Garbage Collection

→ Process by which Java programs perform automatic memory management

→ Java programs compile to bytecode that can be run on a JVM

→ When Java program runs on JVM objects are created on the heap, which is a portion of memory dedicated to the program

→ Some objects are no longer needed so the garbage collector finds these unused objects and deletes them to free up memory.

→ Java garbage collection is an automatic process.

→ Programmer does not need to explicitly mark object to be deleted. Garbage collection implementation lives in the JVM.

eg:- Oracle Hot Spot.

Advantages

↳ ① Makes Java more efficient because garbage collector removes the unreferenced objects from heap memory.

② Automatically done by garbage collector (Part of JVM)

(2)

Object can be unreferenced by
(1) Nulling the reference
(2) Assigning a reference to another
(3) Anonymous object.

(1) Employee e = new Employee(); // Nulling
e = null;

(2) Employee e1 = new Employee(); // Reference to another
Employee e2 = new Employee();
e1 = e2;

(3) new Employee(); // Anonymous

→ finalize() method
↳ method is invoked each time before the object is garbage collected. Method can be used to perform cleanup processing. Defined in Object class as

```
protected void finalize()
{

}
```

→ gc() method
↳ used to invoke the garbage collector to perform cleanup processing. Found in System and Runtime classes. public static void gc() { }

* Garbage collection is performed by daemon thread called Garbage Collector (GC). This thread calls the finalize() method before object is garbaage collected.

```
public class TestGarbage1
{
  public void finalize()
  {
    System.out.println("object is garbage collected");
  }
  public static void main(String args[])
  {
    TestGarbage1  s1 = new TestGarbage1();
    TestGarbage2 s2 = new TestGarbage1();
    s1 = null;
    s2 = null;
    System.gc();
  }
}
```

O/P object is garbage collected
object is garbage collected

Static keyword

→ Used for memory management
→ Apply the keyword for variables, methods, blocks and nested class.
→ Belongs to class and not instance of the class.
→ Static can be
① variable ② Method ③ Block ④ Nested class

I] Java static variable

↳ keyword - static
↳ used to refer common property of all objects
↳ gets memory only once in the class area at the time of class loading
↳ makes program memory efficient (saves)

```
class Student
{
    int rollno;
    String name;
    String college = "MCET";
}
```

→ suppose there are 200 students in the college then all instance data members will get memory each time when the object is created. ~~All students have~~ But here college is a common property of all objects and if we make it static, then this field will get memory only once.

