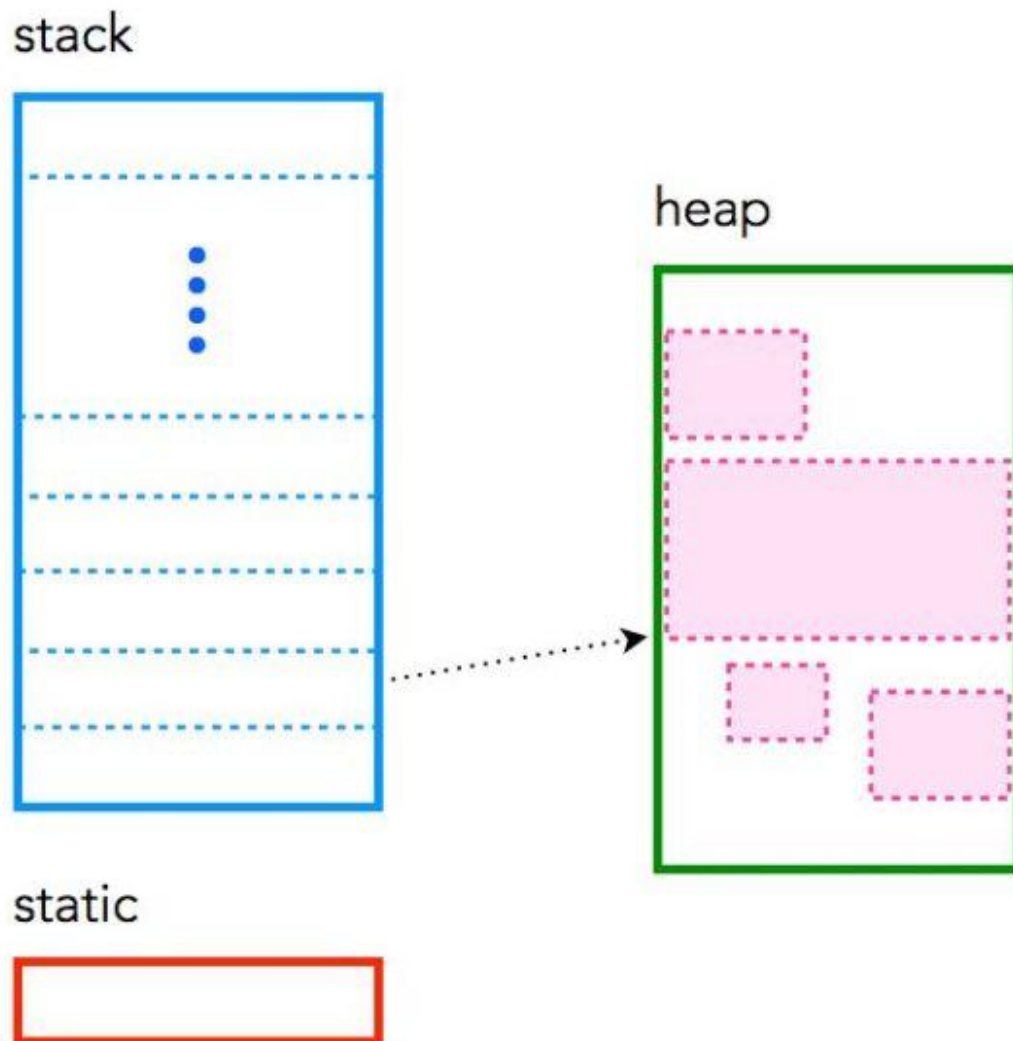


## 1)MEMORY IN C – THE STACK, THE HEAP, AND STATIC:

The great thing about C is that it is so intertwined with memory – and by that I mean that the programmer has quite a good understanding of “*what goes where*“. C has three different pools of memory.

- **static**: global variable storage, permanent for the entire run of the program.
- **stack**: local variable storage (automatic, continuous memory).
- **heap**: dynamic storage (large pool of memory, not allocated in contiguous order).



**STATIC MEMORY**

Static memory persists throughout the entire life of the program, and is usually used to store things like *global* variables, or variables created with the static clause.

