Java

Data Types

Data types are divided into two groups:

- Primitive data types - includes byte, short, int, long, float, double, boolean and char
- Non-primitive data types - such as String, Arrays and Classes

VARIABLES IN JAVA

A Java variable is a piece of memory that can contain a data value. A variable thus has a data type. In Java there are four types of variables:

- Non-static fields
- Static fields
- Local variables
- Parameters

A non-static field is a variable that belongs to an object. Objects keep their internal state in non-static fields. Non-static fields are also called instance variables, because they belong to instances (objects) of a class. Non-static fields are covered in more detail in the text on **Java fields**.

A static field is a variable that belongs to a class. A static field has the same value for all objects that access it. Static fields are also called class variables. Static fields are also covered in more detail in the text on **Java fields**.

A local variable is a variable declared inside a method. A local variable is only accessible inside the method that declared it. Local variables are covered in more detail in the text on **Java methods**.

A parameter is a variable that is passed to a method when the method is called. Parameters are also only accessible inside the method that declares them, although a value is assigned to them when the method is called. Parameters are also covered in more detail in the text on **Java methods**.

VARIABLE TYPES

In Java there are four types of variables:

- Non-static fields
- Static fields
- Local variables
- Parameters

A non-static field is a variable that belongs to an object. Objects keep their internal state in non-static fields. Non-static fields are also called instance variables, because they belong to instances (objects) of a class. Non-static fields are covered in more detail in the text on **Java fields**.

A static field is a variable that belongs to a class. A static field has the same value for all objects that access it. Static fields are also called class variables. Static fields are also covered in more detail in the text on **Java fields**.

A local variable is a variable declared inside a method. A local variable is only accessible inside the method that declared it. Local variables are covered in more detail in the text on **Java methods**.

A parameter is a variable that is passed to a method when the method is called. Parameters are also only accessible inside the method that declares them, although a value is assigned to them when the method is called. Parameters are also covered in more detail in the text on **Java methods**.

JDK VS JRE VS JVM

JVM is Java Virtual Machine. It's the program that runs your compiled Java code - loading code into memory, executing routines, and so on. It's a "virtual machine" because it's like a computer inside your computer - your CPU doesn't run the code directly, instead the virtual machine is the one that runs it.

JRE is Java Runtime Environment. It's a set of programs and libraries used to run Java code. This includes the JVM as well as the Java Standard Libraries (for example where classes such as String and ArrayList are written).

JDK is Java Development Kit. It includes the JRE as well as programs for actually creating Java programs, such as the Java compiler.

Type casting

Type casting is when you assign a value of one primitive data type to another type.
In Java, there are two types of casting:

Widening Casting (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double

Narrowing Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char -> short -> byte

Loops

FOR Loop

The Java for loop is a control flow statement that iterates a part of the programs multiple times.

WHEN TO USE FOR LOOP

If the number of iteration is fixed, it is recommended to use for loop.

SYNTAX:

```
for(init;condition;incr/decr){

// code to be executed

}
```

WHILE

The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.

WHEN TO USE WHILE LOOP

If the number of iteration is not fixed, it is recommended to use while loop.

SYNTAX

```
while(condition){

//code to be executed

}
```

DO-WHILE

The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.

