JAVA Core 2

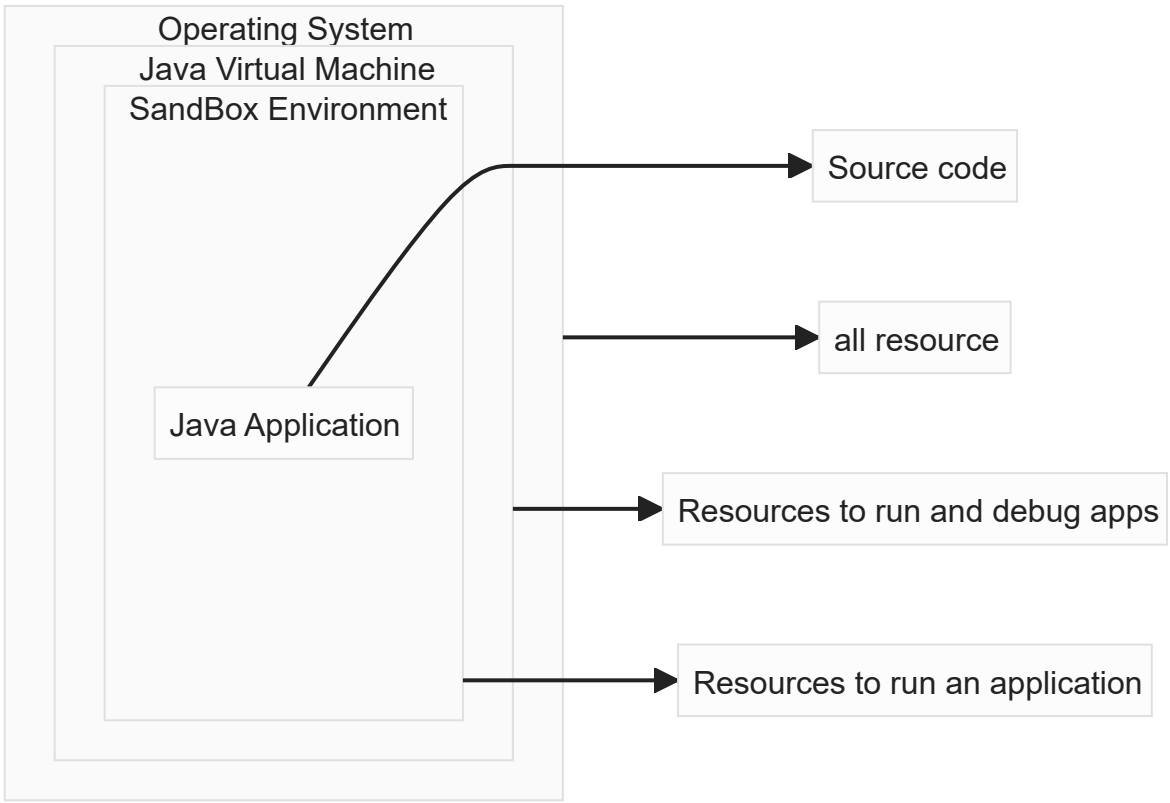
2. References
    1. Creating and Passing Objects
    2. Java Memory management (Stack and Heap)
    3. References Comparison
    4. Object Class methods and Java lang package

2. References

In Java while creating classes, variables, functions everything is stored in the RAM memory while running the program, JVM is responsible for storing and loading the class while running java program.

While working with java source code, JVM creates an isolated environment that is secured by policies and have limited access control over the resources this process is known as sandboxing

A **sandbox** is a security mechanism that provides a restricted environment for running applications. It is designed to contain and isolate potentially harmful code, preventing it from causing damage to the system or accessing sensitive resources.