For example:

```
int theforce;
```

On many systems this variable uses 4 bytes of memory. This memory can come from one of two places. If a variable is declared *outside* of a function, it is considered global, meaning it is accessible anywhere in the program. Global variables are static, and there is only one copy for the entire program. Inside a function the variable is allocated on the stack. It is also possible to force a variable to be static using the static clause. For example, the same variable created inside a function using the static clause would allow it to be stored in static memory.

```
static int theforce;
```

## STACK MEMORY:

The *stack* is used to store variables used on the inside of a function (including the `main()` function). It's a LIFO, "Last-In,-First-Out", structure. Every time a function declares a new variable it is "pushed" onto the stack. Then when a function finishes running, all the variables associated with that function on the stack are deleted, and the memory they use is freed up. This leads to the "local" scope of function variables. The stack is a special region of memory, and automatically managed by the CPU – so you don't have to allocate or deallocate memory. Stack memory is divided into successive frames where each time a function is called, it allocates itself a fresh stack frame.

Note that there is generally a limit on the size of the stack – which can vary with the operating system (for example OSX currently has a default stack size of 8MB). If a program tries to put too much information on the stack, stack overflow will occur. Stack overflow happens when all the memory in the stack has been allocated, and further allocations begin overflowing into other sections of memory. Stack overflow also occurs in situations where recursion is incorrectly used.

A summary of the stack:

- the stack is managed by the CPU, there is no ability to modify it
- variables are allocated and freed automatically
- the stack is not limitless – most have an upper bound
- the stack grows and shrinks as variables are created and destroyed
- stack variables only exist whilst the function that created them exists

## HEAP MEMORY

The *heap* is the diametrical opposite of the stack. The *heap* is a large pool of memory that can be used dynamically – it is also known as the "free store". This is memory that is not

automatically managed – you have to explicitly allocate (using functions such as `malloc`), and deallocate (e.g. `free`) the memory. Failure to free the memory when you are finished with it will result in what is known as a *memory leak* – memory that is still “being used”, and not available to other processes. Unlike the stack, there are generally no restrictions on the size of the heap (or the variables it creates), other than the physical size of memory in the machine. Variables created on the heap are accessible anywhere in the program.

Oh, and heap memory requires you to use **pointers**.

#### **A summary of the heap:**

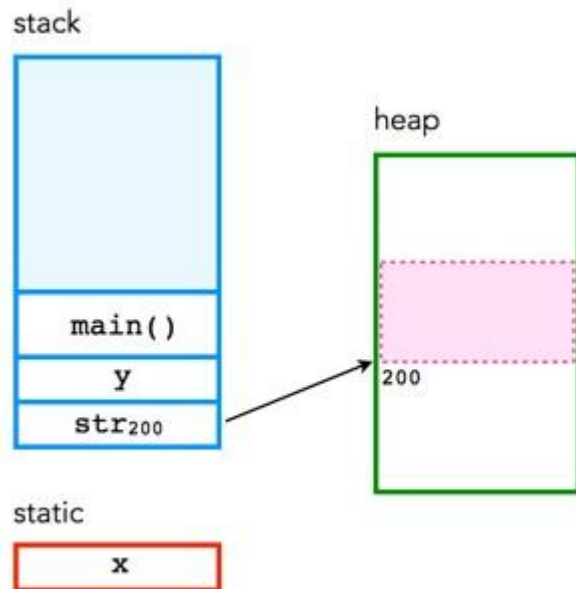
- the heap is managed by the programmer, the ability to modify it is somewhat boundless
- in C, variables are allocated and freed using functions like `malloc()` and `free()`
- the heap is large, and is usually limited by the physical memory available
- the heap requires pointers to access it

#### **AN EXAMPLE OF MEMORY USE**

Consider the following example of a program containing all three forms of memory:

```
#include <stdio.h> #include <stdlib.h> int x;      int main(void) {  int y;      char *str;
y = 4;  printf("stack memory: %d\n", y);  str = malloc(100*sizeof(char));  str[0] = 'm';
printf("heap memory: %c\n", str[0]);  free(str);      return 0; }
```

The variable **x** is static storage, because of its global nature. Both **y** and **str** are dynamic stack storage which is deallocated when the program ends. The function **malloc()** is used to allocate 100 pieces of dynamic heap storage, each the size of `char`, to **str**. Conversely, the function **free()**, deallocates the memory associated with **str**.