WHEN TO USE DO WHILE

If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

SYNTAX

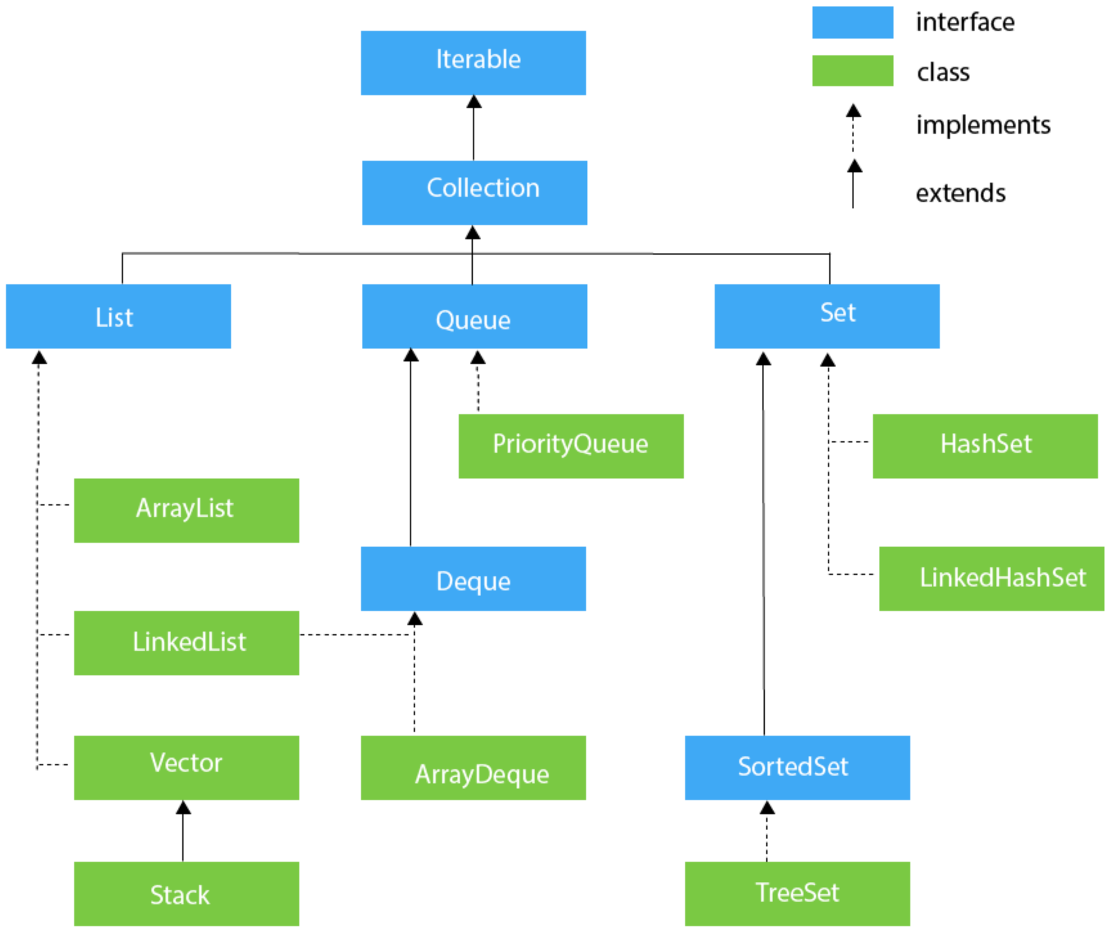
```
do{

//code to be executed

}while(condition);
```

Java Collections

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the classes and interfaces for the Collection framework.



List Interface declaration

public interface List<E> extends Collection<E>

Creating a List of type String using ArrayList

```
List<String> list=new ArrayList<String>();
```

Creating a List of type Integer using ArrayList

```
List<Integer> list=new ArrayList<Integer>();
```

Creating a List of type Book using ArrayList

```
List<Book> list=new ArrayList<Book>();
```

Creating a List of type String using LinkedList

```
List<String> list=new LinkedList<String>();
```

Java ArrayList

Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here. The ArrayList maintains the insertion order internally.

It inherits the AbstractList class and implements List interface.

ArrayList class declaration

public class ArrayList<E> extends AbstractList<E> implements List<E>

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Java Non-generic Vs. Generic Collection

non-generic example of creating java collection.

```
ArrayList list=new ArrayList();//creating old non-generic arraylist
```

generic example of creating java collection.

```
ArrayList<String> list=new ArrayList<String>();//creating new generic arraylist
```

Java LinkedList class

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

Java LinkedList class can contain duplicate elements.

Java LinkedList class maintains insertion order.

Java LinkedList class is non synchronized.

In Java LinkedList class, manipulation is fast because no shifting needs to occur.

