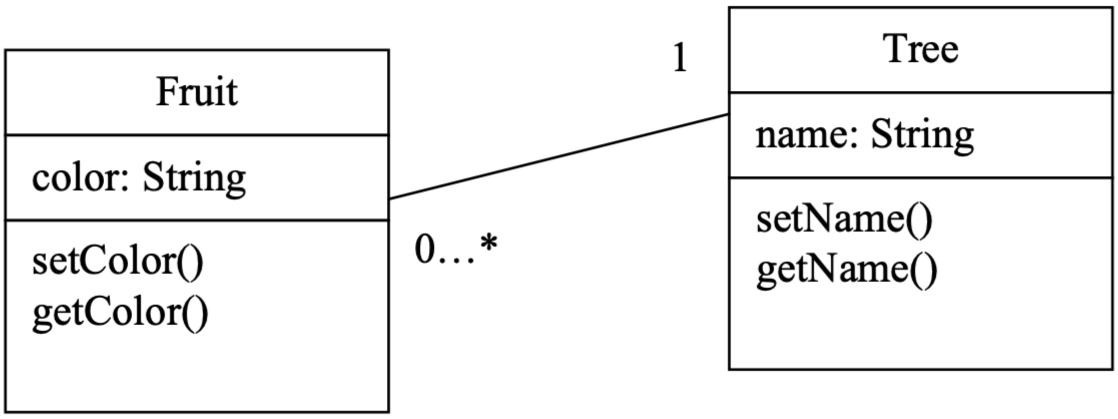

Fig 3. Bidirectional communication between the Fruit and Tree classes in UML

Unlike the previous example, here the tree also "knows" about the fruit it contains.

Implementation of associative links in JAVA

In object-oriented programming, associative connections are implemented using references to objects of the corresponding classes, i.e. in fact, associations are implemented in the usual way, in the form of class properties. Previously, we have already used associative relationships with objects of the String type, describing the properties of the string type. The only difference is that previously we used the standard String class, but now we use our own classes.

For example, the structure shown in Fig. 2 is implemented as follows.

```
// class declaration Fruit
     class Fruit {
     // Description of the color property
     ...
     // Description of the connection between Fruit and Tree
     // Declaring a reference to an object of the Tree class
      private Tree tree;

// Get the value of the Tree property
// (find out which tree this fruit grows on)

public Tree getTree() {

         return tree;
      }

// Set the value of the Tree property (indicate on which tree this fruit
grows)

public void setTree(Tree tree) {
         this.tree = tree;
}
```

```
// Tree class declaration
     class Tree {
     // Description of the name property
     ...

}
```

Bidirectional association is implemented by introducing the corresponding association property into the second class.

Implementation of multiple properties, collections.

A multiple property, such as the “fruit” property of the Tree class, which appears when creating the Tree -> Fruit association in Figure 3, must be a **set of references** of a certain type. In this case, the internal variable can be described as an array of references, for example:

```
// Tree class declaration
     class Tree {
      // Description of the name property
      private Fruit fruits[] = new Fruit[100];

}
```

Access to this property can be provided through the set and get operations, but this is not always convenient for users of the class, since it forces them to operate with a whole set of objects each time. At the same time, you may need operations like “add fruit”, “remove fruit”, etc., which are quite difficult to implement when using arrays. Also, the original dimension of the array is often unknown.

It is much more convenient in such cases to use special data types called “collections”. **A collection** is a special object that is a container for storing many references to other objects.

Main advantages of the collections:

- have a dynamic dimension (i.e. the dimension is not specified when declared and can change during program operation). Thus, the collection size is limited only by the amount of RAM.
- can store objects of any type (all objects stored in the collection are cast to the base type Object)
- there are standard methods for adding and removing collection elements
- collection elements are indexed (similar to arrays), reindexing occurs automatically when adding/removing.

It should be understood that the collection itself represents an object of a certain class, independent of the objects stored in it. A collection can be thought of as a “box” containing objects. This “box” allows you to put objects into it, take them out, move them, find out the number of objects, etc.

One of the collection types is the ArrayList object type (hosted in the java.util package). The collection declaration looks like this:

```
ArrayList <name> = new ArrayList();
```

Note:

To gain access to the ArrayList class in the program, you must load this class from the java.util package. To do this, at the beginning of the program code (before the description of the classes), you need to place the following line:

```
import java.util.ArrayList;
```

Main methods of ArrayList:

.add(Object object) – adding an object to the collection
.remove (Object object) – remove an object from the collection
.get(index) – get an object from the collection by index
.clear() – clear the collection (remove all elements)
.size() – get the size of the collection (number of elements)

