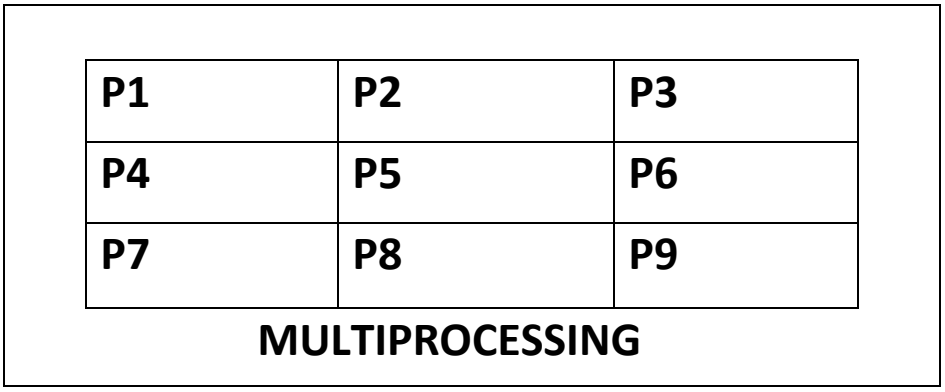
MULTITHREADING

MUSIC PLAYER TASK (https://www.dropbox.com/s/k8zvzpw9h1ijsrh/MUSIC_PLAYER.rar?dl=0)

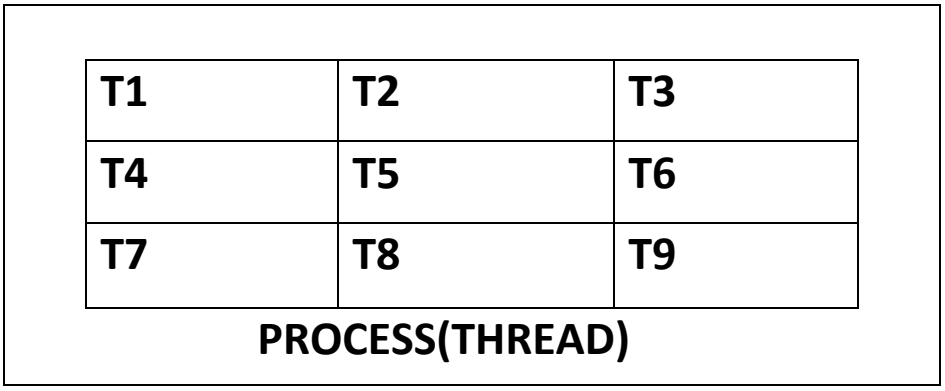
- TO UNDERSTAND THE CONCEPT OF MULTITHREADING WE FIRST NEED TO UNDERSTAND THE CONCEPT OF MULTITASKING AND MULTIPROCESSING

MULTITASKING➔

- PERFORMING OR EXECUTING MORE THAN ONE TASK AT THE SAME TIME IS CALLED AS MULTITASKING
- TASK IS THE END GOAL THAT HAS TO BE ACHIVED
- TASK CAN BE CALLED AS A COMBINATION OF MORE THAN ONE PROCESS



- PERFORMING OR EXECUTING MORE THAN ONE PROCESS AT A SAMETIME IS CALLED AS **MULTIPROCESSING**
- THE STEPS INVOLVED IN COMPLETION OF THE TASK ARE CALLED AS **PROCESS**
- A PROCESS IS A COMBINATION OF MORE THAN ONE THREAD



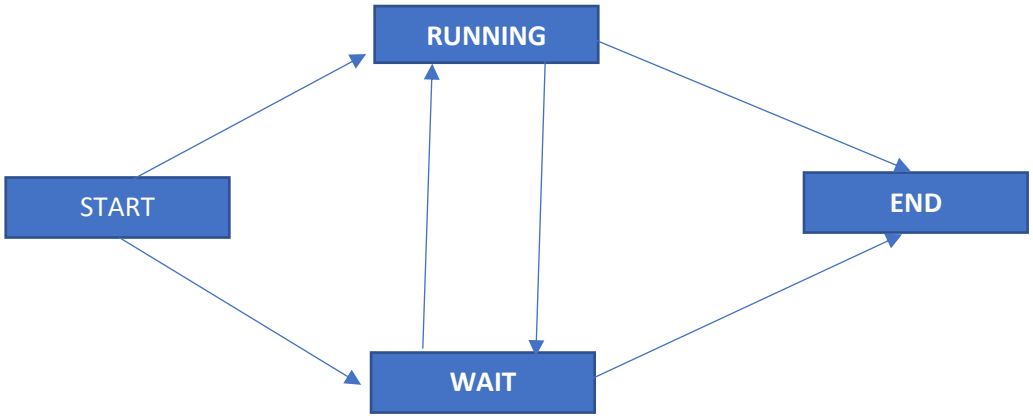
- THREAD IS THE SMALLEST UNIT OF PROCESS
- A THREAD CAN ALSO BE CALLED AS **LIGHT WEIGHT PROCESS**