Java LinkedList class can be used as a list, stack or queue.

Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList.
However, It is synchronized and contains many methods that are not the part of Collection framework.

Example: “ Vector<String> v=new Vector<String>();

```
                v.add(“1");
                v.add(“2");
            Iterator<String> itr=v.iterator();
                  while(itr.hasNext())
                {
                  System.out.println(itr.next());
                 } “
```

Stack

The stack is the subclass of Vector.
It implements the last-in-first-out data structure, i.e., Stack.
The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

Example:

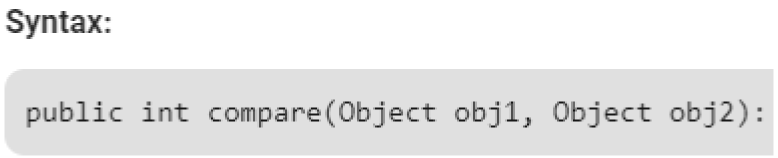
```
“Stack<String> stack = new Stack<String>();
 stack.push(“first element");
 stack.push(“second element");
 stack.pop();
 Iterator<String> itr=stack.iterator();
  while(itr.hasNext()){
 System.out.println(itr.next()); } “
```

HashSet

- HashSet class implements Set Interface.
- It represents the collection that uses a hash table for storage.
- Hashing is used to store the elements in the HashSet. It contains unique items.

Comparator

Comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of two different classes. Following function compare obj1 with obj2.



Syntax:

```
public int compare(Object obj1, Object obj2):
```

How does Collections.Sort() work?

Internally the Sort method does call Compare method of the classes it is sorting. To compare two elements, it asks “Which is greater?” Compare method returns -1, 0 or 1 to say if it is less than, equal, or greater to the other. It uses this result to then determine if they should be swapped for its sort.

You can implement the Comparator method on a class in order to enable alternative sorting or searching methods through the java library.

For instance, if you wanted to sort an ArrayList of type String based on String length, then you could write a class called StringLengthComparator that implements Comparator with the desired comparison.

StringLengthComparator slc = new StringLengthComparator();

int result = slc.compare(a, b);

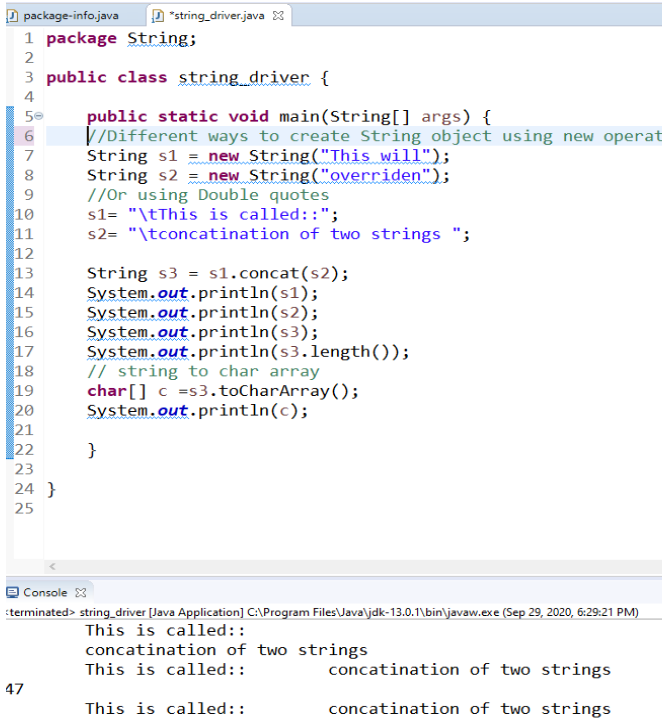
If a is shorter than b, then the method would return a number less than zero. If it is longer than b, then it would return a number greater than zero. If they are the same length, then it would return zero.
You could then sort the ArrayList using Collections.sort.

