ASTANA IT UNIVERSITY	**Cours:** Object-Oriented Programming **Deadline:** January 8, 2024 till (23:59)

Instructor: Toleu Altynbek

ASSIGNMENT 3

Lesson plan:

Methods
Introduction to OOP
Objects
Classes

Methods in Java

You have already learned a large number of commands in Java, which means that you can write fairly complex programs. 10, 20, 30 lines of code in a program is not that big of a program, right?

But a program of 100+ lines is already large, and understanding its code is quite difficult. Is it possible to somehow make writing and reading programs with a lot of code easier?

Yes, and methods (functions) will help us with this.

What is a method? Well, to greatly simplify it, a method is a group of commands that has a unique name. In other words, we simply put several commands into one group and give it a unique name. And that's it – the method is ready.

Example:

No method	With method

`class Solution`
`{`
`  public static void main(String[] args)`
`  {`
`  System.out.print("Wi-");`
`  System.out.println("Fi");`
`  System.out.print("Wi-");`
`  System.out.println("Fi");`
	`class Solution`
	`{`
	`  public static void main(String[] args)`
	`  {`
	`  printWiFi();`
	`  printWiFi();`
	`  printWiFi();`
	`  }`

System.out.print("Wi-");
System.out.println("Fi");
}
}
	public static void printWiFi()
	{
	System.out.print("Wi-");
	System.out.println("Fi");
	}
	}

Method declaration in Java

So what is the correct way to write your method?

When declaring (creating) a method, there are many nuances to consider, but let's start with the basics. How can we still declare the simplest method? A simple method declaration looks like this:

```
public static void name()
{
    method code
}
```

Where name is the unique **name** of the method, and method code is the commands that make up the method. We will look at the meaning of the words **public, static** and **void** a little later.

The method code may contain calls to other methods:

Code	Note

class Solution
{
public static void main(String[] args)
{
printWiFi10Times();
}
 	
public static void printWiFi10Times()
{
for (int i = 0; i < 10; i++)
printWiFi();
}
 	
public static void printWiFi()
{
System.out.print("Wi-");
System.out.println("Fi");
	Calling the method printWiFi10Times()
	
	Declare the method printWiFi10Times
	
	Call the printWiFi() method 10 times in a loop
	
	Declare the method printWiFi
	
	We display the text on the screen: Wi-Fi

Code	Note
}}	

Task 1

The main method prints too many lines of information about various concepts.
It would be logical to separate the output of information about hydrogen and about the island of Java into different methods.
Create a public static void printHydrogenInfo() method that should print all lines about hydrogen.
Also create a public static void printJavaInfo() method that will print all the strings about the island of Java.
Finally, in the main() method, call these two methods.
The overall console output should not change.

Requirements:
• There must be a public static void printHydrogenInfo() method.
• There must be a public static void printJavaInfo() method.
• The printHydrogenInfo() method should print all existing hydrogen strings.
• The printJavaInfo() method should print all existing strings about the island of Java.
• The main() method must call the printHydrogenInfo() and printJavaInfo() methods.

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

       System.out.println("Hydrogen:");
       System.out.println("Java Island:");
       System.out.println("Part of Indonesia.");
       System.out.println("This is a chemical element of the periodic table with symbol H and atomic number 1.");
       System.out.println("Is the lightest element of the table.");
       System.out.println("Hydrogen is used in:");
       System.out.println("The states of Mataram, Majapahit, Demak were born on the island.");
       System.out.println("Java is the most populated island in the world:");
       System.out.println("-Chemical industry;");
       System.out.println("-Oil refining industry;");
       System.out.println("Total population - 140 million people.");
       System.out.println("Population density - 1061 people/sq. km.");
       System.out.println("One of the famous varieties of coffee, Kopi Luwak, is produced here.");
       System.out.println("-Aviation;");
       System.out.println("-Electricity.");
   }
```

Passing parameters

And now the fun begins. As you probably already know from methods like System.out.println(), parameters can be passed to methods. Which, in fact, greatly increases the benefits of creating and using methods.

So how do we declare a method with parameters? It's actually quite simple:

```
public static void name(parameters)
{
  method code
}
```

Where **name** is the unique name of the method, and method code is the commands that make up the method. And parameters are method parameters, separated by commas. Let's better describe this pattern in more detail:

```
public static void name(type 1 name 1, type 2 name 2, type 3 name 3)
{
  method code
}
```

Code	Note

`class Solution`
`{`
`  public static void printLines(String text, int count)`
`  {`
`  for (int i = 0; i < count; i++)`
`  System.out.println(text);`
`  }`
 	
`  public static void main(String[] args)`
`  {`
`  printLines("Hello", 10);`
`  printLines("Пока", 20);`
`  }`
`}`
	Declared the printLines method with parameters:
	String text, int count
	The method displays the string text count times
	
	Call the printLines method with parameters
	text = "Hello"; count = 10;
	text = "Bye"; count = 20;

Task 2

Rename the parameters of the printPersonInfo() method:

firstName in name;
lastName in surname;
favoriteDish in meal;
so that the functionality of the program remains unchanged.
Leave the variable names in the main() method unchanged.
Print to the console using the method.

```
public static void main(String[] args) {
        String firstName = "Sultan";
        String lastName = "Suleyman";
        String favouriteDish = "Steak";
        printPersonInfo(firstName, lastName, favouriteDish);
    }

    public static void printPersonInfo(String firstName, String lastName, String favouriteDish){

    }
```

Task 3

Let's write a utility for working with arrays. The main part of the functionality, the printArray() method displays all array elements in the console.
Another method is reverseArray(). It should reverse the order of the array elements.
The method should only work with arrays of integer values (int[]).

Example:
If the array contained elements:
1, 2, 3, 4, 5, 6, 7, 8, 9, 0
then after calling the reverseArray() method it should contain:
0, 9, 8, 7, 6, 5, 4, 3, 2, 1

Requirements:
• The reverseArray() method must reverse the order of the array elements.

Task 4

It would be nice to have one method for solving different problems. You have the opportunity to write one.
Create 9 more universalMethod() methods. In total there should be 10 of them.

Come up with the parameters they should accept yourself.
Requirements:
• The program must contain 10 methods named universalMethod.

Task 5

The program should display the population of the largest cities in the world and compare them with the largest city - Tokyo. But the program algorithm is a little broken. To fix the error, you need to make sure that in the line displaying information about the largest city in the world, static class variables are used instead of local method arguments.

If the program is running correctly, the output should be as follows:
The population of the city of Jakarta is 25.3 million people.
While the most populous city of Tokyo has a population of 34.5 million people.
The population of the city of Seoul is 25.2 million people.
While the most populous city of Tokyo has a population of 34.5 million people.
The population of the city of Delhi is 23.1 million people.
While the most populous city of Tokyo has a population of 34.5 million people.
The population of New York City is 21.6 million people.
While the most populous city of Tokyo has a population of 34.5 million people.

Requirements:
• Variable names cannot be changed.
• To display information about the largest city, static variables of the city and population classes must be used.
• The output of the program in the console must correspond to the conditions of the task.

```
public class Solution {
    public static String city = "Токио";
    public static double population = 34.5;

    public static void main(String[] args) {
        printCityPopulation("Джакарта", 25.3);
        printCityPopulation("Сеул", 25.2);
        printCityPopulation("Дели", 23.1);
        printCityPopulation("Нью-Йорк", 21.6);
    }

