Memory management

Memory is created by java

↳ Memory management is done by JVM.

① Stack → stores temp variable and sepreate memory for methods

- stores pimitive data types
- stores reference to heap object (strong, weak, soft)

* Thread → Owns a stack memory

variables with in the scope is visible only with in the particular scope it gets deleted.

→ When stack memory gets full

↓

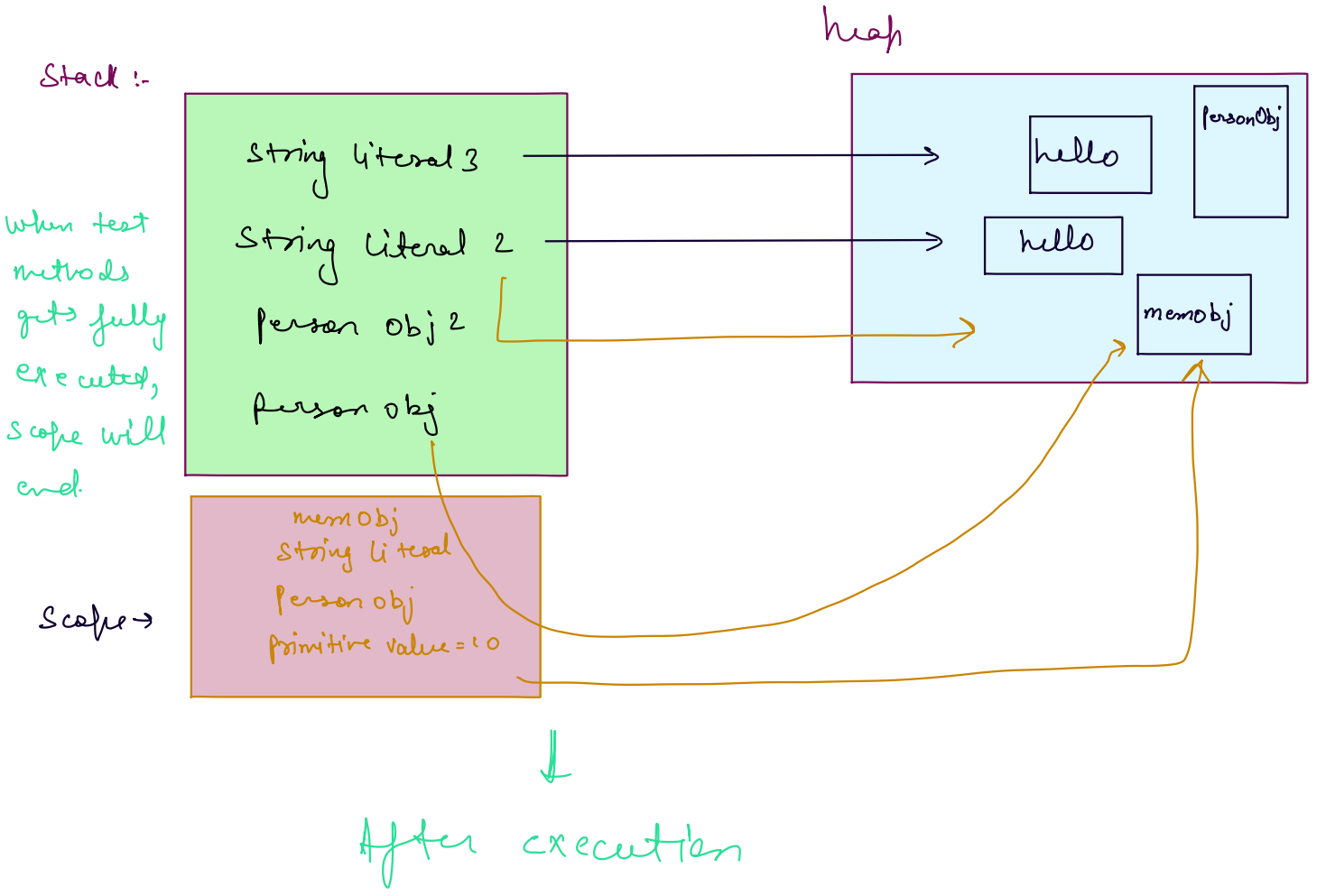
we get stack overflow.

