What is a programming paradigm?
A programming paradigm is a style of a programming language in terms of what concepts if follows and what features it has.

Examples of programming paradigms:
- Imperitive: a style in which the state of a program is changed using a sequence of statements.
- Object Oriented programming: a type of imperitive programming
- Functional
- Logical

Object Oriented programming is a type of imperitive programming in which code is represented using object (which contain multiple pieces of data) and actions on these object called methods.

Objects represent the nouns in a prgram. Examples of Objects:
A student, a bus, a circle, a loan, love

An object consists of data fields (or class variables) which store the state of the object. Each object will have a collection of methods which are the actions that can be performed on the object.

Suppose we have a Cup data type.

```
Cup c = new Cup();
```

A cup may have an empty() method that removes all liquid in the cup.

```
c.empty();
```

The object that comes before the . in a method call (in our case this is c) is called the invoking object.

Inside the definition of method empty() (this is in the Cup class) we can access the invoking object with the keyword this

```
public class Cup
{
    private double liquidAmt;

    // ...

    public void empty()
    {
        this.liquidAmt = 0; // can remove this. since there is no local variable called liquidAmt
    }
}
```

The data type (or type) of an object is what kind of thing it is. An object's data type tells the compiler/interpreter how the object can be used.

Objects of the same type are defined using a class. A class is code that defines a new data type which specifies what the data fields and mehtods of objects of this type will have.

An instance of a class C is an object whose data type is C.
An object is an instance of a class.

The new keyword in Java is used to create an instance of a class.
Ex:

```
Cup c = new Cup();
```

c is a reference variable to an instance of the Cup class.

A constructor is a special method of a class that is called when an instance of the class is created. Constructors are used to initialize and setup a new class instance.

```
public class Cup
{
    private double liquidAmt;

    public Cup()   // Constructor
    {
        liquidAmt = 0;
    }

    // ...
}
```

If you do not give a class a constructor, then Java will supply a public default one for us that thakes no arguments and has an empty body.

Method overloading
Two methods of the same class may have the same name as long as their corresponding parameters have different types or they have a different number of parameters. This is called method overloading.

The signature of a method is its name along with the list of data types of is parameters in order. A method's signature is what distinguishes it from other methods in the same class.

```
public class Walnut
{
    // ...

   public void crack(int force) // method (1)
   {
      // ...
   }

   pubic void crack(double force) // method (2)
   {
      // ...
   }
}
```

The signature of method (1) is: crack(int)
The signature of method (2) is: crack(double)

In main:

```
   Walnut w = new Walnut();
   w.crack(5); // calls method (1) since 5.0 is an int
   w.crack(5.0); // calls method (2) since 5.0 is a double
```

Which version of crack() to call is determined at compile time. This is called static binding.

Since a constructor is a method, it can also be overloaded. So, a class can have multiple constructors as longs as each constructor has a different signature.

A constructor can call another constructor of the same class (on the current object being constructed). This may be done as to not have to duplicate code.
To do this, we use the keyword this: this(/*add-arguments-here*/);
When making a constructor call using this, the line must be the first one in the constructor.

```
public class MagicConch
{
   private String specialMessage;

   public MagicConch(String sm) // constructor (1)
   {
      specialMessage = sm;
   }

   public MagicConch() // constructor (2)
   {
      this("not 4 U");
   }
}
```