Week 6

1. **STATIC VARIABLES AND METHODS**

Whenever a variable is declared as static, this means there is only one copy of it for the entire class, rather than each instance having its own copy. A static method means it can be called without creating an instance of the class. Static variables and methods in Java provide several advantages, including memory efficiency, global access, object independence, performance, and code organization.

Understanding static in Java is pivotal for grasping its role in creating class-level variables and methods. Dive into static concepts and more with a Java Course for comprehensive learning.

In Java, it is possible to use the static keyword with methods, blocks, variables, as well as nested classes. In simple words, if you use a static keyword with a variable or a method inside a class, then for every instance that you create for that class, these static members remain constant and you can’t change or modify them. In fact, you can access these members even without creating an instance of an object for those classes. You can access them simply using the class name. In fact, the main method of a class in Java usually has a static keyword associated with it. But, yes, it depends on the choice of the developer.

There are tons of reserved keywords in Java that cannot be used as names of variables or identifiers. One such frequently used keyword in Java is the “Static” keyword. The most important reason why static keywords are heavily used in Java is to efficiently manage memory. Generally, if you want to access variables or methods inside a class, you first need to create an instance or object of that class. However, there might be situations where you want to access only a couple of methods or variables of a class and you don’t want to create a new instance for that class just for accessing these members. This is where you can use the static keyword in Java.

In Java, it is possible to use the static keyword with methods, blocks, variables, as well as nested classes. In simple words, if you use a static keyword with a variable or a method inside a class, then for every instance that you create for that class, these static members remain constant and you can’t change or modify them. In fact, you can access these members even without creating an instance of an object for those classes. You can access them simply using the class name. In fact, the main method of a class in Java usually has a static keyword associated with it. But, yes, it depends on the choice of the developer.

**Static Keyword in Java**

Static keyword in java in Java indicates that a particular member is not an instance, but rather part of a type. The static member will be shared among all instances of the class, so we will only create one instance of it.

If any member in a class is declared as static, it means that even before the class is initiated, all the static members can be accessed and become active. In contrast to this, non-static members of the same class will cease to exist when there is no object or the object goes out of scope.

On a side note, if you consider the methods inside the “Math” class in Java, you will find that most of its methods are static. You can simply access them using the class name. For example, “Math.abs()”, “Math.pow()”, “Math.PI”, etc.

There are tons of reserved keywords in Java that cannot be used as names of variables or identifiers. One such frequently used keyword in Java is the “Static” keyword. The most important reason why static keywords are heavily used in Java is to efficiently manage memory. Generally, if you want to access variables or methods inside a class, you first need to create an instance or object of that class. However, there might be situations where you want to access only a couple of methods or variables of a class and you don’t want to create a new instance for that class just for accessing these members. This is where you can use the static keyword in Java.

**Static Keyword in Java**

Static keyword in java in Java indicates that a particular member is not an instance, but rather part of a type. The static member will be shared among all instances of the class, so we will only create one instance of it.

If any member in a class is declared as static, it means that even before the class is initiated, all the static members can be accessed and become active. In contrast to this, non-static members of the same class will cease to exist when there is no object or the object goes out of scope.

**Static Variables in Java**

When you create an object or instance for a class in Java, each object will have its own copy of the members such as variables and methods.

|  |
| --- |
| class Person{  int age;  }  class Main{  public static void main(String args[]){  Person p1 = new Person();  Person p2 = new Person();  p1.age = 31;  p2.age = 32;  System.out.println("P1\'s age is: " + p1.age);  System.out.println("P2\'s age is: " + p2.age);  }  } |

In the above example, both the person objects p1 and p2 have their own local copy of the non-static variable age. If you change them, they will store different values then.

However, if the same variable age would have been declared as a static variable, then all the objects declared for this class would share the same copy of the static variable. This is so because of static variables or for that matter, all the static members are not associated with instances, but with classes. Hence, you won’t even have to create an object to access static members. Consider the same example but with a static variable called age.

|  |
| --- |
| class Person{  static int age;  }  class Main{  public static void main(String args[]){  Person p1 = new Person();  Person p2 = new Person();  p1.age = 30;  p2.age = 31;  Person.age = 32;  System.out.println("P1\'s age is: " + p1.age);  System.out.println("P2\'s age is: " + p2.age);  }  } |

**Static Methods in Java**

It is common to often refer to static methods in Java as class methods. The reason being that the static members are associated with the classes and with their objects. Similar to static variables, static methods can also be invoked using the class name. There are some important points that you need to consider when you work with static methods in Java. These are -