Description of multiple connections using the Tree class as an example:

```
// Tree class declaration
class Tree {
   // Description of the name property
          ...

   // Description of the property Fruit
   private ArrayList fruits = new ArrayList();

   // Get all the fruits on a given tree
   public ArrayList getFruits() {
            return fruits;

      }

   // Add fruit
   public void addFruit(Fruit fruit) {
             fruits.add(fruit);

      }

   // Remove fruit
   public void removeFruit(Fruit fruit) {
             fruits.remove(fruit);

      } }
```

Getting a collection element

As noted, getting a collection element is done using the .get(int index) method. Because all objects stored in the collection are cast to type Object, you must convert the element you read back to its original type. Casting to a type is performed according to the following syntax:

(<Type name>) <object reference>

For example, let **tree1** be a reference to an object of the **Tree** class. The **Tree** class contains a collection of **fruits** objects of type **Fruit**; the **getFruits()** method provides access to the **fruits** collection. It is required to display the names (hot **name**) of all fruits in **tree1**:

```
// list all elements of the collection tree1.getFruits()

for (int i=0; i<tree1.getFruits().size(); i++) {

   // we get the i-th element of the collection,

   // convert it to the Fruit type and assign it to the link

   fruit Fruit fruit = (Fruit)tree1.getFruits().get(i);

   // display the value of the name property of the fruit object,

   // which represents the i-th element of the collection

   System.out.println(fruit.getName());

}
```

TASKS

Variant № 1

a) **Model the structure of the enterprise: (add a class diagram to the report)**

Classes	**Properties**
Company	name (get, set)
Department	name (get, set) number of employees (get, set)
Employee	full name (get, set) position (get, set) salary (get, set)

Create one object of the Company class, two objects of the Department class, and three objects of the Employee class, set the property values, and display them on the screen.

b) Link the classes Company, Department and Employee, so that each company contains departments, each department contains information in which company it is located and what employees it contains, each employee - in which department he works.

- Add to the Company class a multiple property "departments" (get) and methods "add department" (add), "remove department" (remove).
- Add to the Department class the "company" property (get) and the "employee" property (get), as well as the "add employee" (add) and "remove employee" methods. Remove the set for the "number of employees" property and make sure that this property is calculated automatically (based on the "employees" property).
- Add the "Department" property to the "Employee" class
-

Create one object of the Company class, add two objects to this company - Departments, add two employees to the first department, and one employee to the second. Display the employees working in one of the departments and their number.

b) add a method to the Company class that searches for an employee by full name. The method contains the input parameter Full Name (String) and returns a value of type Employee. Carry out a search in main(), display the property values of the found employee on the screen, and display in which department he works.

Variant № 2

a) **Model the structure of the bank: (add a class diagram to the report)**

Classes	**Properties**
Bank	name (get, set)
Branch	name (get, set) total deposit amount (get, set)
Contribution	depositor's name (get, set) deposit amount (get, set)

Create one object of the Bank class, two objects of the Branch class, and three deposits, set the property values, and display them on the screen.

b) Link the classes Bank, Branch and Deposit with each other, so that each bank contains branches, each branch contains an indication of which bank it is located in and what deposits it contains, each deposit - in which branch it is located.

- Add a multiple property “branches” (get) and a method “add branch” (add) to the Bank class
- Add to the Branch class the “bank” property (get) and the “deposits” property (get), as well as the “add” methods
contribution" (add), "remove contribution" (remove). Remove the set for the “total deposit amount” property and do
so that this property is calculated automatically (based on the “contributions” property).
- Add the “Branch” property to the “Deposit” class,
- Remove from the “Deposit” class the set for the “deposit amount” property and add method “replenish account (amount)”

Create one object of the Bank class, add two branches to this bank, add two deposits to each branch. Display the deposits of one of the branches and the total amount of deposits for this branch. Top up the account of one of the deposits and again display information about the deposits on the screen.

c) add a method to the Bank class that searches for a deposit by the depositor’s full name. The method contains the input parameter Full Name (String) and returns a value of type Contribution. Carry out a search in main(), display the amount of the found deposit on the screen, and display in which branch it is located.

Variant № 3

a) Model the structure of the airport: (add a class diagram to the report)

Classes	Properties
Airport	name (get, set)
Aircraft	name (get, set) Max. number of passengers (get, set)
Passenger	name (get, set) № footprint (get, set)

Create one object of the Airport class, two objects of the Aircraft class, and three objects of the Passenger class, set property values, and display them on the screen.

b) Link the classes Airport, Aircraft and Passenger, so that each airport contains information about its aircraft, each aircraft - which airport it is located at and what passengers it contains, each passenger - which aircraft it is assigned to .

