UNIT-II
STRINGS

Strings

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

1. char[] ch={'j','a','v','a'};
2. String s=new String(ch);

is same as:

1. String s="java";

String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

What is String in Java?

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

String is Immutable,means we cant add or delete to string without creating another string.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. For Example:

String s="welcome";

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. For example:

String s1="Welcome";

1. String s2="Welcome";//It doesn't create a new instance

EXAMPLE:

Using String Literal

String str1 = "Python";

String str2 = "Data Science";

String str3 = "Python";

Using new Keyword

In Java, a new keyword is also used to create String, as follows:

String str1 = new String ("Java");

String str2 = new String ("C++");

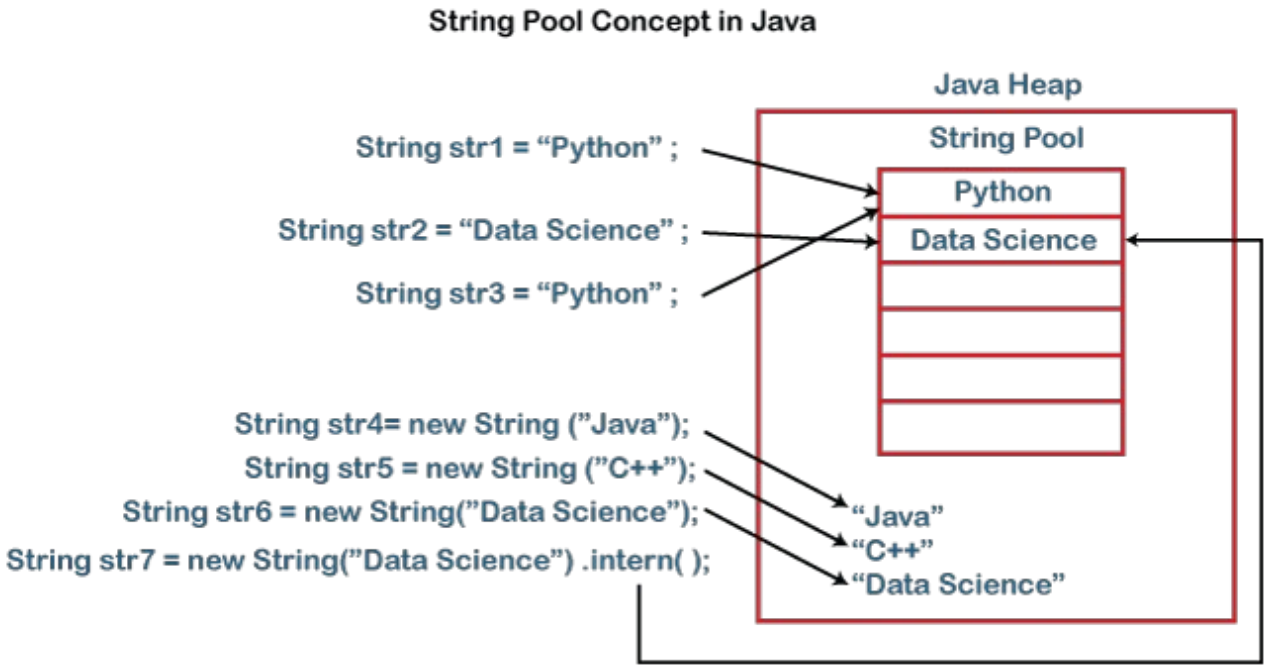
String str3 = new String ("Data Science");

Let's understand what is the difference between them. Let's compare the string literals' references.

s1==s3 //true

s2==s3 //false

Let's see how we found that equal or not.



Java String Example

StringExample.java

```
public class StringExample {
   public static void main(String args[]) {
      String s1 = "java"; // creating string by Java string literal
      char ch[] = {'s', 't', 'r', 'i', 'n', 'g', 's'};
      String s2 = new String(ch); // converting char array to string
     String s3 = new String("example"); // creating Java string by newkeyword
      System.out.println(s1); // Output: java
      System.out.println(s2); // Output: strings
      System.out.println(s3); // Output: example
   }
```

```
}
```

All String Methods

The String class has a set of built-in methods that you can use on strings.