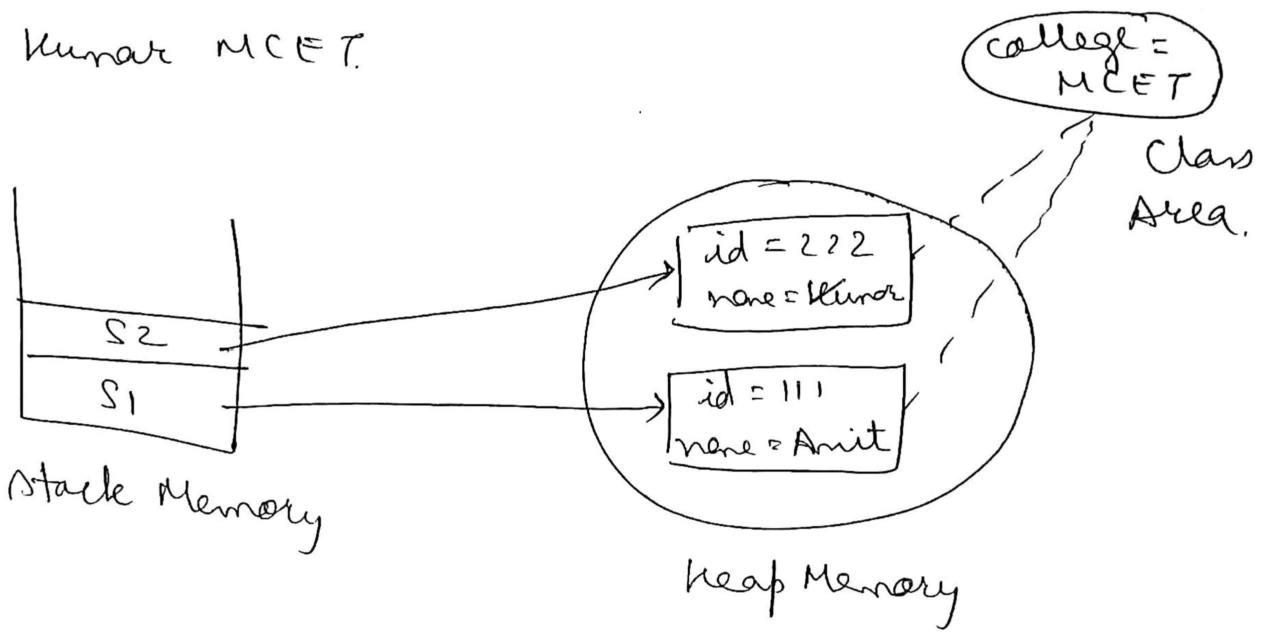
☆ Java static property is shared to all objects

```
class Student
{
  int rollno;
  String name;
  static String college = "MCET";
  Student (int r, String n) {          // constructor
    rollno = r;
    name = n.
  }
  // method to display the values.
  void display()
  {
    S.o.pln( rollno+"    " + name+ "    " + college);
  }
}

// Test class to show the values of objects
public class TestStaticVariable
{
    public static void main(String args[])
    {
       Student s1 = new Student(111, "Amit");
       Student s2 = new Student(222, "Kumar")
       s1.display();
       s2.display();
    }
}
```

If want to change the value of college
Student.college = "Stanley";

O/p

111 Amit MCET
222 Kumar MCET.



→ Example to show the difference of static variable

```
// Program to demonstrate use of an
// instance variable which get
// memory each time when we create
// an object
class Counter
{
  int count=0;
Counter()
{
  count++;
 System.out.println(count);
}
 public static void main(String args[])
 {
   Counter c1=new Counter();
   Counter c2 = new Counter();
   Counter c3 = new Counter();
 }
}
```

O/p 1
1
1

```
// using static variable
class Counter
{
static int count=0 // gets memory
                   only once (retain it
Counter()
{
   count++;
  System.out.println(count);
}
public static void main(String args)
{
   Counter c1 = new Counter(
   Counter c2 = new Counter(
   Counter c3 = new Counter(
 }
}
```

O/p :- 1
2
3

Java static method

→ static keyword with any method it is known as static method

① static method belongs to class and no to object of a class.

② Can be invoked without the need of creating an instance of a class.

③ Can access static data member & can change value of it

```
class student
{
  int rollno;
  String name;
  static string college = "MCET";
  static void change()
  {
    college = "Stanley";
  }
  student (int r, string n)
  {
    rollno = r;
    name = n;
  }
  void display()
  {
    system.out.println(rollno+"  "+name+"  "+coll
  }
}
```

```
public class Test static Method
{
   public static void main (string args[])
   {
      Student.change(); // calling method
      Student s1= new Student(111,"ABC");
      Student s2= new Student (222,"DEF");
      Student s3= new Student(333,"GHI");
      s1.display();
      s2.display();
      s3.display();
   }
}
```

O/P	111	ABC	Stanley
	222		
	333		

→ Restriction for static method

1. Can not use non static data member or call non static method directly
2. this and super keyword cannot be used in static

```
class A
{
   int a=40;
   p.s. void main(string args[])
   {
      system.out.println(a);
   }
}
```

O/P Compile Time Error

Java main method static

↳ Because the object is not required to call a static method. If its non static then JVM creates an object first and then call main() method that leads to extra memory allocation