    public static void printCityPopulation(String city, double population){
    }
}
```

Introduction to OOP principles

Today you will discover a new and interesting world. This world is called OOP - object-oriented programming. You were already introduced to classes and objects earlier. Today you will learn even more about them, much more.

OOP emerged as a response to the increasing complexity of programs. When the number of variables and functions in the first programs began to number in the tens of thousands, it became clear that something had to be done about it. One solution was to combine the data and the functions that operated on it into separate objects.

Now programmers had to separately describe the interaction of objects and the actions that occurred inside the object. This greatly simplified the understanding and writing of programs. However, the question remained open: which functions should be inside an object, and which ones should be between objects.

We tried many approaches, and based on best practices, we formulated 4 principles of OOP. These are: **abstraction, encapsulation, inheritance and polymorphism.** Previously there were only three of them, but then they decided to add abstraction.

Abstraction

On the Internet, people are still arguing over the definition of abstraction in OOP. And the problem is not even that everyone is wrong, but that everyone is right. The smaller the program, the more strongly the abstraction is tied to the Java language; the larger, the more tied to the modeling/simplification of real-world objects.

But it seems that the best minds agreed on the following:

Abstraction is the use of only those characteristics of an object that represent it in the program with sufficient accuracy. The main idea is to represent an object with a minimal set of fields and methods and at the same time with sufficient accuracy for the problem being solved.

In the Java programming language, abstraction is accomplished through the use of abstract classes and interfaces.

Encapsulation

The purpose of encapsulation is to improve the quality of how things interact by simplifying them.
And the best way to simplify something is to hide everything complex from prying eyes. For example, if you are put in the cockpit of a Boeing, you will not immediately figure out how to control it.
On the other hand, for airplane passengers everything looks simpler: buy a ticket, board the plane, take off and land. You can easily fly from continent to continent with only the skills to "buy a ticket" and "board a plane." All the difficulties in the form of preparing the aircraft for flight, takeoff, landing and various emergency situations are hidden from you. Not to mention satellite navigation, autopilot and airport control centers. And this makes our life easier.

In programming terms, encapsulation is “hiding the implementation.” I like this definition. Our class can contain hundreds of methods and implement very complex behavior in different situations. But we can hide all its methods from prying eyes (mark them with the private modifier), and leave only a couple of methods for interaction with other classes (mark them with the public modifier). Then all other classes of our program will see only three methods in this class and call them. And all the difficulties will be hidden inside the classroom, like the cockpit from happy passengers.

Inheritance

There are two sides to inheritance. The programming side and the real life side. In programming terms, inheritance is a special relationship between two classes. But what is much more interesting is what inheritance is from the point of view of real life.

If we need to create something in real life, we have two solutions:

Create the thing we need from scratch, spending a lot of time and effort.
Create the thing we need based on something that already exists.
The most optimal strategy looks like this: we take an existing good solution, modify it a little, adjust it to our needs and use it.

If we trace the history of the emergence of man, it turns out that billions of years have passed since the origin of life on the planet. And if we imagine that man arose from a monkey (based on a monkey), then only a couple of million years have passed. Creating from scratch takes longer. Much longer.

In programming, it is also possible to create one class based on another. The new class becomes a descendant (heir) of the existing one. This is very beneficial when there is a class that contains 80%-90% of the data and methods we need. We simply declare the appropriate class as the parent of our new class, and all the data and methods of the parent class automatically appear in the new class. Isn't it convenient?

Polymorphism

Polymorphism is a concept from the field of programming. It describes a situation where different implementations are hidden behind one interface. If you try to look for its analogues in real life, one of them will be the process of driving a car.

If a person can drive a truck, he can be put behind the wheel of an ambulance or a sports car. A person can control a car regardless of what kind of car it is, because they all have the same control interface: steering wheel, pedals and gear lever. The internal structure of the machines is different, but they all have the same control interface.

If we return to programming, polymorphism allows objects of different classes (usually having a common ancestor) to be accessed uniformly - a thing that is difficult to overestimate. The larger the program, the higher its value.

OOP is about principles. Internal laws. Each of them limits us in some way, giving in return great advantages when the program grows to large sizes. The four principles of OOP are like the four legs of a stool. Take away even one, and the whole system becomes unstable.

Creating an Object

Now we have come to creating objects. You've already encountered this before, but now we'll look at this topic in more detail. It's actually very easy to create objects.

To create an object, you need to use the new operator. Creating an object looks something like this:

```
new Class(parameters)
```

Most often, immediately after creating an object, we save a link to it in a variable, which also very often has the same type as the object being created. So typically you'll see object creation code that looks like this:

```
Class name = new Class(parameters)
```

Where the Class name is the creation of a new variable, and the code to the right of the equal sign is the creation of a new object of type Class.

Examples:

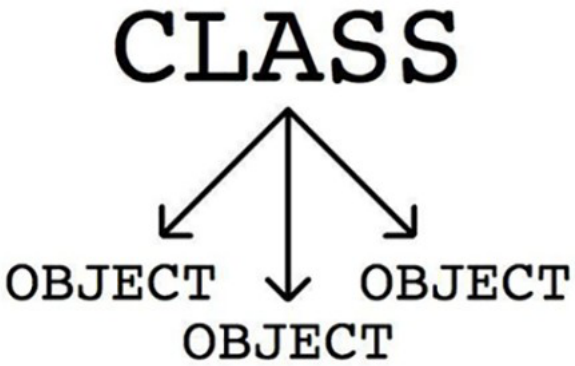
Code	Note
Object o = **new** Object();	Creating an object of type Object
Cat pet = **new** Cat();	Creating an object of type Cat
Scanner console = **new** Scanner(System.in)	Creating an object of type Scanner

Very often programmers call variables the same name as classes, only with a small letter. For a novice programmer, this code can be confusing:

Code
BufferedReader bufferedReader = **new** BufferedReader(reader);
Cat cat = **new** Cat();
PersonInfo personInfo = **new** PersonInfo()

Class in Java