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
getChars()	Copies characters from a string to an array of chars	void
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
isEmpty()	Checks whether a string is empty or not	boolean
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int
length()	Returns the length of a specified string	int
matches()	Searches a string for a match against a regular expression, and returns the matches	boolean
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String
replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String
split()	Splits a string into an array of substrings	String[]
startsWith()	Checks whether a string starts with specified characters	boolean
substring()	Returns a new string which is the substring of a specified string	String
toCharArray()	Converts this string to a new character array	char[]
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String

valueOf()	Returns the string representation of the specified value	String

Java String Operations

1. Get the Length of a String

To find the length of a string, we use the length() method. For example,

```
// create a string
    String s1 = "Hello! World";

// get the length of greet
    int length = s1.length();

    System.out.println("Length: " + length);
```

Output

String: Hello! World

Length: 12

In the above example, the length() method calculates the total number of characters in the string and returns it.

2. Join Two Java Strings

We can join two strings in Java using the concat() method. For example,

```
// create first string

    String first = "Java ";

// create second

    String second = "Programming";

// join two strings

    String joinedString = first.concat(second);

    System.out.println("Joined String: " + joinedString);
```

output:

Joined String: Java Programming

We can also join two strings using the + operator in Java.

3. Compare Two Strings

we can make comparisons between two strings using the equals() method. For example,

```
// create 3 strings

    String first = "java programming";
    String second = "java programming";
    String third = "python programming";

    // compare first and second strings

    boolean result1 = first.equals(second);

    System.out.println("Strings first and second are equal: " + result1);

    // compare first and third strings
    boolean result2 = first.equals(third);

    System.out.println("Strings first and third are equal: " + result2);
```

Output

```
Strings first and second are equal: true
Strings first and third are equal: false
```

The equals() method checks the content of strings while comparing them.

We can also compare two strings using the == operator in Java. However, this approach is different than the equals() method.

Creating Strings Using the New Keyword

So far, we have created strings like primitive types in Java.

Since strings in Java are objects, we can create strings using the new keyword as well. For example,

```
// create a string using the new keyword
String name = new String("Java String");

System.out.println(name);  // print Java String
```

Java StringBuffer Class

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

methods of StringBuffer class

Method	Description
append(String s)	It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
insert(int offset, String s)	It is used to insert the specified string with this string at the specified position. The insert() method is

	overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
reverse()	is used to reverse the string.
capacity()	It is used to return the current capacity.
ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
charAt(int index)	It is used to return the character at the specified position.
length()	It is used to return the length of the string i.e. total number of characters.
substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

Example:

```
class StringBufferExample{

    public static void main(String args[])
        {
        StringBuffer sb=new StringBuffer("Hello");
        sb.append("Java");//now original string is changed
        System.out.println(sb);//prints Hello Java
        }
}
```

Java OOPs Concepts

In this page, we will learn about the basics of OOPs. Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

Simula is considered the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.

Smalltalk is considered the first truly object-oriented programming language.

The popular object-oriented languages are java, C#, PHP, Python, C++, etc.

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object
Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.
Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Collection of objects is called class. It is a logical entity.
A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Objects and Classes in Java

In object-oriented programming technique, we design a program using objects and classes.

Real World Enties:

Real World Enties like Car, Dog, Person, Students has two common characteristics

- **State**
- **Behavior**
  For Example:
  Dog has
- **State : name, color, breed, hungry**
- **Behavior: barking, playing, wagging tail**

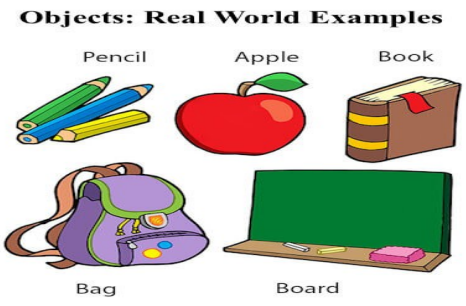
In OOP We can represent realworld Entities as **objects**

What is an object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has two characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.



For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

In oop we can think of each of these real world entities as special type of “thing” in our code called as object.

These software object has similar qualities and as the real world objects they represent.

Each object has its own Information(data) and can perform action(behaviour) related to it just like real world objects.

This is the basic idea of oop representing real world objects in code.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

An object is an entity that will have some well defined behavior.

The structure and behavior of common objects are defined in class.

Software objects are instance of userdefined or Abstarct data type(class).

It is concrete entity based on blueprint defined by class.

When we create an obect of class we are creating an instance of that class with its own unique data.

From one class we define many objects.

Object can not exist without a class.

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

To represent numbers we have int,float,double but how to represent real world objects like car,dog,person,etc.

