National Research University Higher School of Economics
Faculty of Computer Science
Bachelor’s Program in Data Science and Business Analytics (DSBA)

# Introduction to Programming
# Final Exam - Programming Test

Spring Semester 2021/22

Consider an *abstract* class `Teacher` defined as follows:

```
class Teacher
{
public:
      Teacher(const std::string& name)
      : _name(name){}
      virtual ~Teacher() {}
      const std::string& getName() const { return _name; }
      virtual double calcWages() const = 0;
protected:
      std::string _name;
};
```

This class will serve as the base for defining and implementing derived classes of Teachers.

## Problem 1

Declare and implement two *derived* classes `AssociateTeacher` and `InvitedTeacher`, which will *inherit* from the abstract class `Teacher`.

**1.1.** The `AssociateTeacher` class must have a non-public field `int _bonus`, which determines additional payment to this kind of teacher. Also, a corresponding `int getBonus()` constant function that returns the `_bonus` field should be implemented.

**1.2.** The `InvitedTeacher` class must have a field `int _stuGroups`, which means the number of student groups that the teacher has assigned. Also, a corresponding `int getStuGroups()` constant function that returns the `_stuGroups` field should be implemented.

Also, both derived classes `AssociateTeacher` and `InvitedTeacher` *must* have:

**1.3.** *Argument* constructors to initialize the class fields: `_name` and `_bonus` or `_stuGroups` depending on the derived classes.

**1.4.** Implementation of the constant *virtual function* `int calcWages()` which calculates and returns the teacher's wages. For each derived class, the wages are calculated as follows:

| Derived Class | Wages |
|---|---|
| `InvitedTeacher` | `GROUP_TAX * _stuGroups` |
| `AssociateTeacher` | `BASE_SALARY + _bonus` |

Where `GROUP_TAX` and `BASE_SALARY` are integer constants given in the solution template file.

## Problem 2

Consider a class `TArray` that will work as a container of pointers to Teachers. To this end, the class will have a field `std::vector<Teacher*> _arr;`

Note that elements in `_arr` , which are pointers of the abstract class `Teacher`, will point to instances of the derived classes `InvitedTeacher` and `AssociateTeacher`.
Below, there is an illustration of the class that must be completed.

```
class TArray {
public:
	// TODO: To complete...
	size_t getSize() const
	{
	    return _arr.size();
	}
protected:
	std::vector<Teacher*> _arr;
};
```

Implement the following for the class `TArray`:

**2.1.** A ***destructor*** `~TArray()` that releases memory dynamically allocated for each element in `_arr`;

**2.2.** A function `AssociateTeacher* addAssociateTeacher(const std::string& name, int bonus)` that dynamically allocates in memory a *new* instance of the class `AssociateTeacher`, initializes the instance with the arguments `name` and `bonus`, and inserts the pointer AssociateTeacher* (which points to the new instance) in the vector `_arr`. The pointer to the new instance is returned by the function.

**2.3.** A function `InvitedTeacher* addInvitedTeacher(const std::string& name, int groupsNum)` that dynamically allocates in memory a *new* instance of the class `InvitedTeacher`, initializes the instance with the arguments `name` and `groupsNum`, and inserts the pointer InvitedTeacher* (which points to the new instance) in the vector `_arr`. The pointer to the new instance is returned by the function.

**2.4.** An **overload** of the **operator[]**: `Teacher* operator[](size_t index) const`, which will return the element at position `index` of `_arr`
Note: An *exception* of type `std::out_of_range()` must be thrown if the argument `index` is not in the range [`0`, `_arr.size()`- 1]

**2.5.** An **overload** of the **operator<<** outside the class definition

```
    std::ostream& operator<<(std::ostream& os, const TArray& tArr)
```

which will print in the standard output stream information about teachers (one per line), pointed by elements inside the container `_arr` of the TArray `tArr`. For each teacher, a name and his/her wages (calculated with `calcWages()` function) are printed separated by a comma, one teacher per line. There must be a line break `\n` at the end of each line.

For example, for the following code in the left column below, the desired output is shown in the right column. (Assuming `GROUP_TAX = 2000` and `BASE_SALARY = 1500`)

| Example test code | Example output |
|---|---|
| `TArray tArr;`<br>`tArr.addInvitedTeacher("John Caine", 3);`<br>`tArr.addAssociateTeacher("Sam Sawyer", 1000);`<br>`std::cout << tArr;` | `John Caine,6000`<br>`Sam Sawyer,2500` |

## Problem 3

Implement the function:

```
std::pair<int,int> totalWagesOfTopTeachers(const TArray& tArr, int
minGroups, int minBonus)
```

that must return an object of type `std::pair<int,int>` where:

- `Its.first` is the sum of wages of all instances of the class `InvitedTeacher` which are pointed in `tArr` and whose number of groups `_stuGroups` is greater or equal than the argument `minGroups`
- `Its.second` is the sum of wages of all instances of the class `AssociateTeacher` pointed in `tArr` and whose `_bonus` is greater or equal than the argument `minBonus`.

**Hint:** Recall that given a pointer `_arr[i]` of the abstract class `Teacher`, it is indeed possible to determine the concrete derived class of the instance to which `_arr[i]` is pointing using the type conversion operator `dynamic_cast`.

For example, for the following code in the left column below, the desired output is shown in the right column. (Assuming `GROUP_TAX = 2000` and `BASE_SALARY = 1500`)

| Example test code | Example output |
|---|---|
| <pre>TArray tArr;<br>tArr.addInvitedTeacher("John Caine", 4);<br>tArr.addInvitedTeacher("Samuel Johnson", 2);<br>tArr.addAssociateTeacher("Sam Sawyer", 1000);<br>tArr.addAssociateTeacher("Angela Davies",<br>2000);<br><br>std::pair&lt;int,int&gt; p =<br>totalWagesOfTopTeachers(tArr, 3, 1000);<br><br>std::cout &lt;&lt; p &lt;&lt; std::endl;<br><br>/* Teachers pointed by tArr and their wages:<br>   -  John Caine,8000<br>   -  Samuel Johnson,4000<br>   -  Sam Sawyer,2500<br>   -  Angela Davies,3500<br> In this example, the function<br>totalWagesOfTopTeachers(tArr, 3, 1000)<br>returns the sum of all InvitedTeachers whose<br>number of groups is greater or equal than 3,<br>and the sum of all AssociateTeachers with a<br>bonus greater or equal than 1000<br>*/</pre> | <pre>8000 6000</pre> |

## Problem 4*

Force the class `TArray` to comply with the "Rule Of Three". In order to do that you have to extend the class by adding some additional members. Recall which ones are needed.

When copying elements of the `TArray` origin, i.e. the objects behind the `Teacher*` pointers, you must do a deep copy, not a shallow copy. Thus, it is necessary to understand which type of objects are actually behind the `Teacher*` pointers. You must NOT use the `dynamic_cast` operator this time. Instead, declare a pure virtual function `Teacher* cloneMe()` in the class `Teacher`, and override it correspondingly in the derived classes `AssociateTeacher` and `InvitedTeacher`. Each overridden version of the function `cloneMe()` creates a new instance of a respective derived class dynamically, initializing it with the property of an origin object. Then the new instance is returned via a `Teacher*` pointer with an implicit up-cast.

Do not duplicate code. When implementing yet another copy operation, consider using a useful idiom for doing that.

* A starred task of increased complexity.