- EXECUTING OR IMPLEMENTING MORE THAN ONE THREAD AT THE SAMETIME US CALLED AS **MULTITHREADING**

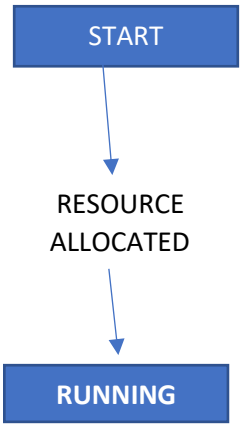
THREAD IN JAVA➔

- IN JAVA THREAD IS SPECIAL CLASS WHICH CAN BE CREATED IN TWO WAYS
  - **I.** EXTENDING THREAD CLASS
  - **II.** IMPLEMENTING RUNNALBLE INTERFACE

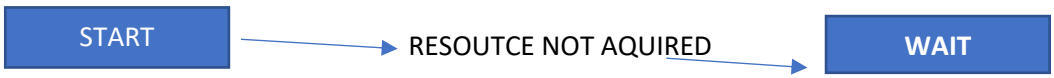
LIFE CYCLE OF A THREAD ➔



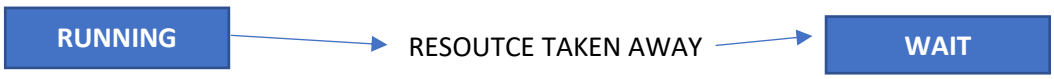
- **START TO RUNNING =**WHEN A THREAD IS INITIALIZED AND THE RESOURCES REQUIRED FOR ITS EXECUTION ARE ALLOCATED TO IT THEN THREAD MOVES FROM START PHASE TO THE RUNNING PHASE



- **START TO WAIT =**WHEN A THREAD IS INITIALIZED BUT THE RESOURCES REQUIRED FOR ITS EXECUTION ARE NOT ALLOCATED TO IT THEN THREAD MOVES FROM START PHASE TO THE WAIT PHASE

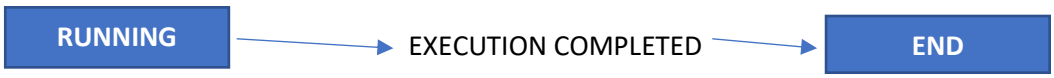


- **RUNNING TO WAIT =**IF THE THREAD IS IN RUNNING PHASE THE RESOURCES ALLOCATED TO IT ARE TAKEN AWAY THEN THREAD MOVES FROM RUNNING PHASE TO THE WAIT PHASE



- **WAIT TO RUNNING =**IF THE THREAD IS IN WAIT PHASE AND THE RESOURCES REQUIRED FOR ITS EXECUTION ARE ALLOCATED TO IT THEN THREAD MOVES FROM WAIT PHASE TO THE RUNNING PHASE

- **RUNNING TO END =**IF THE THREAD IS IN RUNNING PHASE AND COMPETES ITS EXECUTION THEN THREAD MOVES FROM RUNNING PHASE TO THE END PHASE



- **WAIT TO END =**IF THE THREAD IS IN WAITING FOR RESOURCE FOR LONG TIME THEN IT WILL MOVE FROM WAIT PHASE TO THE END PHASE

NOTE:ALL THE PHASES OF THE THREAD ARE REPRESENTED AS METHODS OF A THREAD
1ST WAY OF THREAD CREATION➔

Mythread1.java

```
public class MyThread extends thread {
	@Override
	public void run() {
	System.out.println("thread is running");
	}
}
```

Thread1.java

```
public class Threadmain {
	public static void main(String[] args) {
	MyThread1 mythread1 = new MyThread1();
	mythread1.run();
	}
}
```