We can use usedefined data types like stuctures but can store only informantion,no associated behavior.

Class is a Userdefined data type or Astact data type,behaves like real world object.

Which define the stucture of similar objects.

A class in Java can contain:

- Fields
- Methods

Syntax to declare a class:

```
Class <class_name>{
        fields;//(variables||data||state||properties)
        method;//(actions||behavior)
}
```

here class is a keyword,

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

The new keyword is followed by the constructor of a class, which sets the initial values for the object's fields and performs any other necessary setup. The constructor is a special method that is called automatically when an object is created. Without the new keyword, an object cannot be created and the constructor cannot be called.

Syntax of creating object

ClassName objName = new **ClassName( );**
to create preson1 object to preson class we can write as:
person person1=new person( );
Here,'new' is an operator that creates the object to person class,hence the right hand side part of the statement is responsible for creating an object.The left hand side of statement which is:
person person1;
here,person is the class name and it is also a data type and person1 is the object name. Person1 is a variable of person class.
This variable stores the reference number of the object returned by JVM,afte creating the object.
So we can say person1is variable of person class type.

Clacc code along with method is stored in Method area of JVM,when an object is created, the memory is allocated on heap,after creation of object.JVM produces a unique reference number for object.

Referencing Object Field:

1. Within same class

- we can access instance variables in same class with simple name of variable

2. Out side of the Class:

Syntax:

reference .(dot) fieldname

Example :

person1.name

Object and Class Example

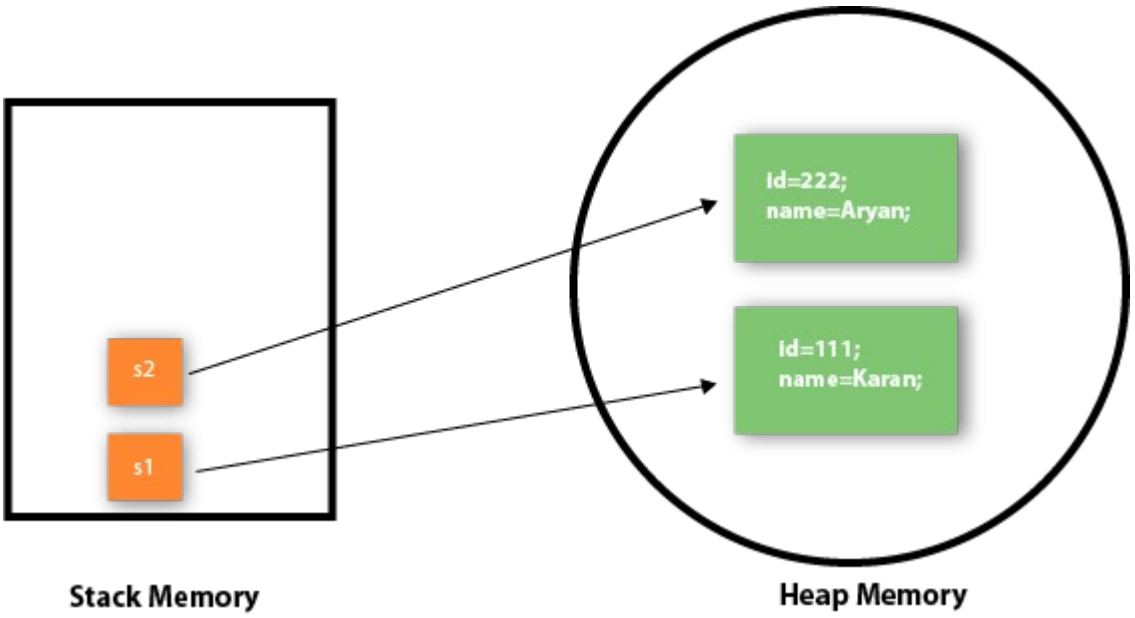
In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

```
//Creating Student class.
 class Student {
String name;
int id;
void displayDetails() {
System.out.println("name:"+name + "\n"+"id is:"+id);
}
}
public class StudentDemo {
public static void main(String args[])
{
Student s1=new Student();
Student s2=new Student();
s1.name="Aryan";  //accessing variables from outside of class
s1.id=111;
s2.name="karan";
s2.id=222;
s1.displayDetails();
s2.displayDetails();
}
}
```

output:

```
name:john
age is:24
name:suraj
age is:30
```



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.