**ENUM (Enumeration) IN JAVA:**

* The Enum in java is a datatype which contains a fixed set of constants.
* Naming conventions: All Enum constants in capital letter.
* Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change).
* Java enum constants are static and final implicitly.
* Enums are used to create our own data type like classes.
* The **enum** data type (also known as Enumerated Data Type) is used to define an enum in Java.

**NOTE:**

* We can define an enum either inside the class or outside the class.
* We can have fields, constructors, methods, and main methods in Java enum.
* Java Enum internally inherits the *Enum class*, so it cannot inherit any other class, but it can implement many interfaces.
* The semicolon (;) at the end of the enum constants are optional.

**Declaring Enums:**

**Syntax:**

enum enum-name{constant 1, constant 2, ….,constant n};

**Sample Format:**

enum Program{HTML, CSS, BOOTSTRAP, JS, JAVA, MYSQL};

**Example:**

class ProgrammingLanguage{

enum Program{ HTML(10), CSS(9), BOOTSTRAP(12), JS(15), JAVA(18), MYSQL(11) };

private int price;

Program(int p)

{

price = p;

}

int getPrice()

{

return price;

}

}

**Accessing Enums:**

**Syntax:**

enum-name.enum-constant

or

enum-reference.enum-constant

enum-name🡪name of the enum.

enum-constant🡪enum constant.

enum-reference🡪enum reference variable.

**Sample Format:**

Program p = Program.JAVA;

* Enum constant, JAVA, is accessed using the enum name, Program.

**values()**

* To access all the values stored in an enum, java provides the ***value()*** method.
* This method returns all the enum constants stored in an enum.

**Example:**

enum Program{ HTML(10), CSS(9), BOOTSTRAP(12), JS(15), JAVA(18), MYSQL(11) };

class EnumTest{

public static void main(String[] args){

for(Program p : Program.values()){

System.out.println(p);

}

}

}

* The ***for-each*** loop is used to traverse through the enum, **Program.**
* The values() method is used to return the enum constant stored in enum.

**To access the method defined in enum**

**Example:**

class ProgrammingLanguage{

enum Program{ HTML(10), CSS(9), BOOTSTRAP(12), JS(15), JAVA(18), MYSQL(11) };

private int price;

Program(int p)

{

price = p;

}

int getPrice()

{

return price;

}

}

class EnumTest{

public static void main(String[] args){

ProgrammingLanguage.Program p = ProgrammingLanguage.JAVA;

System.out.println(p.getPrice());

}

}

* When the statement, *Program p = Program.JAVA*, is executed, the enum constructor is invoked and the value, *18* is assigned to the member variable, *price*.
* Therefore, when the *getPrice()* method is called, the value, *18*, is returned.

**value(), valueOf(), ordinal()**

**Example:**

class EnumExample1{

enum Bike{TVS, HONDA, YAMAHA, HERO};

public static void main(String[] args){

for(Bike b : Bike.values()){

System.out.println(b);

}

System.out.println(“Value of TVS is: ”+ Bike.valueOf(“TVS”));

System.out.println(“Index of TVS is: ”+ Bike.valueOf(“TVS”).ordinal());

System.out.println(“Index of HONDA: ”+Bike.valueOf(“HONDA”).ordinal());

}

}

**Note:**

* Java compiler internally adds value(), valueOf(), and ordinal() methods within the enum at the compile time.
* It internally creates a static and final class for the enum.

**value() method**

* The value() method returns an array containing all the values of enum.

**valueOf() method**

* The valueOf() method returns the value of given constant enum.

**ordinal() method**

* The ordinal() method returns the index of the enum value.