- Add to the Airport class a multiple property “aircraft_vehicles” (get) and methods “add aircraft” (add), “remove aircraft” (remove).
- Add to the Aircraft class the “airport” property (get) and the “passengers” property (get), as well as the “add passenger” (add) and “remove passenger” methods.
- Add the “Aircraft” property to the Passenger class

Create one object of the Airport class, add two objects to this airport - Aircraft, add one passenger to the first one, and three passengers to the second one. Display passengers assigned to one of the aircraft.

c) add a method to the Airport class that searches for a passenger by airline name and seat number. The method contains input parameters Name of vehicle (String) and Seat No. (int) and returns a value of type Passenger. Carry out a search in main() and display the property values of the found passenger on the screen.

Variant № 4

a) **Model the library structure: (add a class diagram to the report)**

Classes	**Properties**
Library	name (get, set)
Department (by genre)	genre name (get, set) number of editions (get, set)
Edition	name (get, set) author (get, set) year of publication (get, set)

Create one object of the Library class, two departments, and three publications, set property values, display them on the screen.

b) Link the classes Library, Department and Publication, so that each library contains information about which departments it contains, each department - in which library it is located, and what publications it contains, each publication - in which department it is located.

- Add to the Library class a multiple property “departments” (get) and methods “add department” (add), “remove department” (remove).

- Add to the Department class the property “library” (get) and the property “edition” (get), as well as the methods “add edition” (add), “remove edition” (remove). Remove the set for the “number of editions” property and make sure that this property is calculated automatically (based on the “editions” property).

- Add the “Department” property to the “Edition” class

Create one object of the Library class, add three objects to this library - Departments, add two publications to the first department, and one publication each to the second and third. Display the name and number of publications for each department of this library. Delete the second section, repeat the output of all library sections.

c) add a method to the Library class that searches for publications by a given year of release. The method contains the input parameter year of issue (int) and returns a collection or array of objects of type Edition. Carry out a search in main(), display the found publications on the screen, and display in which department they are located.

Variant № 5

a) **Model the structure of a cellular communication company: (add a class diagram to the report)**

Classes	**Properties**
Company	company name (get, set)
Tariff	tariff name(get, set)
Subscriber	employee name (get, set) phone number (get, set) account balance (get, set)

Create two objects of the Company class, two tariffs, and three subscribers, set property values, display them on the screen.

b) Link the classes Company, Tariff and Subscriber with each other, so that each Company contains Tariffs, each Tariff contains information about which company it is used in and which subscribers it contains, each subscriber - which tariff it is connected to.

- Add a multiple property “tariffs” (get) and a method “add tariff” (add) to the Company class
- Add to the Tariff class the “company” property (get) and the “subscribers” property (get), as well as methods
“add subscriber” (add), “remove subscriber” (remove).
- Add “number of subscribers” (get) to the Tariff class and make this property
calculated automatically (based on the multiple property “subscribers”).
- Add the “Tariff” property to the “Subscriber” class,
- Remove the set for the “account balance” property from the “Subscriber” class and add
method “replenish account (amount)”

Create two objects of the Company class, add one tariff to each company, connect two subscribers to the first tariff in the first company, and one subscriber to the second. Display subscribers of one of the companies. Top up the account of one of the subscribers and again display information about subscribers on the screen.

c) add a method to the Company class that searches for a subscriber by phone number. The method contains the input parameter Phone number (String or int) and returns a value of type Subscriber. Carry out a search in main(), display the name and tariff of the found subscriber on the screen.

Variant № 6

a) Model the structure of a car dealership: (add a class diagram to the report)

Classes	**Properties**
Car showroom	name (get, set)
Car (make)	brand name (get, set) Max. number of passengers (get, set) cost (get, set) quantity in stock (get, set) boolean availability (get, set)
Purchase Request	buyer's name (get, set) phone number (get, set)

Create one object of the Car Dealership class, two cars, and three requests, set property values, display them on the screen.

b) Link the classes Car Dealership, Car and Purchase Request, so that each Car Dealership contains Cars, each Car contains information about which car dealership it is for sale in and about applications for its purchase, each requisition for which car she relates.

- Add a multiple property “cars” (get) and a method “add a car” (add) to the Car Dealership class.
- Add to the Car class the “car dealership” property (get) and the “purchase requisition” property (get), as well as the “add requisition” (add) and “remove requisition” methods. Remove the set for the “availability” property and make sure that this property is calculated automatically (based on the “quantity in stock” and “requisitions” properties).
- Add the “car” property to the “Purchase Request” class,

Create one object of the Car Dealership class, add two cars to it. For the first one, fill out one application, for the second - three applications. Display information about one of the cars and applications for it.

c) add a method to the Car Dealership class that searches for a car by brand name. The method contains an input parameter Brand name (String) and returns a value of type Car. Carry out a search in main(), display information about the found car on the screen.