Collections.sort(myArrayList, slc);

String in Java

Strings in Java are Objects that are backed internally by a char array. Since arrays are immutable(cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

```
<String_Type> <string_variable> = "<sequence_of_string>";
```



package-info.java | *string_driver.java

```java
package String;

public class string_driver {

    public static void main(String[] args) {
    //Different ways to create String object using new operat
    String s1 = new String("This will");
    String s2 = new String("overriden");
    //Or using Double quotes
    s1= "\tThis is called::";
    s2= "\tconcatination of two strings ";

    String s3 = s1.concat(s2);
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s3.length());
    // string to char array
    char[] c =s3.toCharArray();
    System.out.println(c);

    }

}
```

Console

```
<terminated> string_driver [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Sep 29, 2020, 6:29:21 PM)
        This is called::
        concatination of two strings
        This is called::        concatination of two strings
47
        This is called::        concatination of two strings
```

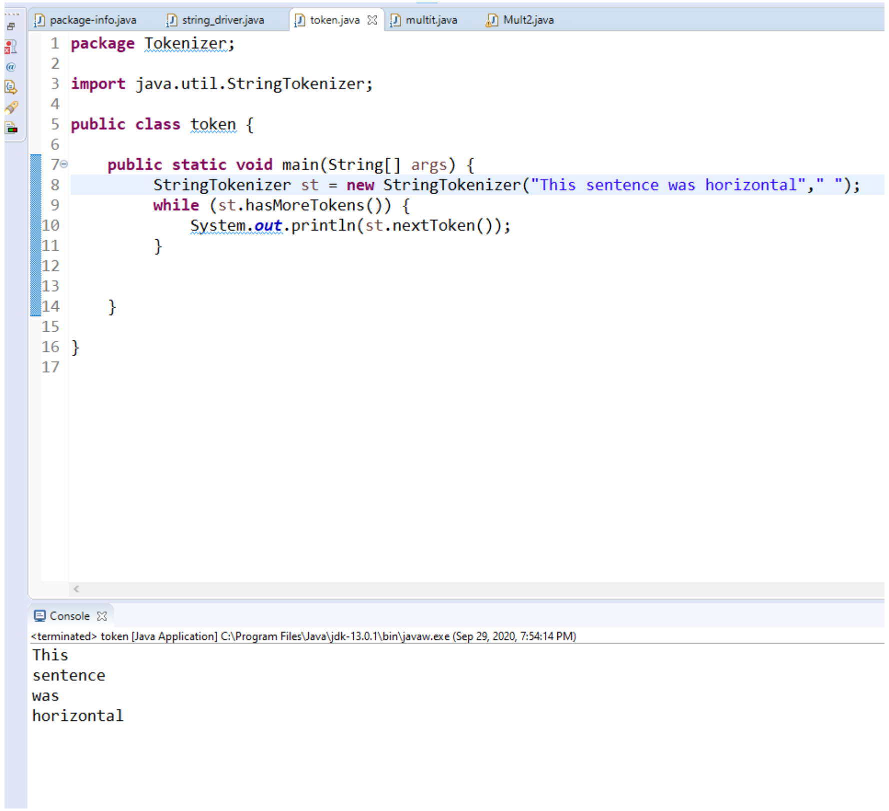
Tokenizer

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the StreamTokenizer class. The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments.
The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the returnDelims flag having the value true or false:

If the flag is false, delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.
If the flag is true, delimiter characters are themselves considered to be tokens. A token is thus either one delimiter character, or a maximal sequence of consecutive characters that are not delimiters.
A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed.

A token is returned by taking a substring of the string that was used to create the StringTokenizer object.