A class is a model, template, or blueprint of an object. It describes the behavior and states what an object of its type supports. For example, the Cat class has its own name, color, owner; The cat also has behavior: eating, purring, walking, sleeping.
Another example, we have a plan for a building and this plan is a class and houses built according to this plan are objects. we can have only one plan but many objects.



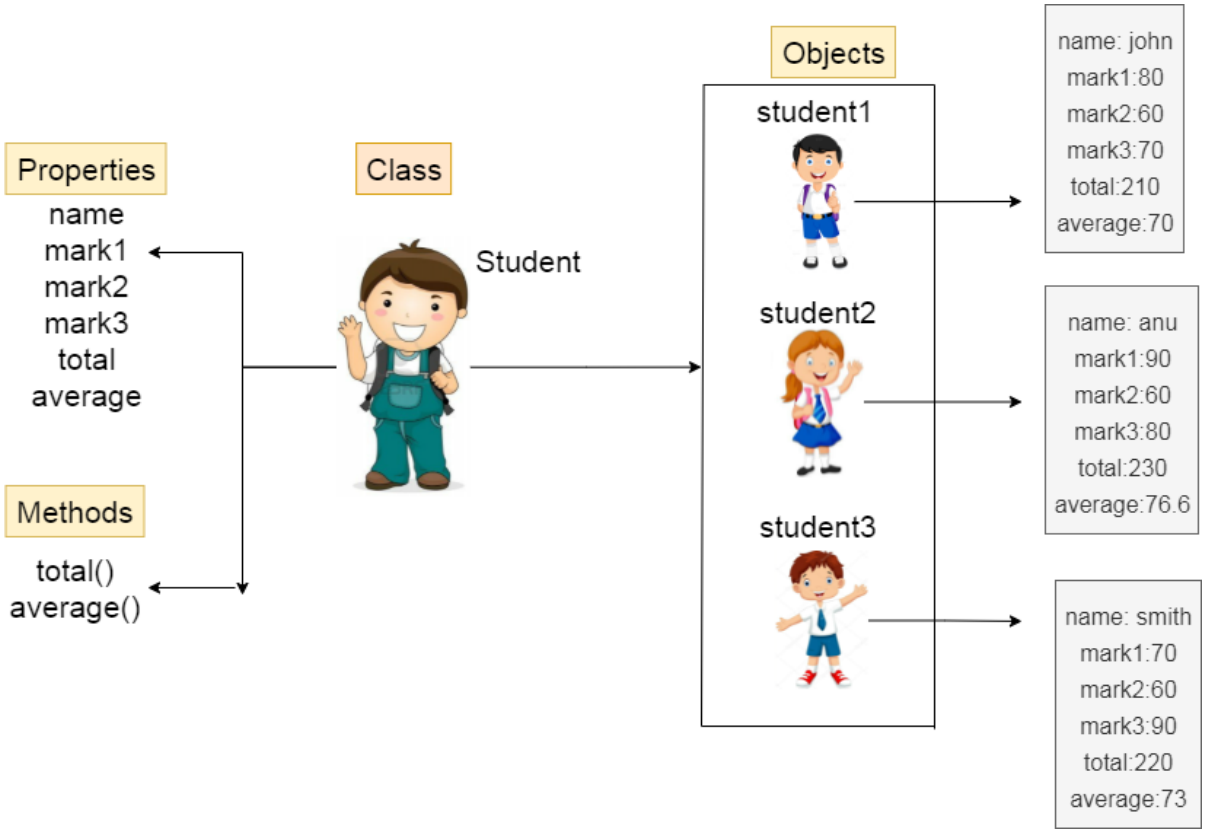
Objects in Java

Objects in Java have states and behavior.

Here's an example. The cat has a state: his name is Barsik, his color is red, his owner is Ali. The cat also has behavior: now Barsik is sleeping. He can also purr, walk and so on. An object is an instance of a class.
Just like objects in the real world, in programming objects have properties and methods. For example, a regular button has a color, size, and action when pressed.

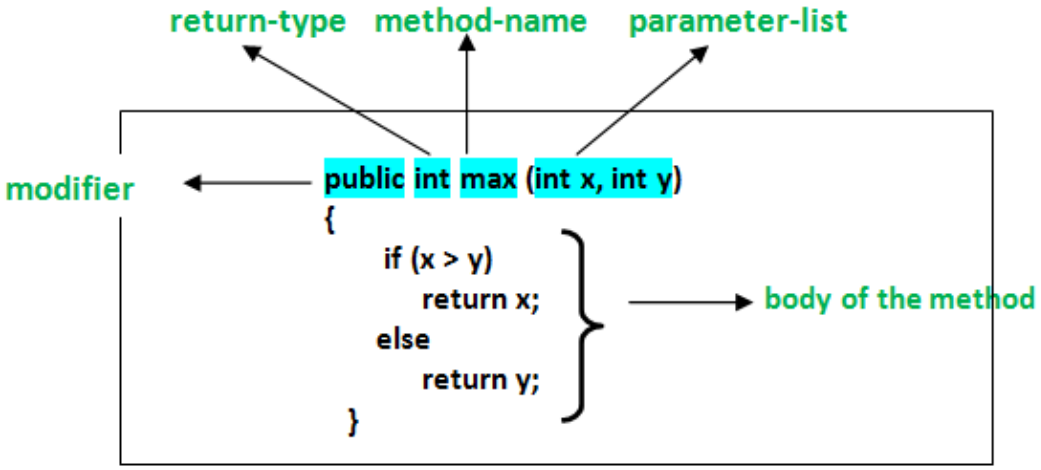
```
public class Cat {