* Heap Memory → Stores object, no order is present

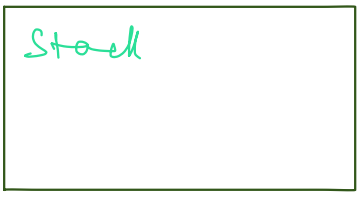
→ garbage collector → used to delete unreferenced object from heap.

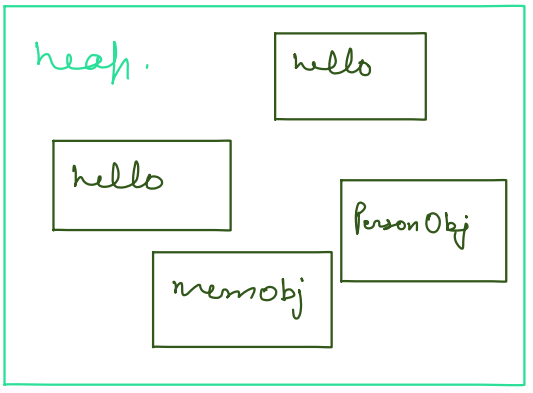
↳ Types → single gc, parallel gc, cms (concurrent mark & sweep) → mark & sweep

☆ heap memory is shared across all threads.



After execution





garbage collector
↓
System.gc()
↓
(depends upon heap memory)

```
2 usages
public class Memory {
    no usages
    int primitivevariable = 10;//primitve data type
    1 usage
    Person p = new Person();// object
    no usages
    String stringliteral="Hello";//string literqal
    1 usage
    Memory memObj=new Memory();
    memObj.memoryManagementTest(p);
    no usages
    private void memoryManagementTest(Person  p){
        Person p2=p; //reference to object
        String stringliteral2="Hello"; //strig literal
        String stringliteral3=new String( original: "Hello");// string object
    }
}
```

Types of reference

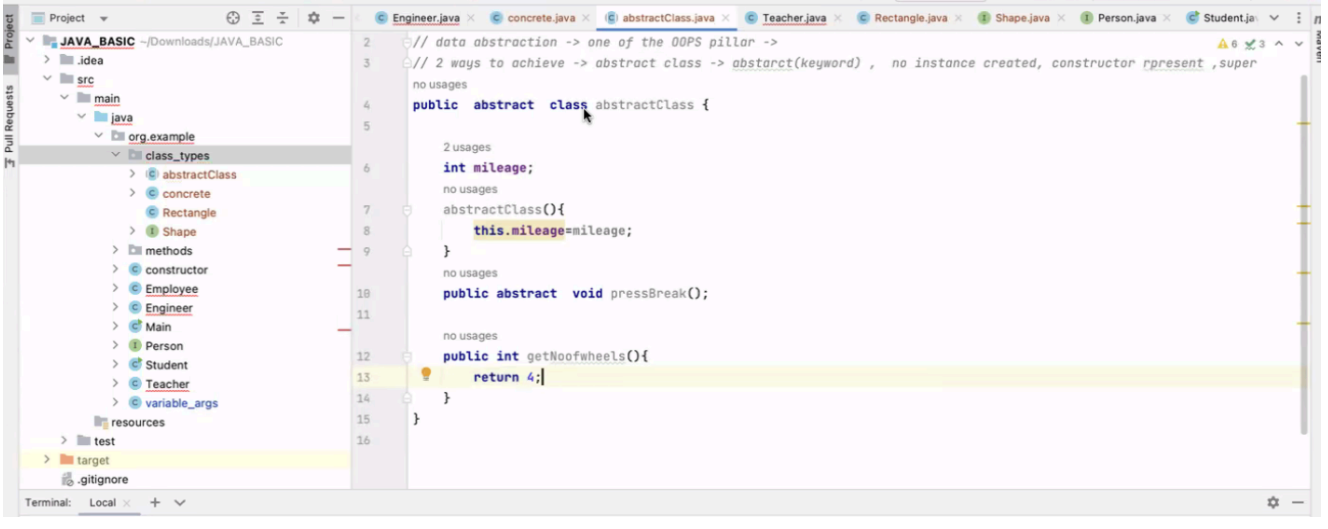
- Strong → Stack variable is referencing an obj in heap.
- weak → Weak Reference <Person> weakObj= new WeakReference <Person>(new Person);
- Soft → shortage of space

