Instance Methods:

- Instance methods in Java are functions or procedures defined within a class that operate on the instance (object) of the class.
- They can access and manipulate instance variables and perform actions specific to each instance of the class.
- Instance methods are called on instances of the class and can have parameters and return values.
- They are defined using the method signature, which includes the access modifier, return type, method name, and parameters.

Instance Variables:

1. Instance variables are non-static fields declared within a class but outside any method or block.
2. They hold unique data for each instance (object) of the class and define the object's state.
3. Instance variables are initialized when an object is created and are accessible within instance methods.
4. They can have different values for different instances of the class.

Instance Initialization Block:

- An instance initialization block is a block of code within a class that is executed when an instance of the class is created.
- It is used to initialize instance variables or perform certain actions before an object's constructor is called.
- Instance initialization blocks are executed in the order they appear in the class, along with the constructor, if present.
- They are particularly useful when you want to perform common setup tasks for all instances.

Instance Flow of Execution:

When an object of a class is created, the following sequence of events occurs:

- Memory is allocated for the object.
- Instance variables are initialized to their default values (if not explicitly initialized).
- The instance initialization blocks are executed in the order they are defined.
- The constructor is called to complete the initialization process.
- Now the object is fully initialized and can be used.

```
class Student {
  String name; // Instance variable
   // Instance initialization block
   {
      System.out.println("Instance initialization block: Initializing student object");
   }
   // Constructor
   public Student(String n) {
      name = n;
      System.out.println("Constructor: Creating student object with name " + name);
   }
   // Instance method
   public void introduce() {
      System.out.println("Hi, I'm " + name);
   }
}
public class Main {
   public static void main(String[] args) {
      Student student1 = new Student("Alice");
      Student student2 = new Student("Bob");

      student1.introduce();
      student2.introduce();
   }
}
```

Output :

Instance initialization block: Initializing student object
Constructor: Creating student object with name Alice
Instance initialization block: Initializing student object
Constructor: Creating student object with name Bob
Hi, I'm Alice
Hi, I'm Bob

Regular Expressions (RegEx)

We can search for multiple instances of a regular expression in a text using the Java Matcher class (java.util.regex.Matcher).

What is a Regex in Java?

A string of characters that makes up a pattern is known as a regular expression, or regex. We can use this specific pattern to identify matched strings whenever we are looking for any data. It can only be a single character, or it might be a more intricate design.

The Java regex package contains classes that enable the use of regular expressions for pattern modification and searching.

//modules and packages in Java

//module
import java.base;

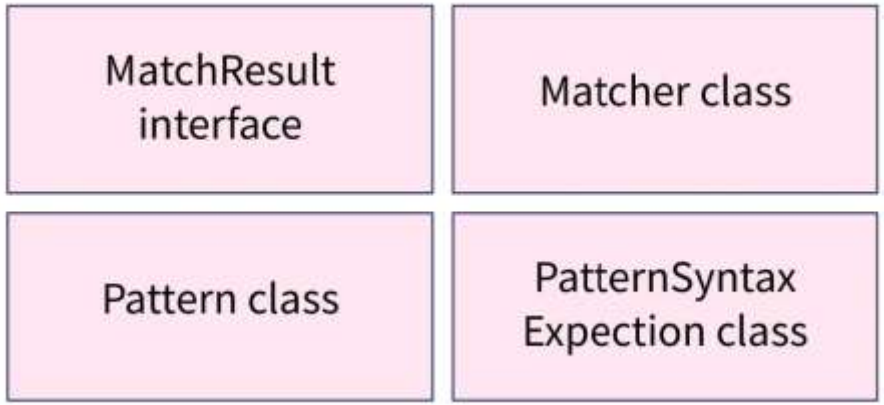
//Package
import java.util.regex.*;

There are three classes and one interface in the Java regex package:

MatchResult interface: It is a regex engine that is utilized to match characters in a sequence.
Pattern Class: To specify the pattern that needs to be searched
Matcher Class: To locate it by matching the pattern in the text.
PatternSyntaxException Class: To highlight any regular expression syntax mistakes.
Regex in Java

Its operation is fairly straightforward. To define a pattern, you must first build a Pattern object from the regular expression. The Pattern object is then used to generate a Matcher object.

There are numerous methods in the Matcher class. The **matches()** method, which returns true if the regular expression matches the text and false otherwise, is the most significant of these. In addition to the **replace()** methods in Java, the Java Matcher Class includes numerous other helpful methods for replacing text in input strings that carry out more complicated tasks.



Index Methods

Method Name	Description
public int start()	The preceding match's start index is returned by this procedure.
public int start(int group)	The start index of the subsequence that the specified group previously used in a match operation is returned by this method.
public int end()	The offset after the last character is matched is returned by this procedure.
public int end(int group)	The offset after the final character of the subsequence that the specified group successfully matched in the previous match operation is returned by this method.

Study Methods

Method Name	Description
public boolean lookingAt()	Starting at the beginning of the region, this technique attempts to match the input sequence against the pattern.
public boolean find()	This approach looks for the subsequent input sequence subsequence that meets the pattern.
public boolean find(int start)	Resets this matcher and then searches the input sequence starting at the specified index for the next subsequence that matches the pattern.
public boolean matches()	With this technique, the entire region is compared to the pattern.

Replacement Methods

Method Name	Description
public Matcher appendReplacement(StringBuffer sb, String replacement)	This method implements a non-terminal append-and-replace step.
public StringBuffer appendTail(StringBuffer sb)	This method implements a terminal append-and-replace step.
public String replaceAll(String replacement)	This method substitutes the supplied replacement string for each subsequence of the input sequence that matches the pattern.
public String replaceFirst(String replacement)	This technique substitutes the supplied replacement string for the first subsequence of the input sequence that matches the pattern.
public static String quoteReplacement(String s)	This method produces a String that will function as a literal replacement of the Matcher class in the appendReplacement method as well as a literal replacement String for the supplied String.