

Shift operators

0000 1111>>2 =0000 0011

0000 1111<<2 = 0011 1100

**Java Strings**

Java Strings are immutable. This means that it is not possible to change the values of the characters withing a string.

When we declare String, a string pool will be created.

While creating through

String str=new String(“Hello”)

Its store in heap table.

Print F specifier

%b a Boolean value true /false

%c a character ‘a’

%d

String s= “Hello, how, are, you , doing”

String[] str=s.split(“.|,|\\.);

Arrays.copyOf(arr, len)

System.arraycopy(from, fromindex, to, toindex, count)

Arrays.sort(arr)

Arrays.equals(arr1,arr2)

Split array

t="I, like to have fun. I want to eat icecream."

t.split(", |,|\\. | |\\.")

$9 ==> String[10] { "I", "like", "to", "have", "fun", "I", "want", "to", "eat", "icecream" }

|  |  |  |  |
| --- | --- | --- | --- |
|  | String | StringBuilder | StringBuffer |
| Modifiable | Immutable | Mutable | Mutable |
| Threadsafe | No | No | yes |

In Java, binding refers to the process of connecting a method call to the method body. There are two types of binding in Java: static binding (or early binding) and dynamic binding (or late binding).

1. **Static Binding (Early Binding):**
   * **Occurs at Compile Time:** In static binding, the compiler determines the method that needs to be called during compile time.
   * **Method Overloading:** It is often associated with method overloading, where the compiler decides which method to call based on the method signature.
   * **Example:**

javaCopy code

class StaticBindingExample { void display() { System.out.println("Static Binding Example"); } public static void main(String[] args) { StaticBindingExample obj = new StaticBindingExample(); obj.display(); // Compiler can determine the method at compile time } }

1. **Dynamic Binding (Late Binding):**
   * **Occurs at Runtime:** In dynamic binding, the determination of the method to be called is made at runtime.
   * **Method Overriding:** It is associated with method overriding in the context of inheritance. The actual method called is resolved during runtime based on the actual type of the object.
   * **Example:**

javaCopy code

class Animal { void sound() { System.out.println("Animal makes a sound"); } } class Dog extends Animal { void sound() { System.out.println("Dog barks"); } } public class DynamicBindingExample { public static void main(String[] args) { Animal obj = new Dog(); obj.sound(); // Compiler doesn't know the actual type at compile time } }

1. In the above example, **sound()** is overridden in the **Dog** class. During runtime, the actual type of the object (**Dog**) is determined, and the overridden method in the **Dog** class is called.

In summary, static binding occurs at compile time and is associated with method overloading, while dynamic binding occurs at runtime and is associated with method overriding in the context of inheritance. The choice between static and dynamic binding depends on the context and the types of polymorphism used in your code.