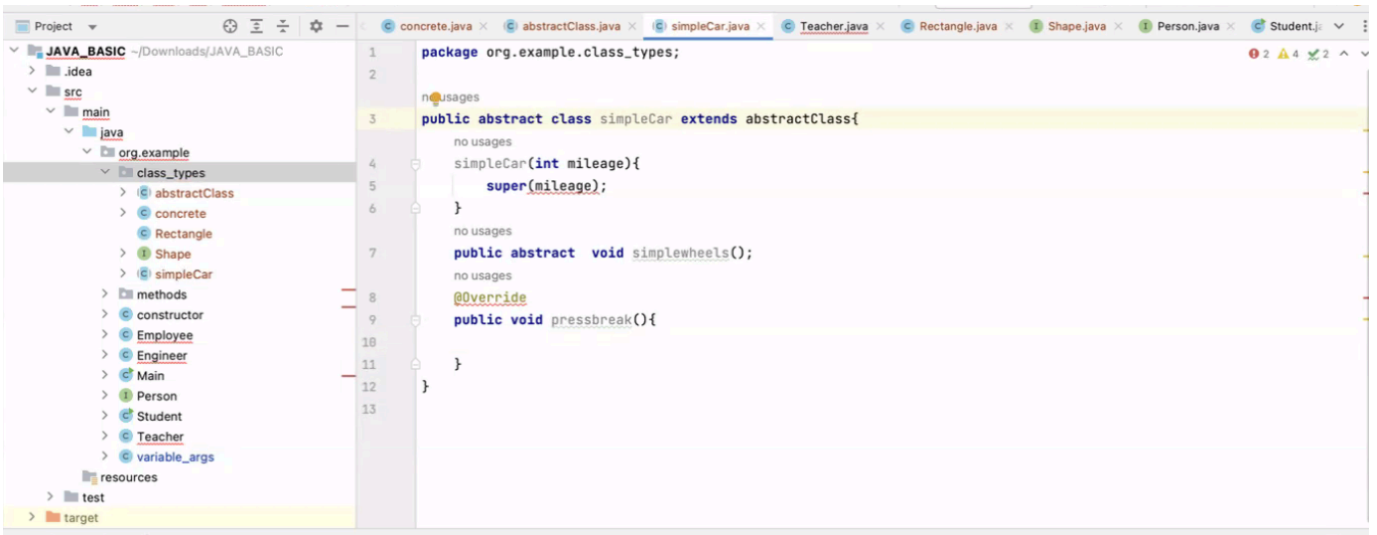
```
Person obj1 = new Person();  1 usage

Person obj2 = new Person();  1 usage
obj1 = obj2; // obj1 1 will be deleted first
```