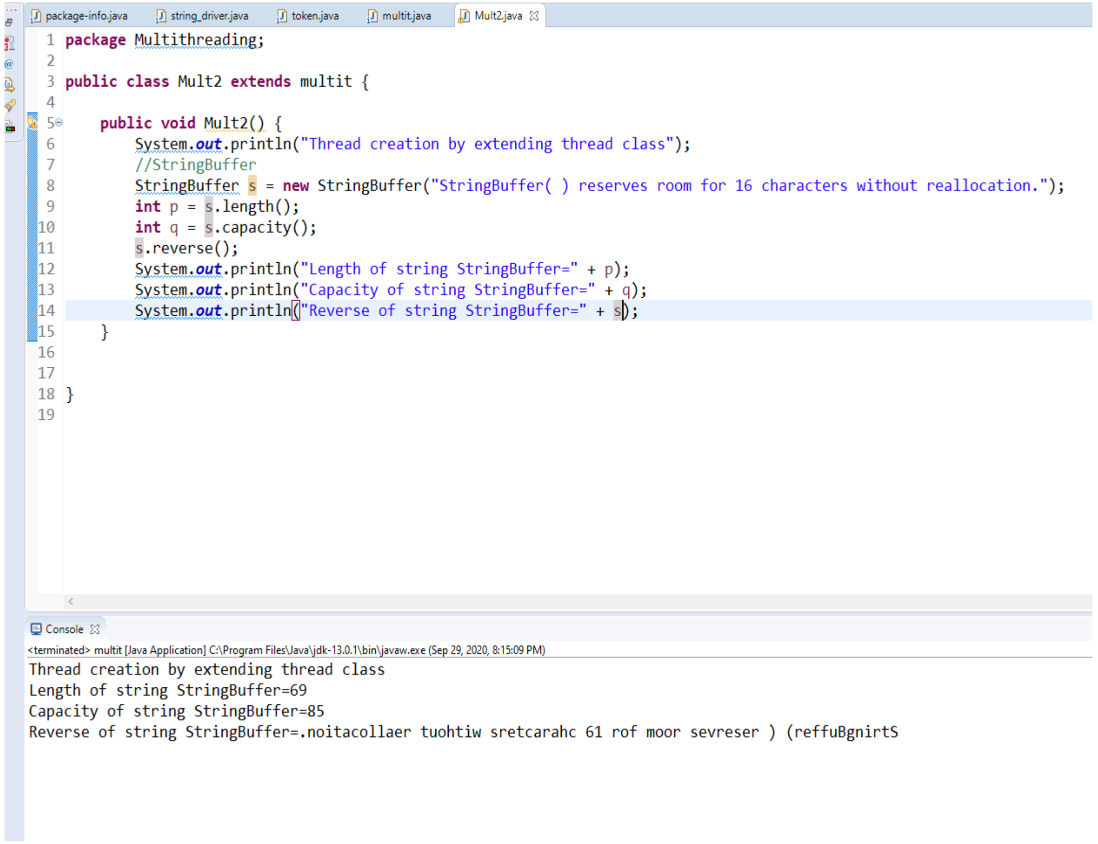
```java
package Tokenizer;

import java.util.StringTokenizer;

public class token {

    public static void main(String[] args) {
            StringTokenizer st = new StringTokenizer("This sentence was horizontal"," ");
            while (st.hasMoreTokens()) {
                System.out.println(st.nextToken());
            }


    }

}

```

```
<terminated> token [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Sep 29, 2020, 7:54:14 PM)
This
sentence
was
horizontal
```