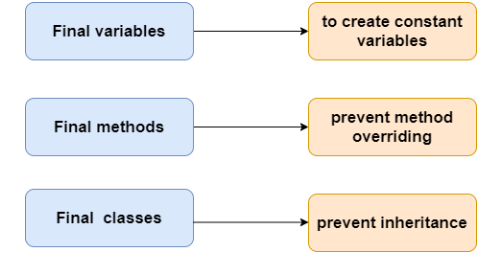
* The static methods of a particular class can only access the static variables and can change them.
* A static method can only call other static methods.
* Static methods can’t refer to non-static variables or methods.
* Static methods can’t refer to “super” or “this” members.

Also, often you will notice that the main method in Java is defined as static. This is so because you don’t need an object to call the main method in Java. If you have defined the main method in Java as non-static, then the Java Virtual Machine (JVM) would have first created an instance for the main class and then called the main method using that instance which would lead to unnecessary memory usage. Moreover, there are tons of static methods defined in the Wrapper Classes and Utility Classes in Java.

|  |
| --- |
| // Java program to demonstrate that  // The static method does not have  // access to the instance variable    import java.io.\*;    public class GFG {  // static variable  static int a = 40;    // instance variable  int b = 50;    void simpleDisplay()  {  System.out.println(a);  System.out.println(b);  }    // Declaration of a static method.  static void staticDisplay()  {  System.out.println(a);  }    // main method  public static void main(String[] args)  {  GFG obj = new GFG();  obj.simpleDisplay();    // Calling static method.  staticDisplay();  }  }  **Output:**  40  50  40 |

1. **FINAL VARIABLE AND METHODS**

All methods and variables can be overridden by default in subclasses. If we wish to prevent the subclasses from overriding the members of the superclass, we can declare them as final using the keyword final as a modifier. The final keyword in java is used to restrict the user.



**Fig 1: Final keyword**

**Final variables:**

If we use variable as final, we cannot change the value of final variable hence it becomes the constant . It is good practice to represent final variables in all uppercase, using underscore to separate words.

|  |
| --- |
| class Student{  final int rollno=18;//final variable  void run(){  rollno=28;  }  public static void main(String args[]){  Student s = new Student();  s.run();  }  }//end of class  **Output:**  error: cannot assign a value to final variable rollno  rollno=28; |

**b) Final methods:**

If we use method with final keyword, it prevents method overriding. When a method is declared with final keyword, it is called a final method.

|  |
| --- |
| class Father{  final void work(){System.out.println("working in office");} // final method  }  class Son extends Father{  void work(){System.out.println("working sums");}  public static void main(String args[]){  Son s = new Son();  s.run();  }  }  **Output:**  error: work() in Son cannot override work() in Father  void work(){System.out.println("working sums") |

**c) Final classes:**

If we use class with final keyword, it prevents inheritance. When a class is declared with final keyword, it is called a final class.

|  |
| --- |
| final class Father{} // final class  class Son extends Father{  void play(){System.out.println("playing...");}  public static void main(String args[]){  Son s = new Son();  s.play();  }  }  **Output:**  error: cannot inherit from final Father  class Son extends Fathe |

**Points to remember about Final Keyword in Java:**

* Once any data member (a variable, method, or class) gets declared as final, it can only be assigned once.
* The final variable cannot be reinitialized with another value.
* A final method cannot be overridden by another method.
* A final class cannot be extended or inherited by another child class.

1. **NESTED CLASS**

Nested classes enable us to logically group classes that are only used in one place, write more readable and maintainable code and increase encapsulation. Types of nested classes available in the language:

* Static nested classes
* Non-static nested classes
* Local classes
* Anonymous classes

**Static Nested Classes**

* As with static members, these belong to their enclosing class, and not to an instance of the class
* They can have all types of access modifiers in their declaration
* They only have access to static members in the enclosing class
* They can define both static and non-static members

|  |
| --- |
| public class Enclosing {    private static int x = 1;    public static class StaticNested {  private void run() {  // method implementation  }  }    @Test  public void test() {  Enclosing.StaticNested nested = new Enclosing.StaticNested();  nested.run();  }  } |

**b) Non-Static Nested Classes**

* They are also called inner classes
* They can have all types of access modifiers in their declaration
* Just like instance variables and methods, inner classes are associated with an instance of the enclosing class
* They have access to all members of the enclosing class, regardless of whether they are static or non-static
* They can only define non-static members

**Local Classes**

* Local classes are a special type of inner classes – in which the class is defined inside a method or scope block.
* They cannot have access modifiers in their declaration
* They have access to both static and non-static members in the enclosing context
* They can only define instance members

|  |
| --- |
| public class NewEnclosing {    void run() {  class Local {  void run() {  // method implementation  }  }  Local local = new Local();  local.run();  }    @Test  public void test() {  NewEnclosing newEnclosing = new NewEnclosing();  newEnclosing.run();  }  } |

**Anonymous Classes**

Anonymous classes can be used to define an implementation of an interface or an abstract class without having to create a reusable implementation.