Java static block

↳ used to initialize static data member

↳ executed before main method at time of class loading.

```
class A2
{
  static
  {
    System.out.println("static block invoked");
  }
  p.s. void main(String args[])
  {
    S.O.pln("hello main");
  }
}
```

O/P → Both lines will get printed

Super Keyword in Java

The super keyword in java is a reference variable that is used to refer parent class objects. The keyword "super" came into the picture with the concept of Inheritance. It is majorly used in the following contexts:

1. Use of super with variables: This scenario occurs when a derived class and base class has same data members. In that case there is a possibility of ambiguity for the JVM. We can understand it more clearly using this code snippet:

```
/* Base class vehicle */
class Vehicle
{
   int maxSpeed = 120;
}

/* sub class Car extending vehicle */
class Car extends Vehicle
{
   int maxSpeed = 180;

   void display()
   {
      /* print maxSpeed of base class (vehicle) */
      System.out.println("Maximum Speed: " + super.maxSpeed);
   }
}

/* Driver program to test */
class Test
{
   public static void main(String[] args)
   {
      Car small = new Car();
      small.display();
   }
}
```

Output:
Maximum Speed: 120

In the above example, both base class and subclass have a member maxSpeed. We could access maxSpeed of base class in sublcass using super keyword.

2. Use of super with methods: This is used when we want to call parent class method. So whenever a parent and child class have same named methods then to resolve ambiguity we use super keyword. This code snippet helps to understand the said usage of super keyword.

```
/* Base class Person */
class Person
{
   void message()
   {
      System.out.println("This is person class");
   }
}

/* Subclass Student */
class Student extends Person
{
```

```
void message()
   {
      System.out.println("This is student class");
   }

   // Note that display() is only in Student class
   void display()
   {
      // will invoke or call current class message() method
      message();

      // will invoke or call parent class message() method
      super.message();
   }
}

/* Driver program to test */
class Test
{
   public static void main(String args[])
   {
      Student s = new Student();

      // calling display() of Student
      s.display();
   }
}
```

Output:
This is student class
This is person class

In the above example, we have seen that if we only call method message() then, the current class message() is invoked but with the use of super keyword, message() of superclass could also be invoked.

3. Use of super with constructors: super keyword can also be used to access the parent class constructor. One more important thing is that, ''super' can call both parametric as well as non parametric constructors depending upon the situation. Following is the code snippet to explain the above concept:

```
/* superclass Person */
class Person
{
   Person()
   {
      System.out.println("Person class Constructor");
   }
}

/* subclass Student extending the Person class */
class Student extends Person
{
   Student()
   {
      // invoke or call parent class constructor
      super();

      System.out.println("Student class Constructor");
```

```
}
}

/* Driver program to test*/
class Test
{
   public static void main(String[] args)
   {
      Student s = new Student();
   }
}
```

Output:

Person class Constructor
Student class Constructor

In the above example we have called the superclass constructor using keyword 'super' via subclass constructor.

Call to super() must be first statement in Derived(Student) Class constructor.
If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass. If the superclass does not have a no-argument constructor, you will get a compile-time error. Object does have such a constructor, so if Object is the only superclass, there is no problem.
If a subclass constructor invokes a constructor of its superclass, either explicitly or implicitly, you might think that a whole chain of constructors called, all the way back to the constructor of Object. This, in fact, is the case. It is called constructor chaining..