StringBuffer

StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

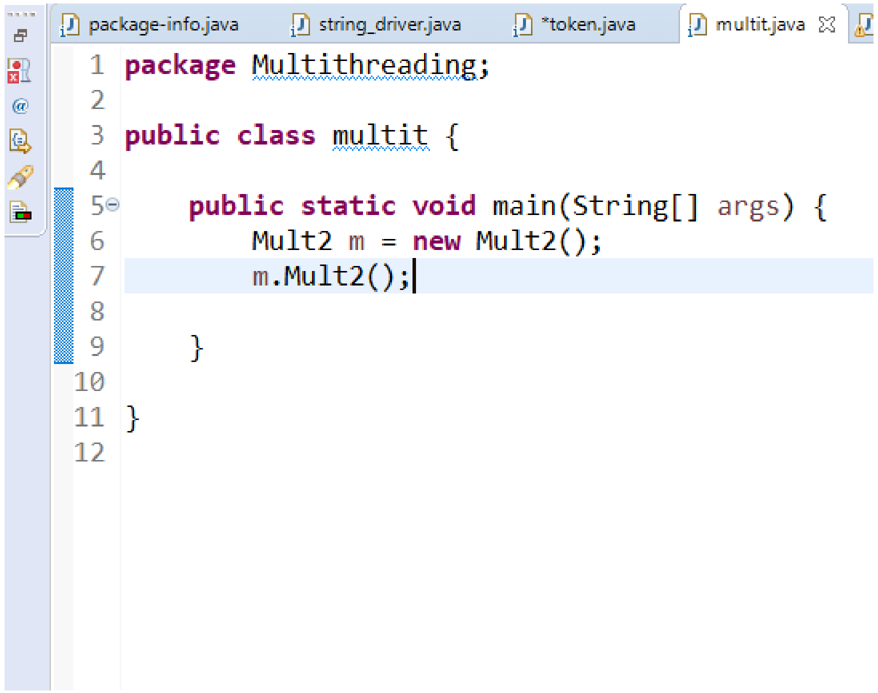
StringBuffer may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.



```java
package Multithreading;

public class Mult2 extends multit {

    public void Mult2() {
        System.out.println("Thread creation by extending thread class");
        //StringBuffer
        StringBuffer s = new StringBuffer("StringBuffer( ) reserves room for 16 characters without reallocation.");
        int p = s.length();
        int q = s.capacity();
        s.reverse();
        System.out.println("Length of string StringBuffer=" + p);
        System.out.println("Capacity of string StringBuffer=" + q);
        System.out.println("Reverse of string StringBuffer=" + s);
    }


}

```

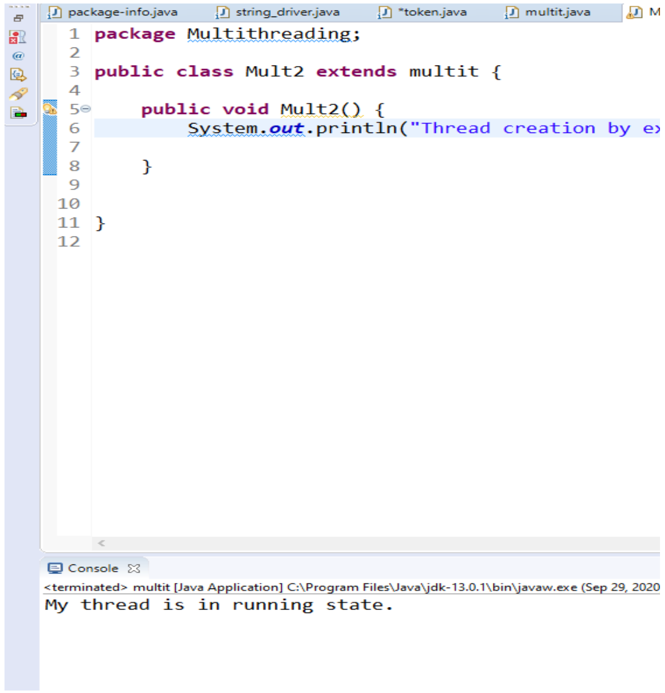
```
<terminated> multit [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Sep 29, 2020, 8:15:09 PM)
Thread creation by extending thread class
Length of string StringBuffer=69
Capacity of string StringBuffer=85
Reverse of string StringBuffer=.noitacollaer tuohtiw sretcarahc 61 rof moor sevreser ) (reffuBgnirtS
```

Multithreading



package-info.java string_driver.java *token.java multit.java

```java
package Multithreading;

public class multit {

    public static void main(String[] args) {
        Mult2 m = new Mult2();
        m.Mult2();

    }

}

```



package-info.java string_driver.java *token.java multit.java M

```java
package Multithreading;

public class Mult2 extends multit {

    public void Mult2() {
        System.out.println("Thread creation by ex

    }


}

```

Console

```
<terminated> multit [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Sep 29, 2020
My thread is in running state.
```