**Stack Allocation:** The allocation happens on contiguous blocks of memory. We call it a stack memory allocation because the allocation happens in the function call stack. The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack. And whenever the function call is over, the memory for the variables is deallocated. This all happens using some predefined routines in the compiler. A programmer does not have to worry about memory allocation and deallocation of stack variables. This kind of memory allocation also known as Temporary memory allocation because as soon as the method finishes its execution all the data belongs to that method flushes out from the stack automatically. Means, any value stored in the stack memory scheme is accessible as long as the method hasn't completed its execution and currently in running state.

#### **Key Points:**

- It's a temporary memory allocation scheme where the data members are accessible only if the method ( ) that contained them is currently running.
- It allocates or deallocates the memory automatically as soon as the corresponding method completes its execution.
- We receive the corresponding error `Java. lang. StackOverFlowError` by JVM, If the stack memory is filled completely.
- Stack memory allocation is considered safer as compared to heap memory allocation because the data stored can only be access by owner thread.
- Memory allocation and deallocation is faster as compared to Heap-memory allocation.
- Stack-memory has less storage space as compared to Heap-memory.

**Heap Allocation:** The memory is allocated during the execution of instructions written by programmers. Note that the name heap has nothing to do with the heap data

structure. It is called heap because it is a pile of memory space available to programmers to allocate and de-allocate. Every time when we made an object it always creates in Heap-space and the referencing information to these objects are always stored in Stack-memory. Heap memory allocation isn't as safe as Stack memory allocation was because the data stored in this space is accessible or visible to all threads. If a programmer does not handle this memory well, a memory leak can happen in the program.

The Heap-memory allocation is further divided into three categories:- These three categories help us to prioritize the data(Objects) to be stored in the Heap-memory or in the Garbage collection.

- 2)What are objects in the program?

An object, in object-oriented programming (OOP), is an abstract data type created by a developer. It can include multiple properties and methods and may even contain other objects. In most programming languages, objects are defined as classes.

Objects provide a structured approach to programming. By defining a dataset as a custom object, a developer can easily create multiple similar objects and modify existing objects within a program. Additionally, objects provide "encapsulation," meaning the data within an object is protected from being modified or destroyed by other functions or methods unless explicitly allowed.

A simple example of an object may be a user account created for a website. The object might be defined as class `userAccount` and contain attributes such as:

- first name
- last name
- email address
- password
- age
- location
- photo

Instead of recreating these properties each time a new user account is created, a web script can simply instantiate a `userAccount` object. Data assigned to the object may be stored in a database if the user account is saved.

A more advanced example of an object is a character in a video game. The character might have standard attributes, such as a name, hitpoints, and movement speed. It may also contain other objects, such as weapons, armor, items, etc. In this case, the character is the "parent object" and the objects it contains are "child objects." Both the parent and child objects can have their own properties and methods. For example, the character may have methods such as "move" and "attack." The "attack" command might reference the "weapon" object, which has its own methods, such as "swing" or "thrust."

**NOTE:** While objects are usually associated with object-oriented programming, in general computer science terminology, an object may refer to a single programming element, such as a variable, constant, function, or method.

- 3)What is the state of obj1, obj2?

#### **1. OBJECT:**

- Object is nothing but collection of data members and method in single unit called object.
- Object is a physical entity. An object might be anything like person, vehicle, things etc. To confirm an entity is an object we can consider three components .

#### **STATE:**

It indicates the characteristics of real world entity and also it refers to the present condition or state of the entity.

Ex: color, height, weight, name etc.

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

**Answer: 0**

**Explanation:** In Java, fields of classes and objects that do not have an explicit initializer and elements of arrays are automatically initialized with the default value for their type (false for boolean, 0 for all numerical types, null for all reference types).

**Local variables in Java must be definitely assigned to before they are accessed, or it is a compile error.**

```
class demo
{
    int a, b;

    demo()
    {
        a = 10;
        b = 20;
    }

    public void print()
    {
        System.out.println ("a = " + a + " b = " + b +
"\n");
    }
}
```

```
class Test
{

    public static void main(String[] args)
    {
        demo obj1 = new demo();
    }
}
```



```
demo obj2 = obj1;

obj1.a += 1;

obj1.b += 1;

System.out.println ("values of obj1 : ");
obj1.print();

System.out.println ("values of obj2 : ");
obj2.print();

}

}
```

**Answer:**

**values of obj1: a = 11 b = 21 values of obj2: a = 11 b = 21**

**Explanation:** Assignment of obj2 to obj1 makes obj2 a reference to obj1. Therefore, any change in obj1 will be reflected in obj2 also.