* They cannot have access modifiers in their declaration
* They have access to both static and non-static members in the enclosing context
* They can only define instance members
* They’re the only type of nested classes that cannot define constructors or extend/implement other classes or interfaces

|  |
| --- |
| public class AnonymousInnerUnitTest {    @Test  public void whenRunAnonymousClass\_thenCorrect() {  SimpleAbstractClass simpleAbstractClass = new SimpleAbstractClass() {  void run() {  // method implementation  }  };  simpleAbstractClass.run();  }  } |

**Shadowing**

The declaration of the members of an inner class shadow those of the enclosing class if they have the same name. In this case, the this keyword refers to the instances of the nested class and the members of the outer class can be referred to using the name of the outer class.

|  |
| --- |
| public class NewOuter {  int a = 1;  static int b = 2;  public class InnerClass {  int a = 3;  static final int b = 4;  public void run() {  System.out.println("a = " + a);  System.out.println("b = " + b);  System.out.println("NewOuterTest.this.a = " + NewOuter.this.a);  System.out.println("NewOuterTest.b = " + NewOuter.b);  System.out.println("NewOuterTest.this.b = " + NewOuter.this.b);  }  }  @Test  public void test() {  NewOuter outer = new NewOuter();  NewOuter.InnerClass inner = outer.new InnerClass();  inner.run();  }  } |

1. **STRING CLASS**

The use of strings is essential to programming. Java has a String class that allows for the creation and manipulation of strings. The character sequence can also be represented using an interface called CharSequence. One of the classes that implements this interface is the String class. Thus, in Java, a string is an object that is essentially a sequence of characters.

For instance, the word "hello" is made up of a string of the characters "h," "e," "l," "l," and "o." Additionally, we have a variety of String methods that make it simple to interact with String in Java.

**What is String Class in Java?**

String refers to a group of characters. String objects in Java are immutable, which simply means they can never be modified after they have been created. Java only supports operator overloading for the String class. The + operator allows us to combine two strings. For instance, "a"+"b"="ab." StringBuilder and StringBuffer are two helpful classes in Java that can be used to manipulate strings.

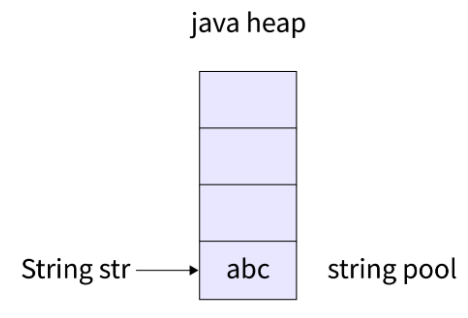
**How to Create a String Object in Java?**

A few of the more well-known techniques to generate a string object in Java are listed below.

**By String Literal**

The most typical method of producing string is this one. In this instance, double quotes are used to surround a string literal.

String str = "abc";



**Fig 2: String heap**

When double quotes are used to generate a String, JVM searches the String pool(A location inside the heap memory is called the String Constant Pool. It includes distinctive strings. The JVM determines whether a string is already present in the string pool before generating a new object in the pool. ) to see whether any other Strings with the same value are already stored there. If an existing String object is found, it simply returns the reference to it; otherwise, a new String object with the specified value is created and stored in the String pool.

**Using New Keyword**

Like any other Java class, we can construct String objects with the new operator. The String class has a number of constructors that can be used to create a String from a char array, byte array, StringBuffer, or StringBuilder.

String str = new String("abc");

char[] a = {'a', 'b', 'c'};

String str2 = new String(a);

**Constructors of String Class in Java**

1. String()

produces a blank string. Due to the immutability of String, it is largely worthless.

2. String(String original)

from another string, generates a string object. String is useless since it is immutable.

3. String(byte[] bytes)

creates a new string using the system's default encoding from the byte array.