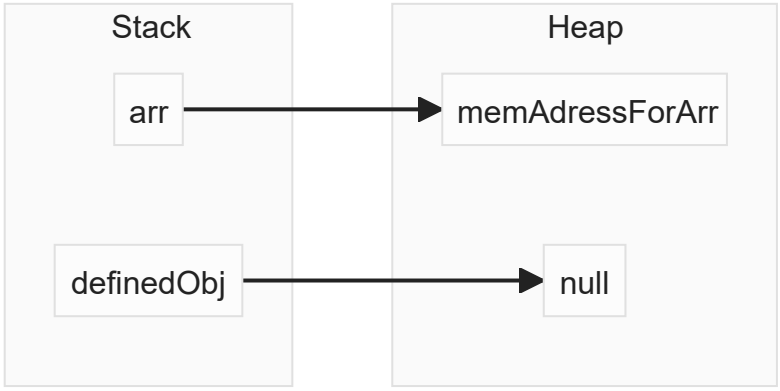
While running java application, how and when to load the classes is the responsibility of JVM until then the classes were not allocated in the memory. Ones the classes is loaded it is allocated memory in the Method Area or Meta space this area is completely different from stack and heap memory.

Stack Memory is responsible for having the variables which points to the memory location in either the heap or have memory in the stack itself. It follows the FIFO order

Heap Memory is the area where the actual memory is allocated for any object or non-primitive data type. It works just like a Tree and have dynamic memory allocation

2.1 Creating and passing Objects

For Objects, Java Manages the memory using heap and stack, the heap memory is dynamic and increases as per the application need, it stores all the non-primitive data, like arrays, object, strings.
it allocates the memory for them while making a reference to the stack memory, where a variable is pointing to the memory location in heap. iff it is non-primitive.



ones the need for arr is over, and it is not pointing in the heap Java's garbage collector will free the memory.
The memory in heap is allocated using (new) key word. to make an object in java :
`ObjectClassName nameOfTheObject = new OjectClassName();`

This can be divided into 2 phases :

- **Phase 1 :** Left Hand Side of "=" represent that a variable of name "nameOfTheObject" is made in the stack memory and is pointing to nothing.
- **Phase 2 :** Right Hand Side of "=" represent that the variable is now pointing to a new memory location allocated in the heap memory.

```
public class TopicRefrences {
        public static void main(String[] args) {
                // only in stack pointing to nothing can't be used
                Human willSmith;
                // object pointing to null
                Human jade = null;
                // properly allocated object
                Human falcon = new Human();
                System.out.println(jade);
                System.out.println(falcon);
        }
}
class Human {

}
```

Whenever the Object is passed in the method, and any changes are done in that object, then those changes are depicted in the original Object as well, **Java supports pass by value only there is no support for pass by reference**, In java data types and objects are always passed by the stack value, the stack value is pointing to the memory address in the heap that is why it will manipulate the real value and will work like passing by reference as seen by the end user.

For example this code will output the `falcon.name` as "falcon" not "homosapein"