Difference between super and super() in Java with Examples

super :The super keyword in java is a reference variable that is used to refer parent class objects. The keyword "super" came into the picture with the concept of Inheritance. Basically this form of super is used to initialize superclass variables when there is no constructor present in superclass. On the other hand, it is generally used to access the specific variable of a superclass.

```
// Java code to demonstrate super keyword
 /* Base class vehicle */
class Vehicle {
   int maxSpeed = 120;
}
/* sub class Car extending vehicle */
class Car extends Vehicle {
   int maxSpeed = 180;
   void display()
   {
      /* print maxSpeed of base class (vehicle) */
      System.out.println("Maximum Speed: "
                    + super.maxSpeed);
   }
}
/* Driver program to test */
class Test {
   public static void main(String[] args)
   {
      Car small = new Car();
      small.display();
   }
}
```

Output:
Maximum Speed : 120

super()

The super keyword can also be used to access the parent class constructor by adding '()' after it, i.e. super(). One more important thing is that 'super()' can call both parametric as well as non-parametric constructors depending upon the situation

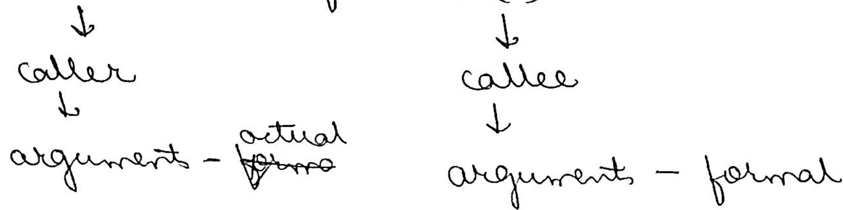
```
// Java code to demonstrate super()
 /* superclass Person */
class Person {
   Person()
   {
      System.out.println("Person class Constructor");
   }
}
 /* subclass Student extending the Person class */
class Student extends Person {
   Student()
   {
      // invoke or call parent class constructor
      super();

      System.out.println("Student class Constructor");
   }
}
 /* Driver program to test*/
class Test {
   public static void main(String[] args)
   {
      Student s = new Student();
   }
}
```

Output:
Person class Constructor
Student class Constructor

SUPER	SUPER()
The super keyword in Java is a reference variable that is used to refer parent class objects.	The super() in Java is a reference variable that is used to refer parent class constructors.
super can be used to call parent class' variables and methods.	super() can be used to call parent class' constructors only.
The variables and methods to be called through super keywordd can be done at any time,	Call to super() must be first statement in Derived(Student) Class constructor.
If one does not explicitly invoke a superclass variables or methods, by using super keyword, then nothing happens	If a constructor does not explicitly invoke a superclass constructor by using super(), the Java compiler automatically inserts a call to the no-argument constructor of the superclass.

Parameter Passing in Java

→ Different ways in which parameter data can be passed into and out of methods and functions



→ Types of parameter

① Formal parameter → a variable and its type as they appear in prototype of the funct or method

funct_name (datatype var_name);

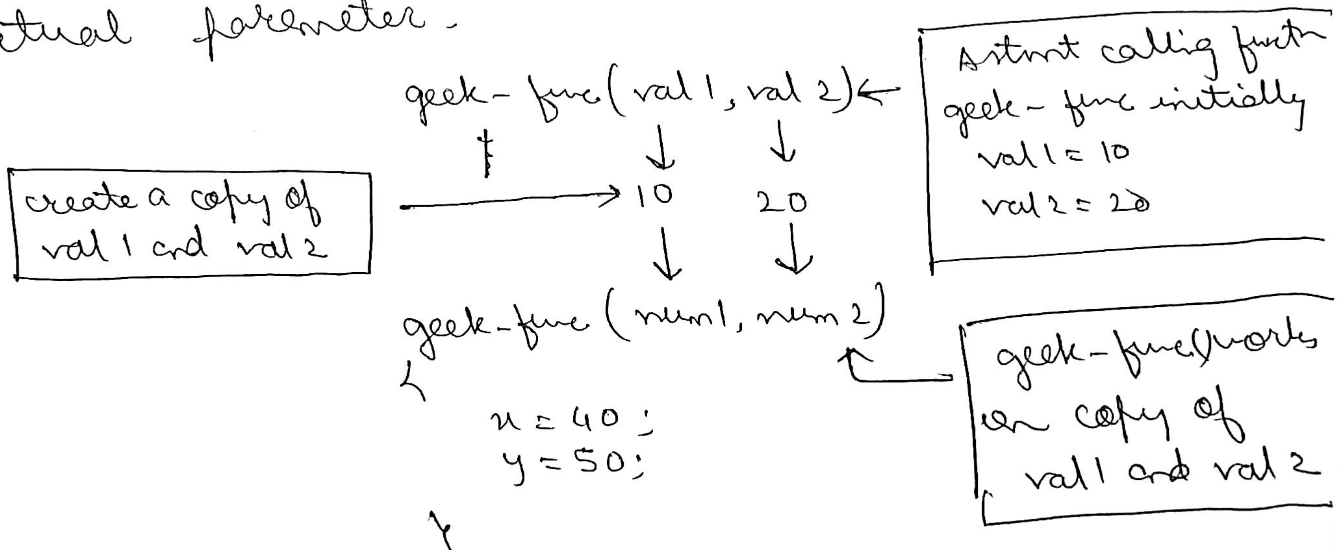
② Actual parameter → variable or expression corresponding to a formal parameter that appears in the funct or method call in calling environment