**Example:**

class EnumExample2{

enum Phone{SAMSUNG, REDMI, NOKIA}

public static void main(String[] args){

Phone p = Phone.SAMSUNG;

System.out.println(p);

}

}

**JAVA STRING**

* String is a sequence of characters, but in java, string is an object that represents a sequence of characters.
* Package used to create string: java.lang

**Example**

char[] c = {‘J’, ‘A’, ‘V’, ‘A’};

String s = new String(ch);

Same as:

String s = “JAVA”;

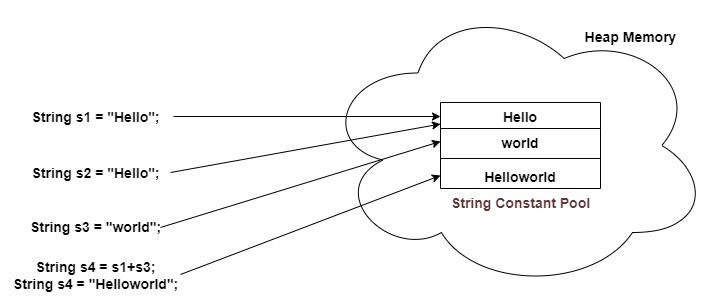
**How to create String Object**

There are 2 ways to create string object.

1. By string literal
2. By new keyword

**By String literal:**

* Java String literal is created by using double quotes.
* To manipulate string, we can use classes such as ***String***, ***StringBuilder***, ***StringBuffer***, provided by java.
* In Java, String class is an immutable class. This means that once a string object is created, we cannot change its value.
* However, the reference variable of the *String* class are mutable.



**Example:**

public class StringLiteralEx{

public static void main(String[] args){

String s1 = “Hello”;

String s2 = “Hello”;

System.out.println(s1 == s2); // output: true

}

}

**Example:**

public class StringLiteralEx1{

public static void main(String[] args){

String s1 = “Hello”;

String s2 = “Hello”;

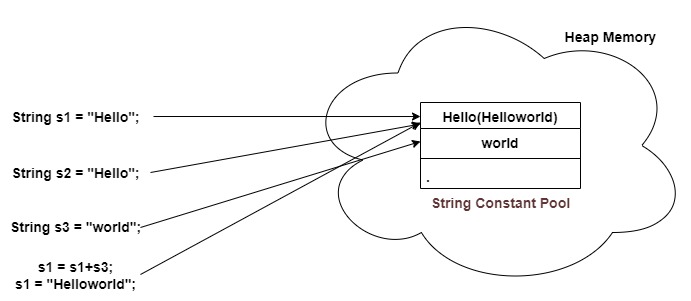
String s3 = “world”;

String s4 = s1 + s3;

System.out.println(s4); // output: Helloworld

}

}



* Each time you create a string literal, the JVM checks the "string constant pool" first.
* If the string already exists in the pool, a reference to the pooled instance is returned.
* If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

**Note:**

* In JVM, a special memory named string constant pool is used to store string literals.

**Example:**

public class StringLiteralEx2{

public static void main(String[] args){

String s1 = “Hello”;

String s2 = “Hello”;

String s3 = “world”;

s1 = s1 + s3;

System.out.println(s1); // output: Helloworld

}

}

**Example:**

public class StringLiteralEx3{

public static void main(String[] args){

String s1 = “Hello”;

String s2 = “Hello”;

String s3 = “world”;

s1 = s1.concat(s3);

System.out.println(s1); // output: Helloworld

}

}

**Why java uses the concept of string literal?**

**Solution:**

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

**By new keyword:**

* JVM will create a new string object in normal (non-pool) heap memory and the literal string value will be placed in the string constant pool.
* The variable will refer to the object in a heap(non-pool).

A diagram of a stringing system

Description automatically generated

A diagram of a stringing system

Description automatically generated

**Example:**

public class StringObjectEx{

public static void main(String[] args){

String s1 = new String(“Hello”);

String s2 = new String(“Hello”);

System.out.println(s1 == s2); // output: false

}

}

**Example:**

public class StringObjectEx2{

public static void main(String[] args){

String s1 = new String(“Hello”);

String s2 = “Hello”;

String s3 = “world”;

s1 = s1 + s3;

System.out.println(s1); // output: Helloworld

}

}

**Example:**

public class StringObjEx3{

public static void main(String[] args) {

String s1 = new String("Hello");

String s2 = new String("Hello");

String s3 = s1 + s2;

System.out.println(s3);

String s5 = s1 + s4;

System.out.println(s5);

s5 = s5+("program");

System.out.println(s5);

}

}