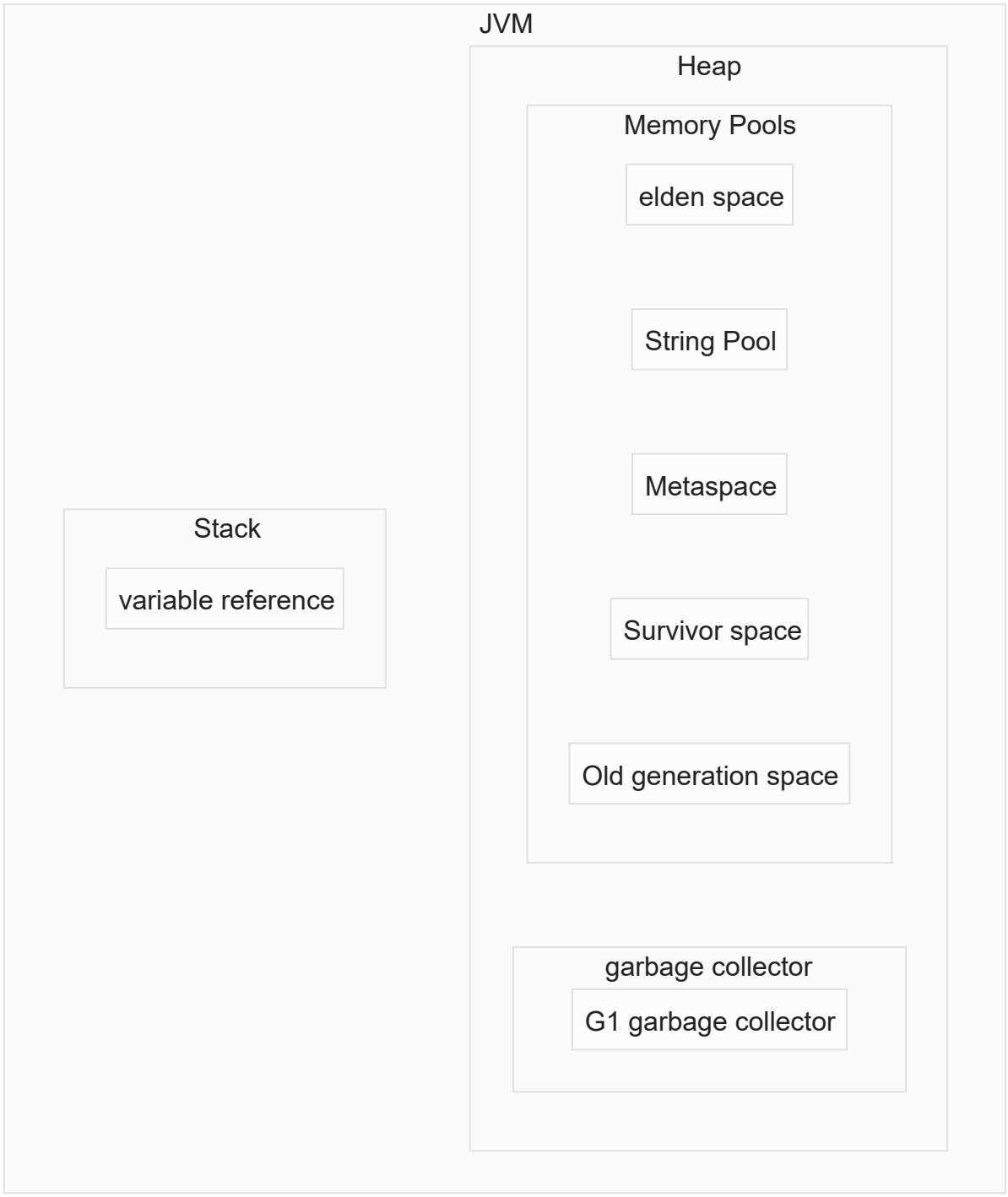
```
public class TopicRefrences {
	// passed by value but changed value for every variable pointing to that
reference
	public static void changeName(Human human, String name) {
		human.name = name;
	}

	public static void main(String[] args) {
		// only in stack pointing to nothing can't be used
		Human willSmith;
		// object pointing to null
		Human jade = null;
		// properly allocated object
		Human falcon = new Human();
		Human f2 = falcon;
		Human f3 = f2;
		changeName(f3, "falcon");
		System.out.println(jade);
		System.out.println(falcon);
		System.out.print(falcon.name);
	}
}

class Human {
	public String name = "homosapein";
}
```

2.2 Java Memory Management (Stack and Heap)

The memory management in java is done by JVM, it is responsible to manage the heap and stack memories that is allocated while running the program, the memory is divided basically in 2 sections - Stack memory, Heap memory.



In Java, memory pools refer to different areas or regions of memory within the Java Virtual Machine (JVM) heap where objects are allocated. These memory pools are used to organize and manage memory in a way that optimizes performance and allows for efficient garbage collection.

1. **Eden Space:**

- Newly created objects are initially allocated in the Eden space of the Young Generation.

2. **Survivor Spaces (S0 and S1):**
   - Objects that survive a garbage collection in the Young Generation are moved to one of the Survivor spaces. The survivor spaces act as a buffer before an object is promoted to the Old Generation.
3. **Old Generation:**
   - Objects that survive multiple garbage collections in the Young Generation are eventually promoted to the Old Generation. Full garbage collections are performed less frequently in the Old Generation.
4. **Metaspace (Java 8 and later):**
   - In Java versions prior to Java 8, the Permanent Generation (PermGen) stored metadata related to classes and methods. In Java 8 and later, PermGen was replaced by Metaspace, which is a native memory area that dynamically adjusts its size.
5. **String Pool:**
   - In Java 8, string literals are stored in the Metaspace (previously PermGen). Strings are interned to reduce memory usage. previously there is a separate space to store string known as string pool.
6. **G1 garbage collector**
   - The Garbage-First (G1) garbage collector is designed to be an improvement over the traditional garbage collectors like the Parallel Garbage Collector and the CMS (Concurrent Mark-Sweep) garbage collector. G1 was introduced in Java 7 and is the default garbage collector in Java 9 and later versions. It is intended to provide better predictability, improved performance, and more efficient use of system resources.