funct_name (var_name(s));

→ Parameter Passing

① Pass by value

↳ changes made to formal parameter do not get transmitted back to caller. Any modificatn to formal parameter variable inside the called funct or method will affect only separate storage locatn and does not make changes to actual parameter.



```
class CallbyValue
{
  public static void Example(int x, int y)
  {
     x++;
     y++;
  }
}
public class Call1
{
   public static void main(String args[])
   {
      int a=10;
      int b= 20;
   CallbyValue ob= new CallbyValue();
   System.out.println("Value of a: "+a +
                       " & b: "+b);
    ob.Example(a,b);
   System.out.println("Value of a: "+a+ "&b: "+
                                          b);
   }
}
```

O/P :- Value of a : 10 & b : 20
Value of a : 10 & b : 20.

Problem
① Inefficiency in storage location
② For objects & arrays, copy semantics are costly.

(2) Call by reference (aliasing)

↳ Changes made to formal parameters gets transmitted back to caller through parameter passing

↳ Any changes to formal parameter are reflected in actual parameter in calling environment as formal parameter receives a reference (pointer) to actual data.

↳ efficient for both time & space.

```
class A
{
    int x;
    void func(A obj)
    {
        obj.x = 20;
    }
}
class B
{
    p.s.v.main(String args[])
    {
        A a = new A()
        a.x = 10;
        a.func(a);
    }
}
```

obj

2008

4016

a.x

~~10~~ 20

2008.

```
class CallbyReference
{
    int a, b;
    CallbyReference(int x, int y)
    {
        a = x;
        b = y;
    }
    void changeValue(CallbyReference obj)
    {
        obj.a += 10;
        obj.b += 20;
    }
}
class Call
{
    public static void main(string args[])
    {
        CallbyReference r1 = new CallbyReference(10, 20);
        System.out.println("Value of a: " + r1.a
                + " & b: " + r1.b);

        r1.changeValue(r1);
        System.out.println("Value of a: " + r1.a + " & b: " +
                r1.b

    }
}
```

O/P: Value of a: 10 & b: 20
Value of a: 20 & b: 40

Final keyword

↳ if we don't want to override existing attribute values then declare attribute as final

```
~~public~~ class myClass
{
    final int x=10;
    final double PI=3.14;
    public static void main(String args[])
    {
        myClass o1= new myClass();
        o1.x= 50;
        o1.PI=25;

        system.out.println(o1.x);
        System.out.println(o1.PI);
    }
}
```

O/P :- cannot assign a value to final variable x (Error)

—||—

String Handling

Page. No.

→ Sequence of characters.

→ Java, string is an object representing sequence of characters.

→ Object of a class and no automatic appending of null character by the system.