   String name;
   int age;
   String color;
   String owner;

   public void sayMeow() {
      System.out.println("Meau!");
   }

   public void sleeping() {
      System.out.println("sleep!");
   }
}
```

```
public static void main(String[] args) {
        Cat barsik = new Cat();
        barsik.age = 3;
        barsik.name = "Barsik";
        barsik.color = "Red";
        barsik.owner = "Ali";

        barsik.sayMeow();
        barsik.sleeping();

    }
```



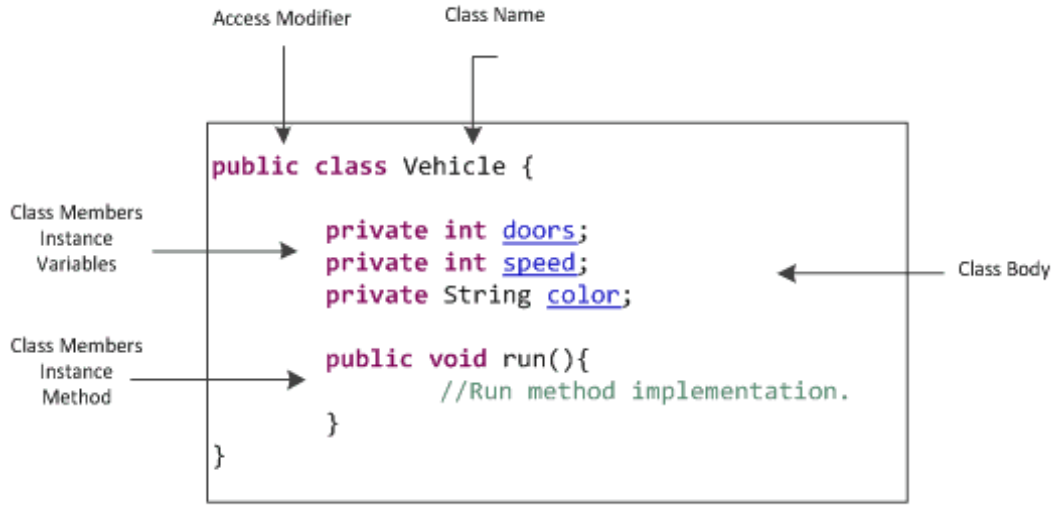
Methods in Java

Methods are intended to describe logic, work with data and perform all actions. Each method defines a behavior. A class can contain many methods. For example, we can write a sleep() method for the Cat class (to sleep) or a purr() method for purring.



Instance Variables in Java

Each object has a unique set of instance variables. The state of an object is typically formed by the values that are assigned to these instance variables. For example, the cat's name or age can be variable.



Java is statically typed and also a strongly typed language because, in Java, each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

Constructor

You've probably often seen that when an object is created, some parameters are passed to it. Moreover, parameters are passed to some objects, but not to others. How does this whole mechanism with parameters actually work?

Everything is simple here too: each class has a special method (or methods) that are responsible for processing parameters when creating an object. Such methods are called constructors. And one method, respectively, is a constructor.

It is not difficult to distinguish a constructor method from a regular method in code. This method has two features:

the name of the constructor method is the same as the name of its class (and begins with a capital letter)
The constructor method does not have a return type.
Its general appearance is usually like this:

```
modifiers Class(parameters)
{
   code
}
```

Code	Note

`public class Point`
`{`
`  public int x;`
`  public int y;`
 	
`  Point(int x, int y)`
`  {`
`  this.x = x;`
`  this.y = y;`
`  }`
`}`
	Point class
	
	Point class constructor

`public class Solution`
`{`
`  public static void main(String[] args)`
`  {`
`  Point point = new Point(5, 10);`
`  }`
`}`
	When an object of the Point class is created, the class constructor will be called.

Task 6

Let's try to construct our first skyscraper and announce the result in the console. To do this, you need to call the constructor of the Skyscraper class in the main method and in the body of the constructor display the text “The skyscraper has been built.” in the console.

Requirements:
• The Skyscraper class must have a public constructor without parameters.
• In the main method you need to create one object of the Skyscraper class.
• The Skyscraper class constructor should display the text "The skyscraper has been built." to the console.

Task 7

Let's build a residential complex of three high-rise buildings. To do this, we will use three different ways to display information:

Announces the result of construction.
Announces the result and indicates the number of floors.
Announces the result and indicates the developer.

Example output:
The skyscraper has been built.
The skyscraper has been built. Number of floors - 50
The skyscraper has been built. Developer - JavaRushDevelopment

To solve the task, you need to declare three different constructors and display text in them. The main method is not involved in testing.
Requirements:
• The Skyscraper class must have three public constructors.
• The Skyscraper class must have a parameterless constructor.
• The Skyscraper class must have a constructor with a parameter of type int.
• The Skyscraper class must have a constructor with a String parameter.
• In a parameterless constructor, the output must match the example in the condition.
• In a constructor with a parameter of type int, the output must match the example in the condition.
• In a constructor with a String parameter, the output must match the example in the condition.