Variant № 7

a) **Model the structure of the music collection: (add a class diagram to the report)**

Classes	**Properties**
Collection	name (get, set) owner's name (get, set)
Musical medium (album)	author/group (get, set) genre (get, set) year of manufacture (get, set) total duration of sound (get, set)
Musical composition	name (get, set) duration (get, set)

Create one object of the Collection class, two musical media (albums), and three musical works, set property values, display them on the screen.

b) Link the classes Collection, Musical Media and Musical Work with each other, so that each collection contains musical media, each media contains information about which collection it is in and what musical works it contains, each musical work - on what media it is contained.

- Add to the Collection class a multiple property “media” (get) and methods “add media” (add), “remove media” (remove).
- Add to the Music Media class the “collection” property (get) and the “musical works” property (get), as well as the “add a musical work” (add) and “remove a musical work” (remove) methods. Remove the set for the “total duration of sound” property and make sure that this property is calculated automatically (based on the “musical pieces” property).
- Add the “Musical Media” property to the “Musical Work” class
-

Create one object of the Collection class, add two media - albums - to this collection, add two works to the first album, and one work to the second. Display the musical works of one of the media and the total duration of the album.

c) add a method to the Collection class that searches for a piece of music by title. The method contains an input parameter Title (String) and returns a value of type Musical work. Carry out a search in main(), display information on the found work on the screen, and display on what media it is contained.

Variant № 8

a) **Model the structure of the urban housing registry: (add a class diagram to the report)**

Classes	**Properties**
City	name (get, set)
Building	street name (get, set) house number (get, set) basic monthly payment per sq.m. area (get, set)
Room	number (get, set) area (get, set)

Create one object of the City class, two buildings, and three objects - rooms, set property values, display them on the screen.

b) Link the classes City, Building and Room with each other, so that each city contains buildings, and each building contains rooms; Each room must also contain information about which building it belongs to.

- Add to the City class a multiple property “buildings” (get) and methods “add building” (add), “remove building” (remove).
- Add the “rooms” property (get) to the Building class, as well as the “add room” (add) and “remove room” methods. Add the “total area” property (get), calculated as the total area of all rooms in the building.
- Add the “building” property to the “Room” class

Create one object of the City class, two buildings, add two objects - rooms - to the first, and one room to the second. Display information about the first building (the values of all properties, including the total area) and the rooms in this building.

c) add a method to the City class that searches for a building by street name and house number. The method contains input parameters Street name (String) and House number (int or String) and returns a value of type Building. Search for a building in main(), display the found building (the values of all properties) on the screen.

Variant № 9

a) **Model the structure of the zoo:**

Classes	**Properties**
Zoo	name (get, set)
Aviary/cage	number (get, set) size (get, set) Max. number of animals (get, set) current number of animals (get, set)

Animal	name (get, set) boolean predator (get, set)

Create one object of the Zoo class, two objects - Cells, and three objects - Animals, set property values, display them on the screen.

b) Link the classes Zoo, Cage and Animal with each other, so that each zoo contains cages, each cage contains animals, each animal “knows” in which cage it is kept.
- Add to the Zoo class a multiple property “cells” (get) and methods “add a cell” (add), “remove a cell” (remove).
- Add the “animals” property (get) to the Cell class, as well as the “add animal” (add) and “remove animal” methods. Remove the set for the “current number of animals” property and make sure that this property is calculated automatically (based on the “animals” property).
- Add the “cell” property to the “animal” class
Create one object of the Zoo class, there are two cells in it, add two animals to the first, and one animal to the second. Display the animals contained in one of the cages.

c) add a method to the Zoo class that searches for an animal by name. The method contains an input parameter Name (String) and returns a value of type Animal. Carry out a search in main(), display the property values of the found animal on the screen.

Variant № 10

a) Model the structure of an automated ATM:

Classes	**Properties**
Bank	name (get, set)
Bank account	number (get, set) PIN code (get, set) remainder (get, set)
ATM	identification number (get, set) address (get, set)

Create one object of the Bank class, two objects - Accounts, and three objects - ATM, set property values, and display them.

b) Link the classes Bank, Account and ATM, so that each Bank contains Accounts and ATMs, each Account contains information about which bank it is contained in, each ATM – which bank it serves.
- Add to the Bank class a multiple property “accounts” (get) and methods “add account” (add), “remove account” (remove)
- Add to the Bank class a multiple property “ATMs” (get) and a method “add ATM” (add)
- Add the “bank” property to the Account class (get)
- Add the “bank” property to the ATM class (get)
- Remove the set for the “account balance” property from the “Account” class. Add methods to the Account class
“replenish account (amount)” and “withdraw from account (amount)”

Create one object of the Bank class, two accounts and three ATMs in it. Display the status of one of the accounts. Withdraw a certain amount from this account and again display information about the status of this account on the screen.

c) Add the “withdraw money from account” method to the ATM class with the “PIN code” and “amount” parameters. This method contacts the bank serviced by this ATM, searches for an account using this PIN code and withdraws the specified amount (by calling the “withdraw from account (amount)” method from the found account). In main(), check the operation of the created method.