4. String(byte bytes[], String charsetName)

utilises the character encoding supplied. UnsupportedEncodingException is thrown if the encoding is not supported.

5. String(byte[] byte\_arr, Charset char\_set)

In order to create a new String, decode the byte array. It decodes using the char set.

6. String(byte bytes[], int offset, int length)

The first byte to be decoded is indicated by the offset. The length defines how many bytes need to be decoded. If offset, length, or offset is more than bytes, this constructor raises an IndexOutOfBoundsException.

7. String(byte bytes[], int offset, int length, Charset charset)

Similar to the constructor above, the only difference is that the encoding to be used must be specified.

8. String(byte bytes[], int offset, int length, String charsetName)

The character set encoding name is supplied as a string, unlike the example before. If the encoding is not supported, this will throw an UnsupportedEncodingException.

9. String(char value[])

uses the character array to generate the string object.

10. String(char value[], int offset, int count)

uses the character array to generate the string object.

11. String(char value[], int offset, int count)

The initial character's index is specified by the offset. The amount of characters to use is determined on the length. If offset, length, or offset and value are negative, this constructor raises an IndexOutOfBoundsException.

12. String(int[] codePoints, int offset, int count)

Creates a string from an array of Unicode code points as input. If any of the code points are incorrect, it raises the IllegalArgumentException exception. If the offset length, or offset is more than codePoints, this constructor raises an IndexOutOfBoundsException.

13. String(StringBuffer buffer)

uses the data contained in the string buffer to produce a new string. Internally, this constructor uses the StringBuffer function toString()

14. String(StringBuilder buffer)

Takes the contents of the string builder and produces a new string.

**Methods of String Class in Java**

String Methods are highly helpful since they make working with Strings in Java very simple. The String methods in Java allow us to do many different things, including determining a string's length, concatenating several strings, determining whether two strings are equal, changing a string's characters' case, and much more. Let's talk about a few of Java's most well-liked and frequently utilised String methods. It should be noted that the Java class java.lang.String offers all of these string methods.

Java String split() -Splits the string at the specified string (regex)

Java String compareTo()-Compares two strings in the dictionary order

Java String compareToIgnoreCase()-Compares two strings ignoring case differences

Java String length()-Returns the length of the string

Java String replace()-Replace all matching characters/text in the string

Java String replaceAll()-Replace all substrings matching the regex pattern

Java String substring()-Returns a substring from the given string

Java String equals()-Compares two strings

Java String equalsIgnoreCase()-Compares two strings ignoring case differences

Java String contains()-Checks whether the string contains a substring

Java String indexOf()-Returns the index of the character/substring

Java String trim()-Removes any leading and trailing whitespace

Java String charAt()-Returns the character at the given index

Java String toLowerCase()-Converts characters in the string to lower case

Java String concat()-Concatenates two strings and returns it

Java String valueOf()-Returns the string representation of a value

Java String matches()-checks whether the string matches the given regex

Java String startsWith()-Checks if the string begins with the given string

Java String endsWith()-Checks if the string ends with the given string

Java String isEmpty()-Checks whether a string is empty or not

Java String intern()-Returns a canonical representation of the string

Java String getBytes()-Encodes the string into a sequences of bytes

Java String contentEquals()-Checks whether the string is equal to charSequence

Java String hashCode()-Returns a hash code for the string

Java String join()-Joins the given strings using the delimiter

Java String replaceFirst()-Replace the first matching substring

Java String subSequence()-Returns a subsequence from the string

Java String toCharArray()-Converts the string to a char array

Java String format()-Returns a formatted string

1. **COMMAND LINE ARGUMENTS**

The command line argument in java is the information passed to the program at the time of running the program. It is the argument passed through the console when the program is run. The command line argument is the data that is written right after the program’s name at the command line while executing the program. The arguments passed to the java program through this command line can be received by the program as an input and used within the code.

**How to access java command line arguments?**

The method of accessing the command line arguments in java is very straightforward. To use these arguments within our java code is simple. They are stored as an array of Strings passed to the main(). It is mostly named as args.

|  |
| --- |
| // Program to check for command line arguments  public class Example {  public static void main(String[] args) {  // check if the length of args array is < 0  if (args.length <= 0) {  System.out.println("No command line arguments found.");  } else {  System.out.println("The first command line argument is: " + args[0]);  System.out.println("All of the command line arguments are: ");  // iterating the args array and printing all of the command line arguments  for (String index : args)  System.out.println(index);  }  }  } |