START ( ) ; ➔

- IT IA A NONSTATIC METHOD PRESENT IN THE THREAD CLASS
- TO START A CLASS OBJECT AS A THREAD IT IS MANDATORY TO USE START METHOD
- THIS METHOD IMPLICTLY CALLS THE RUN METHOD FROM THE TARGET CLASS

RUN ( ) ; ➔

- THIS METHOD IS USED TO DEFINE THE BEHAVIOUR (EXECUTION LOGIC) OF THE THREAD

2ND WAY OF THREAD EXECUTION

Mythread.java

```
public class MyThread implements Runnable {
	@Override
	public void run() {
		System.out.println("thread is running");
	}
}
```

Threadmain.java

```
public class Threadmain {
	public static void main(String[] args) {
		MyThread1 mythread1 = new MyThread1();
		Thread thread=new Thread(mythread1);
		thread.start();
	}
}
```

- IN THIS CASE THERE IS NO RELATION BETWEEN MYTHREAD CLASS AND THREAD CLASS HENCE WE CANNOT ACCESS THE START METHOD WITH THE HELP OF MYTHREAD CLASS OBJECT
- THAT IS WHY ER NEED TO CREATE THE OBJECT FOR THREAD CLASS AND PASS THE OBJECT OF MYTHREAD CLASS AS THE ARGUMENT TO THE THREAD CLASS OVERLOADED CONSTRUCTOR BY

DOING THIS THE THREAD CLASS WILL IGNORE ITS OWN BEHAVIOUR AND ACCEPT THE BEHAVIOUR OF MYTHREAD CLASS

- EXPECTED OUTPUT

```
Mythread is now running
Mythread is now running
Mythread is now running
Mythread is now running
Mythread1 is now running
Mythread1 is now running
Mythread1 is now running
Mythread1 is now running
Mythread1 is now running
```

- THE ACTUAL OUTPUT MIGHT DIFFER FROM EXPECTED OUTPUT WHEN MULTIPLE THREADS ARE CALLED FOR EXECUTION SIMALTANEOUSLY IS THREAD WILL BE ALLOCATED WITH A SEPRATE STACK THE OUTPUT WILL DEPEND ON THE ORDERS OF EXECUTION OF THE STACKS THE CONTROL OF EXECUTION MIGHR BE GIVEN FROM ONE STACK TO ANOTHER AT ANY POINT OF TIME THIS DONE BY COMPONENT CALLED AS THREAD SCHELUDER

THREAD SCHELUDER➔

- THREAD SCHELUDER IS A COMPONENT IN MULTITHREADING THAT IS RESPONSIBLE TO MANAGE AND CONTROL THE EXECUTION OD MULTIPLE THREADS
- IT IS ALSO RESPONSIBLE TO CREATE DEDICATED FOR EACH THREAD AND TRANSFER THE CONTROL BETWEEN STACK BASED ON THREAD PROPERTIES

THREAD PROPERTIES➔

- EVERY THREAD IS CREATED WILL HAVE PROPERTIES OF ITS OWN THAT ARE AS FOLLOWS
    I. NAME
    II. PRIORITY
- THESE THREAD PROPERTIES CAN BE ACCESS AND MODIFIED WITH THE HELP OF ITS GETTERS AND SETTERS

MyThread1.java

```
public class MyThread1 extends Thread {
	@Override
	public void run() {
		System.out.println("thread1 is running" +this.getName());
		System.out.println("the priority of the thread is"
					+this.getPriority());
		}
```

```
}
```

ThreadMain.java

```
public class Threadmain {
	public static void main(String[] args) {
		MyThread1 mythread1 = new MyThread1();
		mythread1.setName("chomu");
		mythread1.setPriority(3);
		mythread1.start();
	}
}
```

GetName( ); ➔

- IT IS A NONSTATIC METHOD PRESENT IN THE THREAD CLASS.
- IT IS USED TO RETRIVE THE NAME OF THE THREAD.

