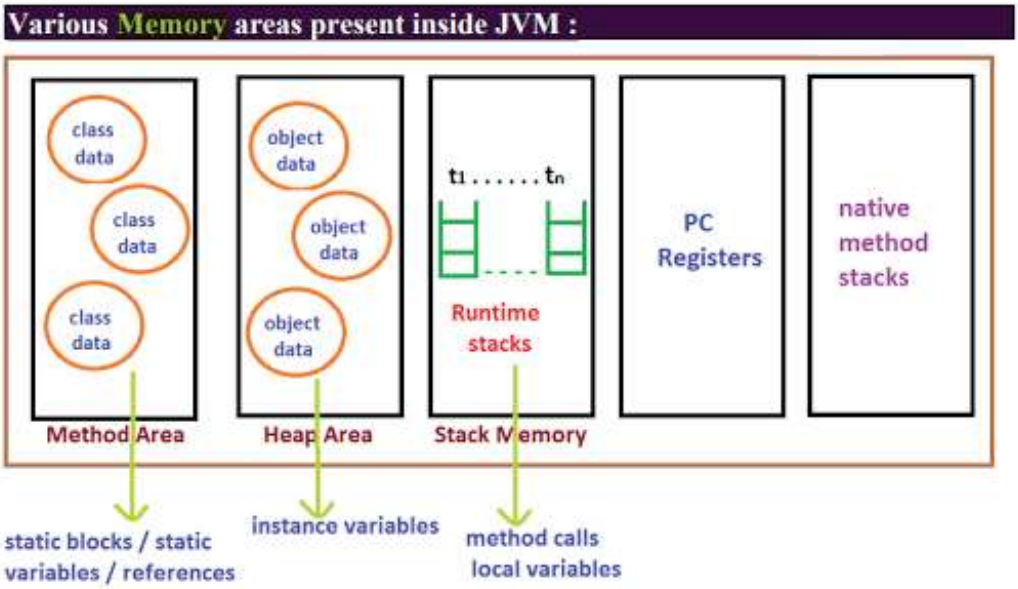


- Class level binary data including static variables will be stored in method area
- Objects and corresponding instance variables will be stored in the Heap area.
- For every method the JVM will create a Runtime stack, all method calls performed by that Thread and corresponding local variables will be stored in that stack. Every entry in stack is called Stack Frame or Action Record.
- The instruction which has to execute next will be stored in the corresponding PC Registers.
- Native method invocations will be stored in native method stacks.

## MethodOverloading

- Two methods are said to be overloaded if and only if both have the same name but different argument types.
- In the 'C' language we can't take 2 methods with the same name and different types. If there is a change in argument type compulsory we should go for a new method name.

**Example :**

abs() for int datatype
labs() for long datatype
fabs() for float datatype

- Lack of overloading in "C" increases complexity of the programming.
- But in java we can take multiple methods with the same name and different argument types.

abs(int) for int datatype
abs(long) for long datatype
abs(float) for float datatype

- Having the same name and different argument types is called method overloading. All these methods are considered as overloaded methods.
- Having overloading concept in java reduces complexity of the programming

**Example**

```
class Test {
	public void m1(){
		System.out.println("zero arg method");
	}
	public void m1(int i){
		System.out.println("int arg method");
	}
	public void m1(double d){
		System.out.println("double arg method");
	}
	public static void main(String[] args) {
		Test t = new Test();
		t.m1();
		t.m1(10);
		t.m1(10.5);
	}
}
```

Output
zero arg method
int arg method
double arg method

**Note:**
**Conclusion :**
**In overloading, the compiler is responsible to perform method resolution(decision) based on the reference type(but not based on run time object). Hence overloading is also considered as compile time polymorphism(or) static polymorphism (or)early binding.**

## Case 1: Automatic promotion in overloading.

- In overloading if the compiler is unable to find the method with exact match we won't get any compile time error immediately.
- First the compiler promotes the argument to the next level and checks whether the matched method is available or not if it is available then that method will be considered if it is not available then the compiler promotes the argument once again to the next level.
- This process will be continued until all possible promotions still if the matched method is not available then we will get a compile time error. This process is called automatic promotion in overloading.

The following are various possible automatic promotions in overloading.

**Example**

```
class Test {
	public void m1(int i){
		System.out.println("int arg method");
	}
	public void m1(float d){
		System.out.println("float arg method");
	}
	public static void main(String[] args) {
		Test t = new Test();
		t.m1('a');//int arg method
		t.m1(10l);//float arg method
		t.m1(10.5);//CE
	}
}
```

**Example**

**CASE-2**

```
class Test {
	public void m1(String s){
		System.out.println("String arg method");
	}
	public void m1(Object d){
		System.out.println("Object arg method");
	}
	public static void main(String[] args) {
		Test t = new Test();
		t.m1("sachin");//String arg method
		t.m1(new Object());//Object arg method
		t.m1(null);//String arg method
	}
}
```

**Note :**

While resolving overloaded methods, exact matches will always get high priority.

While resolving overloaded methods, the child class will get more priority than the parent class.

**Example**

**CASE-3**

```
class Test {
	public void m1(String s){
		System.out.println("String arg method");
	}
	public void m1(StringBuffer d){
		System.out.println("StringBuffer arg method");
	}
	public static void main(String[] args) {
		Test t = new Test();
		t.m1("sachin");//String arg method
		t.m1(new StringBuffer("dhoni"));//StringBuffer arg method
		t.m1(null);//CE
	}
}
```

**Example**

**CASE-4**

```
class Test {
	public void m1(int i,float f){
		System.out.println("int, float arg method");
	}
	public void m1(float f, int i){
		System.out.println("float,int arg method");
	}
	public static void main(String[] args) {
		 Test t = new Test();
		 t.m1(10,10.5f);//int,float arg method
		 t.m1(10.5f,10);//float,int arg method
		 t.m1(10,10);//CE
		 t.m1(10.5f,10.5f);//CE

	}
}
```