Java Training Day-2

OOPs

- User defined / Developer (store multiple data type) /Class / Template / Blueprint Based on the class we create objects
    - Example in C, C++ Structures in C, C++ will be only have variables

```
Struct Employee{
        Int empId;
        Char name[];
        Float salary;
}obj1,obj2;

Obj1.empId=10;
Obj1.name=”Bharath”;
Obj1.salary=10000.00f;
```

    - Example in Java. Classes in java will be have both the variables and methods

```
class Employee{
        Int empId;
        Char name[];
        Float salary;
}
Employee obj1 = new Employee ();
Employee obj2 = new Employee ();
```

New is a keyword/reserved word

When we use new an object will be created or instantiated

Example:

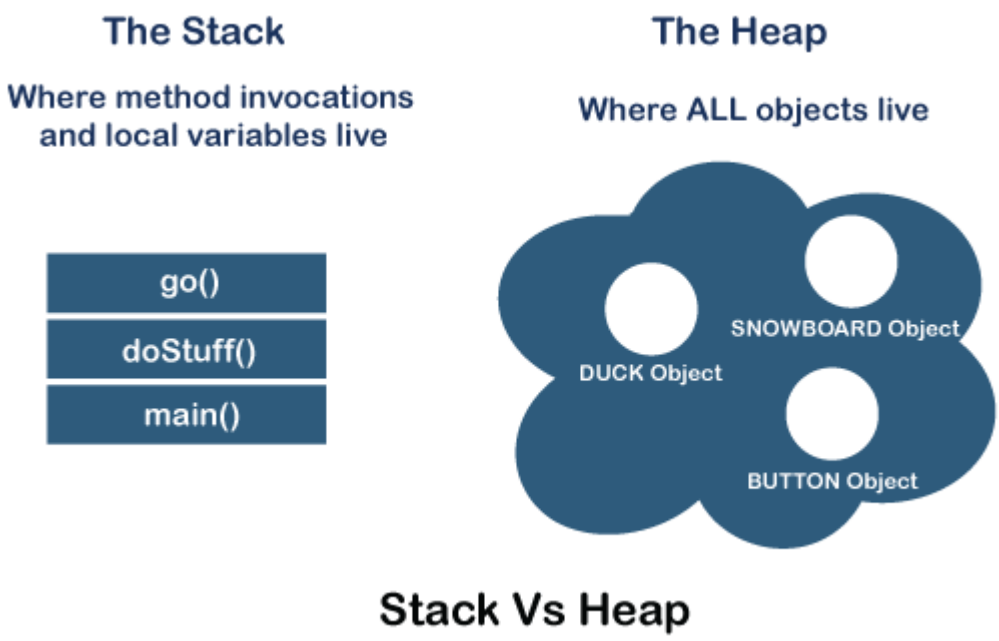
```
Employee obj1 = new Employee ();
Employee obj2 = new Employee ();
```

In the above example Employee is the class and obj1 is the method instantiated from the class Employee

Memory

1) Stack
   The stack memory is a physical space (in RAM) allocated to each thread at run time. It is created when a thread creates. Memory management in the stack follows LIFO (Last-In-First-Out) order because it is accessible globally. It stores the variables, references to objects, and partial results. Memory allocated to stack lives until the function returns. If there is no space for creating the new objects, it throws

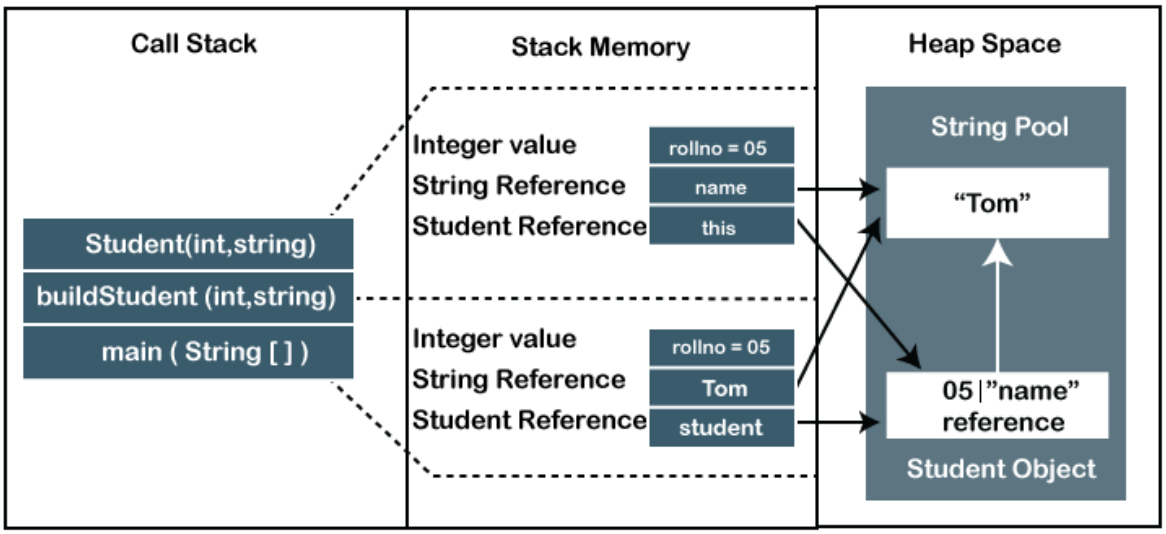
the java.lang.StackOverFlowError. The scope of the elements is limited to their threads. The JVM creates a separate stack for each thread.