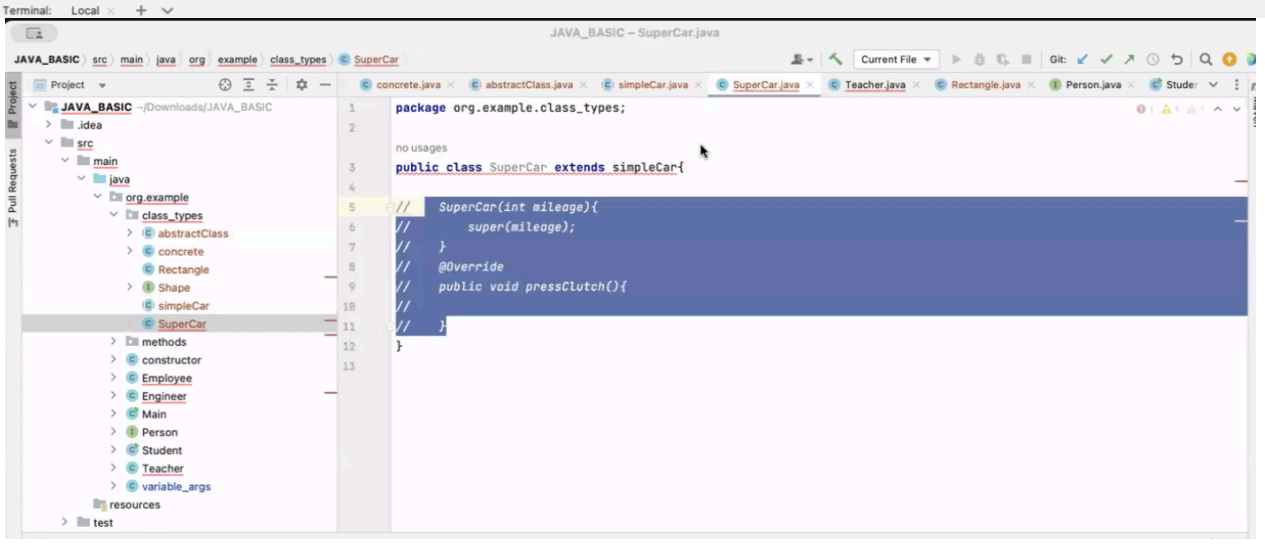
* We can create instance by new keyword
* All the classes have implementation.
* It can also be a child class from interface or extend abstract class → public / private
* Data abstraction → one of the oops pillar

2 ways to achieve

1) Abstract (keyword)
2) no instance created. (constructor represent, super keyword)







→ static → load JVM → class → static

→ Object → instance block execute → constructor

if 2 objects instance blocks also executes twice

```java
public class MyFirst {
    public static void main(String[] args) {
        MyFirst obj = new MyFirst(n);
    }
    static int a = 10;  2 usages
    static int n;  3 usages
    int b = 5;  2 usages
    int c;  1 usage
    public MyFirst(int m) {  1 usage
        System.out.println(a + ", " + b + ", " + c + ", " + n + ", " + m);
    }
    // Instance Block
    {
        b = 30;
        n = 20;
    }
    // Static Block
    static
    {
        a = 60;
    }
}
```

60, 30, 0, 20, 0

Q2

```
public class First_C {
      public void myMethod()
    {
    System.out.println("Method");
    }

    {
    System.out.println(" Instance Block");
    }

    public void First_C()
    {
    System.out.println("Constructor ");
    }
    static {
        System.out.println("static block");
    }
    public static void main(String[] args) {
    First_C c = new First_C();
    c.First_C();
    c.myMethod();
  }
}
```

Static block
Instance block

Constructor
Method

Q3.

```
class Base {
  public void show() {
    System.out.println("Base::show() called");
  }
}

class Derived extends Base {
  public void show() {
    System.out.println("Derived::show() called");
  }
}

public class Main {
  public static void main(String[] args) {
    Base b = new Derived();;
    b.show();
  }
}
```

Q4.

```
final class Complex {

  private final double re;
  private final double im;

  public Complex(double re, double im) {
    this.re = re;
    this.im = im;
  }

  public String toString() {
    return \"(\" + re + \" + \" + im + \"i)\";
  }
}

class Main {
  public static void main(String args[])
  {
    Complex c = new Complex(10, 15);
    System.out.println(\"Complex number is \" + c);
  }
}
```