2.3 References Comparison

In Java, when comparing objects for equality or ordering, we need to override or use specific methods to achieve the desired result. Two common types of comparisons in Java are reference comparison (`==` operator) and content-based comparison using `equals()` method.

1. **Reference Comparison (`==` Operator):**
   - Checks if two references point to the same memory location.
   - Returns `true` if the references point to the exact same object in memory.
   - Returns `false` if the references point to different objects, even if the objects have the same content.
   - Used when checking two references point to the same object instance, used for comparing primitive types (e.g., `int`, `char`).
   `object1 == object2`

```
String str1 = new String("Hello");
String str2 = new String("Hello");
// returns false as str1 and str2 pointing to 2 different instance of the String object.
boolean areSameReferences = (str1 == str2);
```

2. **Content-Based Comparison (`equals()` Method):**
   - Checks if two objects have the same content according to the overridden equals() method.
   - It depends on the implementation of the equals() method in the class.
   - By default, the `equals()` method in the `Object` class compares references (same as `==`) sometimes we need classes to override `equals()` to compare content.
   - It is used when comparing the content of objects for equality.
   - Commonly overridden in custom classes to provide meaningful content-based comparisons.
   `object1.equals(object2)`

```
String str1 = new String("Hello");
String str2 = new String("Hello");
boolean areEqualContents = str1.equals(str2); // Returns true (since content is the same)
```

2.4 Object Class methods and Java lang package

The `java.lang` package is a fundamental package in Java and serves as the core of the Java programming language. It contains fundamental classes and interfaces that are automatically imported into every Java program. Many of these classes are part of the Java language itself, providing essential functionality for basic operations.

1. **Object Class**
2. **Class Class**
3. **String Class**
4. **StringBuilder and StringBuffer Classes**
5. **Math Class**
6. **Wrapper Classes**
7. **Enum Class**
8. **Throwable Class and Exception Hierarchy**
9. **Thread Class**
10. **System Class**

Object Class

In Java, the `Object` class is at the root of the class hierarchy, and many classes inherit from it. The `Object` class provides several methods that are commonly used or overridden in other classes. Here are some of the commonly used methods from the `Object` class:

1. `equals(Object obj)`:
   - Compares the current object with the specified object for equality.
   - Typically overridden in user-defined classes to provide meaningful content-based comparisons.
2. `hashCode()`:
   - Returns a hash code value for the object.
   - Overridden in conjunction with `equals()` to maintain consistency when objects are used in collections like `HashMap` or `HashSet`.
3. `toString()`:
   - Returns a string representation of the object.
   - Overridden to provide a meaningful and human-readable string representation of the object.
4. `getClass()`:
   - Returns the runtime class of an object.
   - Retrieves information about the type of the object.
5. `clone()`:
   - Creates and returns a copy of the object.

- The class must implement the `Cloneable` interface, and the `clone()` method should be overridden.

6. **`notify()`, `notifyAll()`, `wait()`:**
   - Methods used for inter thread communication.
   - `notify()`: Wakes up one of the threads that are currently waiting.
   - `notifyAll()`: Wakes up all threads that are currently waiting.
   - `wait()`: Causes the current thread to wait until another thread invokes `notify()` or `notifyAll()`.
   - It is used in multithreading scenarios.
7. **`finalize()`:**
   - Called by the garbage collector before an object is reclaimed.
   - Deprecated in modern Java versions instead use try with resources and the `AutoCloseable` interface for resource management.
   - `AutoCloseavle` interface automatically call garbage collector when the need of resource is full filled