Stack Vs Heap

2) Heap

It is created when the JVM starts up and used by the application as long as the application runs. It stores objects and JRE classes. Whenever we create objects it occupies space in the heap memory while the reference of that object creates in the stack. It does not follow any order like the stack. It dynamically handles the memory blocks. It means, we need not to handle the memory manually. For managing the memory automatically, Java provides the garbage collector that deletes the objects which are no longer being used. Memory allocated to heap lives until any one event, either program terminated or memory free does not occur. The elements are globally accessible in the application. It is a common memory space shared with all the threads. If the heap space is full, it throws the java.lang.OutOfMemoryError. The heap memory is further divided into the following memory areas:



Garbage Collector

In java garbage collector will be there in the heap memory location.

The garbage collector will be called using command System.gc ()

The garbage collector will be delete the unused or least frequent used data in the heap and allocate the required memory space for the current running

Assignment operator

= is the assignment operator i.e, LHS = RHS

Ex: int number1=10;

Comparing operator

== is the comparing operator

Comparing the LHS and RHS values for the validations

Ex: int a=10;

if (a==10){

Print("Correct")

}

Operators in Java

Operators are the symbols that we use to perform operations. There are many types of operators in Java.

- Unary Operator,
  - The Java unary operators require only one operand. Unary operators are used to perform various operations.
  - incrementing/decrementing a value by one
  - negating an expression
  - inverting the value of a Boolean
    - Example :
    - Postfix : *expr*++, *expr*—
    - Prefix: ++*expr*, --*expr*, +*expr*,-*expr* ~!

```
public class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++); //10 (11)
System.out.println(++x); //12
```

```
System.out.println(x--); //12 (11)
System.out.println(--x); //10
}}
```

- Arithmetic Operator,
  - All mathematical operations can be performed by using the arithmetic operators. Ex: +, -, *, /, %. BODMAS rule will be applicable during arithmetic operations.
- Shift Operator,
  - Left shift operator
    - The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

      Ex:
      ```
      public class OperatorExample{
      public static void main(String args[]){
      System.out.println(10<<2);//10*2^2=10*4=40
      System.out.println(10<<3);//10*2^3=10*8=80
      System.out.println(20<<2);//20*2^2=20*4=80
      System.out.println(15<<4);//15*2^4=15*16=240
      }}
      ```
  - Right shift operator
    - The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

      ```
      public OperatorExample{
      public static void main(String args[]){
      System.out.println(10>>2);//10/2^2=10/4=2
      System.out.println(20>>2);//20/2^2=20/4=5
      System.out.println(20>>3);//20/2^3=20/8=2
      }}
      ```
- Relational Operator,
  - Compariosion < , > , <=, >=
  - Equality ==, !=
- Bitwise Operator,
  - Bitwise AND &
  - Bitwise exclusive OR ^
  - Bitwise inclusive OR |
- Logical Operator,
  - Logical AND &&
  - Logical OR ||

- Ternary Operator
  - Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

```
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

- Assignment Operator.
  - =, +=, -=, *=, /=, %=, ^=, &=, <<=, >>=, >>>=

Class in java (Syntax)

```
class <ClassName>{
    <dataType> variable1: variableValue; // Property/attribute/ In oops state
    <dataType> variable2: variableValue;
    <dataType> variable3: variableValue;

    <returnDataType> <methodName>{
        …….
        ……..
    }// body of the method