→ In Java, three classes that can create string

① class String
② class String Buffer
③ class String Builder

} Part of java.lang package

eg:- ① String str1 = "abcd" → sequence of characters

String ↓ class String; str1 ↓ string object

char s1[] = {'a', 'b', 'c', 'd'};

② Object of class String can be created by new operator

string str = new String("abcd");

str ↓ object of class string; new → new operator for creating object of class string

② In other two classes objects are created by

StringBuffer bufstr = new StringBuffer("abcd");
StringBuilder buildstr = new StringBuilder("abcd");

Storage of Strings

→ Objects of string have special storage facility which is not available to objects of other two string classes

→ Memory allocated to Java program is divided into

① Stack
② Heap

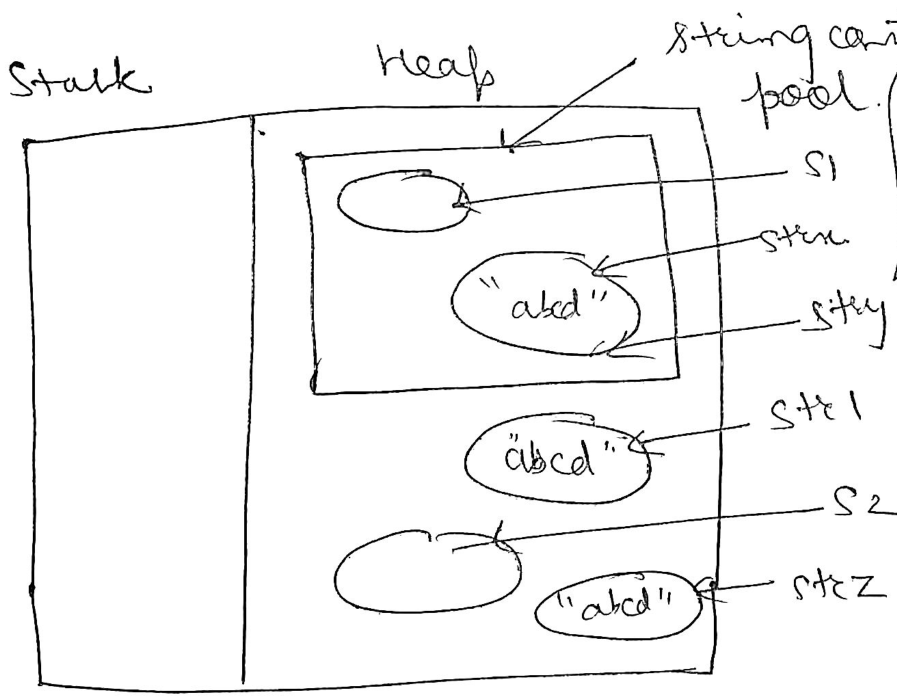
→ Variables stored on heap, program stored on stack.

→ Memory management segment in heap is called as String constant pool.

→ Ways of creating String class objects.

String strx = "abcd";
↳ stored in String constant pool

String str2 = new String("abcd");
↳ defined using new operator stored on heap memory.



String stry = "abcd"
Here no duplicate object is created as reference to the object is already there in the pool. so strx assign to stry. But outside pool if new object is created & even the object exist then also new object is created.

String constant pool stores only unique strings

// Program to illustrate storage of string


```
public class StringTest
{
 public static void main(String args[])
 {
    String s1 = "abcd";
    String s2 = "abcd";
    String s3 = new String("abcd");
    String s4 = new String("abcd");
    String s5 = new String();
    String s6 = "  ";
 S.o.pln("Are refs of string s1 and s2 same?" + (s1 == s2));
 S.o.pln("Are ref of s1 and s3 same?" + (s1 == s3));
 S.o.pln("Are ref of s4 and s3 same?" + (s4 == s3));
 S.o.pln("Are ref of s5 and s6 same?" + (s5 == s6));
 S.o.pln("Are s1 and s3 equal?" + s1.equals(s3));
 }
}
```

o/p   true   false   false   false   true.