Q5

```
class Base {
  final public void show() {
    System.out.println("Base::show() called");
  }
}

class Derived extends Base {
  public void show() {
    System.out.println("Derived::show() called");
  }
}

class Main {
  public static void main(String[] args) {
    Base b = new Derived();;
    b.show();
  }
```

```
class Base {
    public static void show() {
        System.out.println(\"Base::show() called\");
    }
}

class Derived extends Base {
    public static void show() {
        System.out.println(\"Derived::show() called\");
    }
}

class Main {
    public static void main(String[] args) {
        Base b = new Derived();
        b.show();
    }
}
```

```
abstract class demo
{
    public int a;
    demo()
    {
        a = 10;
    }

    abstract public void set();

    abstract final public void get();

}

class Test extends demo
{

    public void set(int a)
    {
        this.a = a;
    }

    final public void get()
    {
        System.out.println(\"a = \" + a);
    }

    public static void main(String[] args)
    {
        Test obj = new Test();
        obj.set(20);
        obj.get();
    }
}
```

```
class Test
{
    public static void main (String[] args)
    {
```

```
        int arr1[] = {1, 2, 3};
        int arr2[] = {1, 2, 3};
        if (arr1 == arr2)
            System.out.println(\"Same\");
        else
            System.out.println(\"Not same\");
    }
}
```

```
class Test {
  public static void main(String args[]) {
    int arr[] = new int[2];
    System.out.println(arr[0]);
    System.out.println(arr[1]);
  }
}
```

```
class Test
{
  public static void main (String[] args)
  {
    int arr1[] = {1, 2, 3};
    int arr2[] = {1, 2, 3};
    if (arr1.equals(arr2))
      System.out.println(\"Same\");
    else
      System.out.println(\"Not same\");
  }
}
```

```
public class Main {
  public static void main(String args[]) {
    int arr[] = {10, 20, 30, 40, 50};

    for(int i=0; i < arr.length; i++)
    {
      System.out.print(\" \" + arr[i]);
    }
  }
}
```

```
class Main {
  public static void main(String args[]) {
    int t;
    System.out.println(t);
  }
}
```

```
class Test {
  public static void main(String[] args) {
    for(int i = 0; 0; i++)
    {
      System.out.println(\"Hello\");
      break;
    }
  }
}
```

```
class Test
{
  public static void main(String[] args)
  {
    Double object = new Double(\"2.4\");
    int a = object.intValue();
    byte b = object.byteValue();
    float d = object.floatValue();
    double c = object.doubleValue();

    System.out.println(a + b + c + d );

  }
}
```

```
class Test
{
  static int a;

  static
  {
    a = 4;
    System.out.println (\"inside static block\\n\");
    System.out.println (\"a = \" + a);
  }

  Test()
  {
    System.out.println (\"\\ninside constructor\\n\");
    a = 10;
  }

  public static void func()
  {
    a = a + 1;
    System.out.println (\"a = \" + a);
  }

  public static void main(String[] args)
  {

    Test obj = new Test();
    obj.func();

  }
}
```

```
class Point {
  int m_x, m_y;

  public Point(int x, int y) { m_x = x; m_y = y; }
  public Point() { this(10, 10); }
  public int getX() { return m_x; }
  public int getY() { return m_y; }

  public static void main(String args[]) {
    Point p = new Point();
    System.out.println(p.getX());
  }
}
```

```
package main;
class T {
  int t = 20;
}
class Main {
  public static void main(String args[]) {
    T t1 = new T();
    System.out.println(t1.t);
  }
}
```


```
package main;
class T {
  int t = 20;
}
class Main {
  public static void main(String args[]) {
    T t1 = new T();
    System.out.println(t1.t);
  }
}
```


```
class Test
{  int count = 0;

  void A() throws Exception
  {
    try
    {
      count++;

      try
      {
        count++;

        try
        {
```

```
            count++;
            throw new Exception();

        }

        catch(Exception ex)
        {
            count++;
            throw new Exception();
        }
    }

    catch(Exception ex)
    {
        count++;
    }
  }

  catch(Exception ex)
  {
    count++;
  }

}

void display()
{
  System.out.println(count);
}

public static void main(String[] args) throws Exception
{
  Test obj = new Test();
  obj.A();
  obj.display();
}
}
```