**4)Do you access all fields of obj1 in the class Tester.java ? Why?**

**"YES" Because**

**Java Classes and Objects**

**We discussed the concept of classes and objects in the previous chapter. Before learning to create classes and objects using Java, let's see one more example.**

**We know that a student has a name and a roll number. So, we can say that Student is a class, and name and roll number are the attributes of this class. Furthermore, we**

know that there can be many students and each student will have a name and a roll number. Thus, all students are the objects of the Student class.

So, you must have understood the basic concept of classes and objects. Now, let's see how to create a class and its objects.

### Creating Classes and Objects in Java

A class is created using the **class** keyword.

```
class Student {    String name;    int roll_no; }
```

In the above declaration, **Student** is the name of a class. In other words, **Student** is a class.

In the body of the class, the attributes **name** and **roll\_no** are defined.

So, we can say that each student has a name and a roll number.

Attributes are variables defined inside a class.

Now let's create an object of this class.

```
// Student class class Student {    String name;    int roll_no; } // Test (Main) class
class Test {    public static void main(String[] args) {        // creating object st of class
Student        Student st = new Student();    } }
```

In the above example, **Test** is the main class containing the **main** method.

The class which has the main method inside it is known as the main class. Till now we were writing all our code in the main class.

We created the **Student** class ha

ving two attributes - **name** and **roll\_no**.

**Student st = new Student()** → We created an object **st** of the **Student** class. We can also say that **st** is a Student.

The terms object and instance are used interchangeably.

Any number of objects of a class can be created.

```
// Student class class Student {    String name;    int roll_no; } // Test (Main) class
class Test {    public static void main(String[] args) {        // creating objects st1 and
st2 of class Student        Student st1 = new Student();        Student st2 = new
Student();    } }
```

Here, we created two objects of the **Student** class - **st1** and **st2**. Thus, **st1** and **st2** are two Students having some name and roll number.

You now know how to create a class with attributes and its objects. Now let's move forward and assign values to attributes.

- 5)What is the current object when the program runs to the line "obj2.createSound();"?

Object-oriented programming System(OOPs) is a programming paradigm based on the concept of "objects" that contain data and methods. The primary purpose of object-oriented programming is to increase the flexibility and maintainability of programs. Object oriented programming brings together data and its behaviour(methods) in a single location(object) makes it easier to understand how a program works.

current object when the program runs to the line "obj2.createSound();" is "obj2"

- 6)In the method main, can you use the keyword "this" to access all fields of obj2? Why?

this keyword in Java is a reference variable that refers to the current object of a method or a constructor. The main purpose of using this keyword in Java is to remove the confusion between class attributes and parameters that have same names.

Following are various uses of 'this' keyword in Java:

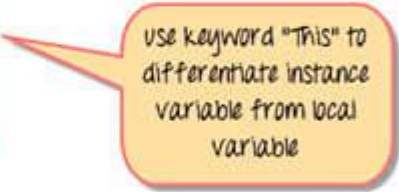
- It can be used to refer instance variable of current class
- It can be used to invoke or initiate current class constructor
- It can be passed as an argument in the method call
- It can be passed as argument in the constructor call
- It can be used to return the current class instance

```

class Account{
int a;
int b;
public void setData(int a , int b){
    this. a=a;
    this. b=b;
}
public static void main(string args[]){
Account obj = new Account();

}

```



**"NO"** We cannot use "this" keyword in main method to access the objects

The "this" keyword is used as a reference to an instance. Since the static methods doesn't have (belong to) any instance you cannot use the "this" reference within a static method. If you still, try to do so a compile time error is generated.

And main method is static therefore, you cannot use the "this" reference in main method.

The static methods belong to the class and they will be loaded into the memory along with the class. You can invoke them without creating an object. (using the class name as reference).

**Example**

```

public class Sample{  static int num = 50;  public static void demo(){
System.out.println("Contents of the static method");  }  public static void
main(String args[]){    Sample.demo();  }}

```

## **Output**

### **Contents of the static method**