1. **Class**
   - A class is like a blueprint for creating an object(instances).
   - It holds the attributes or details about an object.

   Ex:

```
class Person{
       protected String name;
       protected int age;
}
```

   - The “**Person**” here is the class, it holds some basic **attributes or properties** of a person which is a name and age.
   - No parenthesis is needed when declaring a class, don’t confuse it for a method.
   - Declaring classes beginning with capital letters is a good practice, following Java’s class naming convention.

2. **Constructors**
   - A constructor is a special method that gets called when an object of the class is created.
   - Its purpose is to set up the object with initial values.

   Ex:

```
class Person {
    protected String name;
    protected int age;

    // Below is an example of a constructor
    // The constructor takes parameters and assigns them to the class's attributes
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

   - The keyword “**this**” refers to the current instance of the class (the current object).
   - It helps distinguish between the class’s attributes (name and age) and the constructor’s parameters (which also have the same names: name and age).

An analogy using LEGO for both classes and constructors:

Class:

Think of the class as the LEGO instruction manual. It defines how the model (object) will look and what parts it will have. However, it's just a set of instructions—it doesn't give you a built model yet.

Constructor:

The constructor is the process of building the LEGO model using the manual (class). It's like putting the pieces together, following the steps from the manual. The constructor ensures that when the LEGO model (object) is created, it's put together correctly.

Full Analogy:

1. **Class = LEGO Instruction Manual:**
    - It contains all the details (parts list, steps) but doesn't build the model itself.
2. **Constructor = Assembly Process:**
    - It's the process of following the manual to create the actual LEGO model (object). Without it, the pieces would just sit there.
3. **Instance = Finished LEGO Model:**
    - Once the constructor finishes assembling the pieces, you have a completed model (instance). You can create multiple models (instances) from the same manual (class).

In programming, when you create an object, you're essentially using the constructor to assemble the pieces of the class into a functioning instance (LEGO model).

3. **Objects/Instances**
    - Instance is the actual object created from the class using its blueprint.
    - Each instance or object created has its own unique values for each attribute defined in a class.

Ex:

```
//below is the main method
public static void main(String[] args){

        Person person1 = new Person("Mark", 19);
        Person person2 = new Person("Anton", 20);

}
```

Explanation:

- Instance: The actual object created from the class, in this case, the **person1** and **person2** are the objects created from **Person** class.

When creating an instance, the process follows this pattern:

1. **Class name**: Start by typing the class name (Person in this case).
2. **Object name/identifier**: Choose an identifier (e.g., person1, person2).
3. **Equal sign (=)**: Use the assignment operator to assign the object to the variable.
4. **"new" keyword**: The new keyword is used to create the instance.
5. **Constructor with arguments**: Call the constructor (e.g., new Person("Mark", 19)) to initialize the object, passing values that match the constructor parameters.

4. **Methods/Behavior**
   - A method, also known as functions in other programming languages, are block of codes that perform specific task when called in the program.
   - The method is composed of access modifier, return type, identifier, parameters, body, and return statement.
   - The method when declared has to have parenthesis after the method name.

Ex 1:

```
//below is an example of a method
//this method has no return type and no return statement
public void viewInfo(){
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
}
```

Explanation:

1. **Access Modifier** - this defines the method's visibility or accessibility from other class or parts of program.
   - In this case: the "**public**" is the access modifier of this method.
2. **Return Type** - the data type of the value that the method will return.
   - In this case: The "**void**" is the return type of this method, and doesn't return any value.
3. **Identifier/Method Name** - name of the method declared following Java naming convention.
   - In this case: The method name is "**viewInfo**" which follows the camelCase naming convention.
4. **Parameters** – are inputs/values passed to be processed inside a method. A method can take zero or more parameters.
   - In this case: the viewInfo method takes **no parameters**, and doesn't require any input when called in the program.
5. **Method Body** – it is the block of codes enclosed in the "**{}**" that defines what the method does. It can contain statements, loops, and conditions.
   - In this case: The method body consists of two "System.out.println" statements, which print the name and age of the person.
6. **Return Statement** – if the return type of the method is not void, it must contain a return statement at the end of the code block returning the value to the user.
   - In this case: since the return type is "**void**" the method will not return anything.

Ex 2:

```
//below is an example of a method
//this method has a return type of a string and returns variable containing a string.
public String viewInfo(String name, int age){
        String info = "Name: " + name ", Age: " + age;

       return info;
       //returns the variable info which is a String.

       //you can also return a literal String. Below is the example
       //return "Name: " + name ", Age: " + age;

}
```

- This method takes parameters of String name and int age, which are passed into the code block for processing.

Analogy of a method with parameters and return type:

Imagine a **coffee machine** that takes inputs (like water, coffee beans, and sugar) and produces coffee.

- **Method**: The coffee machine is the method itself.
- **Parameters**: The ingredients (water, coffee beans, sugar) are the parameters you provide to the machine to make the coffee.
- **Return value**: The coffee you get is the return value of the method.

Breaking it down:

- When you call a method, it's like starting the coffee machine.
- The parameters are the specific ingredients you give to the machine, just like you provide inputs to a method.
- The machine (method) processes those ingredients (parameters) and returns coffee (the result).

5. **Access Modifiers**
   - It is a keyword that sets the level of access or visibility of for classes, methods, and instance fields/attribute.

   In Java there are four main access modifiers:

   1. **Public**
   - the code is accessible from any other classes.
   - For example, a public method or variable from another class can be called within the other classes.

   2. **Private**
   - The code is only accessible within the class it was declared in.
   - Private methods and fields cannot be called or accessed within other classes even with the same file or package.

   3. Protected
   - The code is accessible only within the same package or its subclass even if the subclass is in a different package.

   4. Default (package-private) (no modifier defined)
   - The code is accessible only within the same package.
   - If no access modifier is specified, it is package-private by default.