```
public class Skyscraper {
   public static final String SKYSCRAPER_WAS_BUILD = " The skyscraper has been built.";
   public static final String SKYSCRAPER_WAS_BUILD_FLOORS_COUNT = " The skyscraper has been built. Number of floors -";
   public static final String SKYSCRAPER_WAS_BUILD_DEVELOPER = " The skyscraper has been built. Developer -";
```

```
    // write your code here

    public static void main(String[] args) {
        Skyscraper skyscraper = new Skyscraper();
        Skyscraper skyscraperTower = new Skyscraper(50);
        Skyscraper skyscraperSkyline = new Skyscraper("JavaRushDevelopment");
    }
}
```

Task 8

Let's build a business center that consists of two buildings, one of which is just starting to be built, and the second is at the planning stage, so little is known about it.
You will need to create two constructors: one with parameters, the second without. They should both initialize the floorsCount and developer fields.

Requirements:
• The Skyscraper class must have two public constructors.
• The Skyscraper class must have a parameterless constructor.
• The Skyscraper class must have a constructor with parameters of type int and type String.
• The Skyscraper class must have a private non-static field of type int called floorsCount.
• The Skyscraper class must have a private, non-static String field called developer.
• In a parameterless constructor, the fields must be initialized to "5" and "JavaRushDevelopment".
• In a constructor with parameters, the fields must be initialized with the arguments of that constructor.

```
public class Skyscraper {
    private int floorsCount;
    private String developer;

    // write your code here

    public static void main(String[] args) {
        Skyscraper skyscraper = new Skyscraper();
        Skyscraper skyscraperUnknown = new Skyscraper(50, " Unknown ");
    }
}
```

Task 9

Feel like a car designer and find a solution to make the plant work again. To do this, you need to add initialization of fields in constructors with the appropriate parameters. If the parameter is missing, then you need to initialize the field with the default value. For the year field this is the current year (4321), for the color field it is Orange.

Requirements:
• The CarConcern class must have four private final fields.
• The CarConcern class must have a public constructor with three parameters that initializes the appropriate fields.
• The CarConcern class must have a public constructor with two parameters that initializes the appropriate fields.
• The CarConcern class must have a public constructor with one parameter that initializes the appropriate fields.

```
public class CarConcern {
    private final String manufacturer = "Lamborghini";
    private final String model;
    private final int year;
    private final String color;

    public CarConcern(String model, int year, String color) {
        // write your code here
    }

    public CarConcern(String model, int year) {
        // write your code here
    }

    public CarConcern(String model) {
        // write your code here
    }
}
```

Task 10

The construction of the building was planned as a restaurant and was successfully completed, but after some time the owners decided to convert it into a barbershop. We need to make sure that the building is universal, and its purpose can be changed without creating a new one.
To do this, create an initialize method that will set the value of the type field (to determine the type of building), and remove the constructor.

Requirements:
• The Building class must have a private field type of type String.
• There should be no declared constructors in the Building class.
• The Building class must have a non-static public void initialize method with a parameter of type String.

• Initialization of the type field must be in the initialize(String) method.

```
public class Building {
    private String type;

    public Building(String type) {
        this.type = type;
    }

   // write your code here

    public static void main(String[] args) {
        Building building = new Building("Ресторан");
        building.initialize("Барбершоп");
    }
}
```

Task 11

School Subject Class: Create a Subject class that stores information about a school subject (subject name, number of hours). Implement methods for setting and getting information about the subject.

Task 12

Working with Students: Write a Student class that contains information about a student (name, age, GPA). Implement methods to display student information and update their GPA.

Task 13

Creating a Triangle Class: Implement a Triangle class that stores information about the sides of a triangle. Add a method to calculate the perimeter and area of the triangle.

Task 14

Library Management: Create Book and Library classes. Book should contain information about a book (title, author, publication year). Library should have methods to add a book to the library and display the list of books.

Task 15

ATM Management: Write an ATM class that has methods to check the balance, withdraw, and deposit funds into an account.

Task 16

Working with Shapes: Design classes Square and Circle. Square should store information about the side length, and Circle about the radius. Add methods to calculate the area and perimeter for each shape.