}//body of the class
```

Assignment

1. Sub, mul, division
2. Print name dynamically
3. Get the full name of the employee (return)
4. Create two employees and sum their salary
5. Create two employees and let me know who is drawing higher salary

When we code

We will create the class and then the objects will be instantiated from the class

When we think

We will first think about the object and create a class of the which methods need to be in the class.

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access (Refer this for details).
2. **class keyword:** class keyword is used to create a class.
3. **Class name:** The name should begin with an initial letter (capitalized by convention).
4. **Superclass (if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces (if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body surrounded by braces, { }.

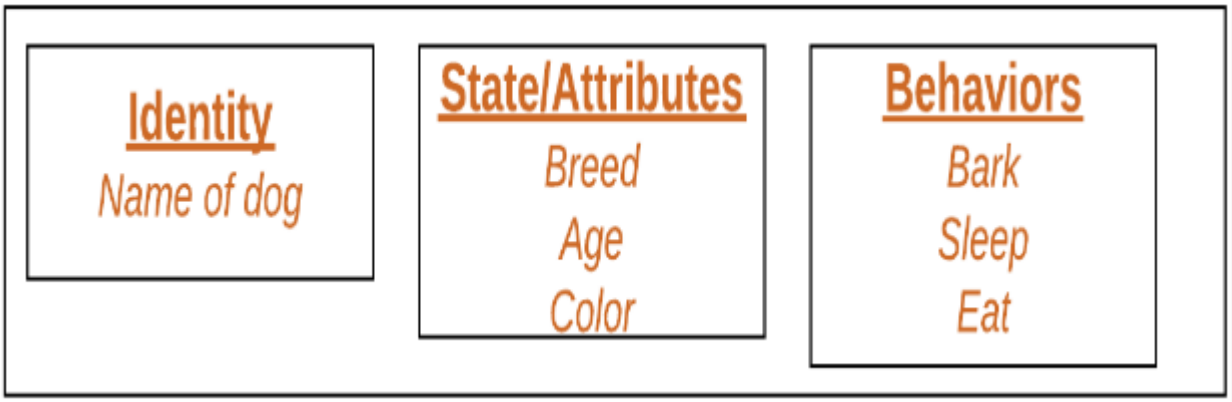
Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behaviour of the class and its objects.
There are various types of classes that are used in real time applications such as nested classes, anonymous classes, lambda expressions.

Object

It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behaviour**: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object

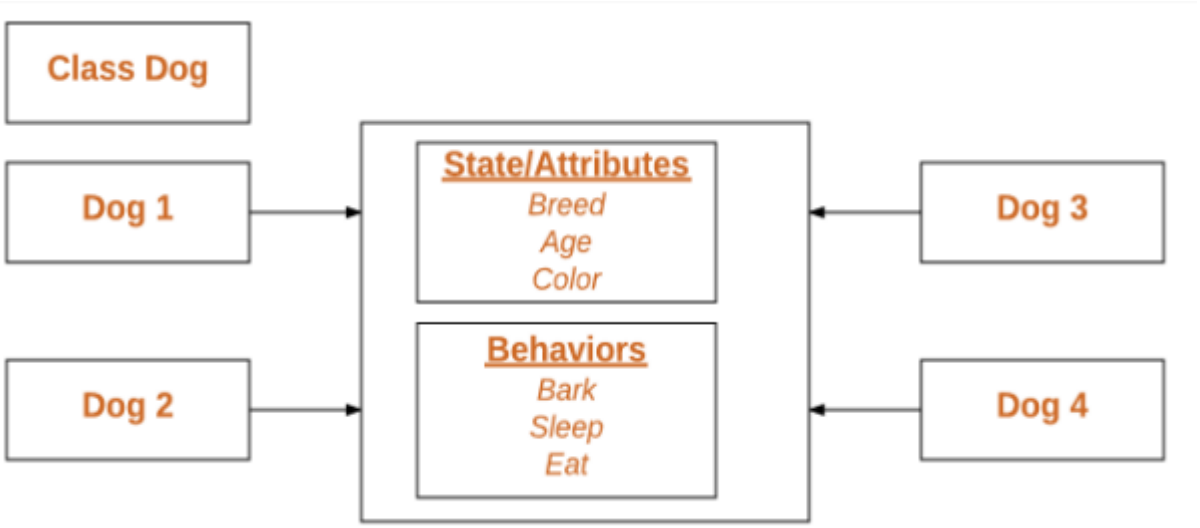


Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behaviour of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example:



As we declare variables like (type name ;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general, we **can’t** create objects of an abstract class or an interface.
Dog tuffy;

If we declare reference variable (tuffy) like this, its value will be undetermined (null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

```
// Class Declaration


public class Dog
{
	// Instance Variables
	String name;
	String breed;
	int age;
	String color;

	// Constructor Declaration of Class
	public Dog(String name, String breed,
				int age, String color)
	{
		this.name = name;
		this.breed = breed;
		this.age = age;
		this.color = color;
	}

	// method 1
	public String getName()
	{
		return name;
	}

	// method 2
	public String getBreed()
	{
		return breed;
	}
```

```
    // method 3
    public int getAge()
    {
        return age;
    }

    // method 4
    public String getColor()
    {
        return color;
    }

    @Override
    public String toString()
    {
        return("Hi my name is "+ this.getName()+
            ".\nMy breed,age and color are " +
            this.getBreed()+"," + this.getAge()+
            ","+ this.getColor());
    }

    public static void main(String[] args)
    {
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");
        System.out.println(tuffy.toString());
    }
}
```