or

```
class Base extends Exception {}
class Derived extends Base  {}

public class Main {
  public static void main(String args[]) {
   // some other stuff

  try {
     // Some monitored code
     throw new Derived();
  }
  catch(Base b)    {
     System.out.println(\"Caught base class exception\");
  }
  catch(Derived d)  {
     System.out.println(\"Caught derived class exception\");
  }
 }
}
```

only

```
class Main {
  public static void main(String args[]) {
     try {
        throw 10;
     }
     catch(int e) {
        System.out.println(\"Got the  Exception \" + e);
     }
  }
}
```

Q1

```
// Note static keyword after import.
import static java.lang.System.*;

class StaticImportDemo
{
  public static void main(String args[])
  {
      out.println(\"FHSJHAKJ\");
  }
}
```

"FHSJHAK"

Q1

```
class simple
{

public static void main(String[ ] args)
{
simple obj = new simple( );
obj.start( );
}
void start( )
{
long [ ] P= {3, 4, 5};
long [ ] Q= method (P);
System.out.print (P[0] + P[1] + P[2]+":");
System.out.print (Q[0] + Q[1] + Q[2]);
}
long [ ] method (long [ ] R)
{
R [1]=7;
return R;
}
} //end of class
```

Q1

```
Class Test {
   public static void main(String[] args) {
      Test obj = new Test();
      obj.start();
   }
   void start() {
      String stra = "do";
         String strb = method(stra);
      System.out.print(": "+stra + strb);
   }
   String method(String stra) {
      stra = stra + "good";
      System.out.print(stra);
      return good";
   }
}
```

Q7

```
class Test {
  int i;
}
class Main {
  public static void main(String args[]) {
     Test t = new Test();
     System.out.println(t.i);
   }
}
```

```
// precondition: x>=0
public void demo(int x)
{
   System.out.print(x % 10);
   if (x % 10 != 0) {
      demo(x / 10);
   }
   System.out.print(x % 10);
}
```

Which of the following is printed as a result of the call **demo(1234)**?

```
class Test
{
   public void demo(String str)
   {
      String[] arr = str.split(\";\");
      for (String s : arr)
      {
         System.out.println(s);
      }
   }

   public static void main(String[] args)
   {
      char array[] = {\'a\', \'b\', \' \', \'c\', \'d\', \';\', \'e\', \'f\', \' \',
                      \'g\', \'h\', \';\', \'i\', \'j\', \' \', \'k\', \'l\'};
      String str = new String(array);
      Test obj = new Test();
      obj.demo(str);
   }
}
```

Q8

```
class Test {
public static void swap(Integer i, Integer j) {
     Integer temp = new Integer(i);
     i = j;
     j = temp;
   }
   public static void main(String[] args) {
     Integer i = new Integer(10);
     Integer j = new Integer(20);
     swap(i, j);
     System.out.println(\"i = \" + i + \", j = \" + j);
   }
}
```

i=20 , j=10

Q1

```
class Main {
 public static void main(String args[]){
   final int i;
   i = 20;
   System.out.println(i);
 }
}
```

→ 20

Q2

```
class Main {
 public static void main(String args[]){
   final int i;
   i = 20;
   i = 30;
   System.out.println(i);
 }
}
```

→ error
↓
final cannot be overwritten

Q3

```
class Base {
 public final void show() {
     System.out.println(\"Base::show() called\");

   }
}
class Derived extends Base {
   public void show() {
      System.out.println(\"Derived::show() called\");
   }
}
public class Main {
   public static void main(String[] args) {
       Base b = new Derived();;
       b.show();
   }
}
```

→ Compilation error
↓
final → constant
↓
we cannot overwrite this.

Q4

```
class T {
  int t = 20;
  T() {
    t = 40;
  }
}
class Main {
   public static void main(String args[]) {
      T t1 = new T();
      System.out.println(t1.t);
   }
}
```

→ → 40 (overwrite)
↓
constructor called.