SetName( ); ➔

- IT IS A NONSATAIC METHOD PRESENT IN THE THREAD CALSSS WHICH ACCEPTS A STRING ARGUMENT
- SetName(String args);
- IT IS USED TO MODIFY OR SET THE NAME OF THE THREAD

getPriority( ); ➔

- IT IS NONSTATIC METHOD IN THREAD CLASS
- IT IS USED TO RETRIVE PRIORITY OF THE THREAD CALSS

setPriority( ); ➔

- IT IS NONSTATIC METHOD PRESENT IN THREAD CLASS
- WHICH ACCEPTS AN INTEGER ARGUMENT
- SetPriority(int Priority) ;
- IT IS USED TO MODIFY OR SET THE PRIORITY OF THE THREAD

currentThread( ); ➔

- IT IS A STATIC METHOD PRESENT IN THE THREAD CLASS IT HELPS US TO POINT TOWARDS THE CURRENT OBJECT OF THE THREAD CLASS

MyThread.java

```
public class MyThread implements Runnable {
	@Override
	public void run() {
		System.out.println("thread is
				running"+Thread.currentThread().getName());
		System.out.println("the priority of thead is
				"+Thread.currentThread().getPriority());
	}
}
```

ThreadMain.java

```
public class Threadmain {
	public static void main(String[] args) {
		MyThread mythread = new MyThread();
		Thread thread=new Thread(mythread);
		thread.setName("mythread");
		thread.setPriority(2);
		thread.start();
	}
}
```

Stop( ); ➔

- IT IS NON STATIC METHOD PRESENT INSIDE THREAD CLASS
- IT IS USED TO MOVE THREAD FROM A RUNNING PHASE TO THE STOP PHASE
- WITH THE HELP OF THIS METHOD WE CAN STOP THE EXECUTION OF CURRENTLY EXECUTING THREAD FORCEFULLY
- **NOTE: THE STOP METHOD IS DEPRICATED (SOMETHING OLD OR NOT IN USE).**

Account.java

```
public class account {
	double account_balance;

   account(double balance){
		this.account_balance=balance;
   }
   public double check_balance() {
		return account_balance;
   }
   public void deposite(double amount) {
     System.out.println("depositing"+amount+"in account");
```

```
        System.out.println("current balance:"+check_balance());
    }
    public void withdraw(double amount) {
        System.out.println("withdraw "+ amount+" from account");
        System.out.println("current balance "+check_balance());
    }
}
```

Husband.java

```
public class husband extends Thread {
        account acc;
        public husband(account ac) {
                this.acc=ac;
        }
        public void run() {
                acc.deposite(1000);
                acc.withdraw(500);
        }
}
```

Wife.java

```
public class wife extends Thread {
        account acc;
        public wife(account ac) {
                this.acc=ac;
        }
        public void run() {
                acc.deposite(500);
                acc.withdraw(5000);
        }
}
```

AccountMain.java

```
public class accmain {
        public static void main(String[] args) {
                account ac = new account(10000);
                husband hu = new husband(ac);
                wife wi = new wife(ac);
                hu.start();
                wi.start();
        }
}
```

- IN MULTITHREADING ALL THE THREADS ARE UNAWARE ABOUT THE OPERATIONS PERFORMED ON SAME RESOURCE BY ALL THE OTHER THREADS WHICH MAY LEAD TO DATA INCONSISTANCY
- IN THIS CASE THE HUSBAND THREAD AND WIFE THREAD ARE WORKING ON THE SAME OBJECT ON THE ACCOUNT CLASS BUT BOTH THE THREADS ARE UNAWARE OF EACH OTHER OPERATIONS ON THE RESOURCE WHICH RESULT IN CONSISTANT FINAL ACCOUNT BALANCE

EXAMPLE TASK: (https://www.dropbox.com/s/qx24jjs4dq8j6az/task.rar?dl=0)

SYNCHRONIZATION

- IT IS A PROCEDURE OF ACHEVING DATA CONSISTINCY WHILE EXECUTING MULTIPLE THREADS THAT ARE WORKING OR OPERATING ON THE SAME RESOURCE
- TO ACHIVE SYNCHRONIZATION WE NEED TO MAKE THE USE OF SYNCHRONIZED KEYWORD
- IF SYNCHRONIZED KEYWORD IS USED ALONG WITH A RESOURCE THEN ONLY ONE THREAD IS ALLOWDED TO ACCESS IT AT A TIME
- IT CAN BE SAID THAT IF A RESOURCE IS SYNCHRONIZED THEN THE THREAD WHICH IS ACCESSING THAT RESOURCE WILL APPLY LOCK ON THAT RESOURCE

LOCKS IN MULTI THREADING➔

- IN MULTI THREADING THERE ARE TYPES OF LOCKS
  - **I.** OBJECT LOCK
  - **II.** CLASS LOCK

SHARED RESOURCE➔

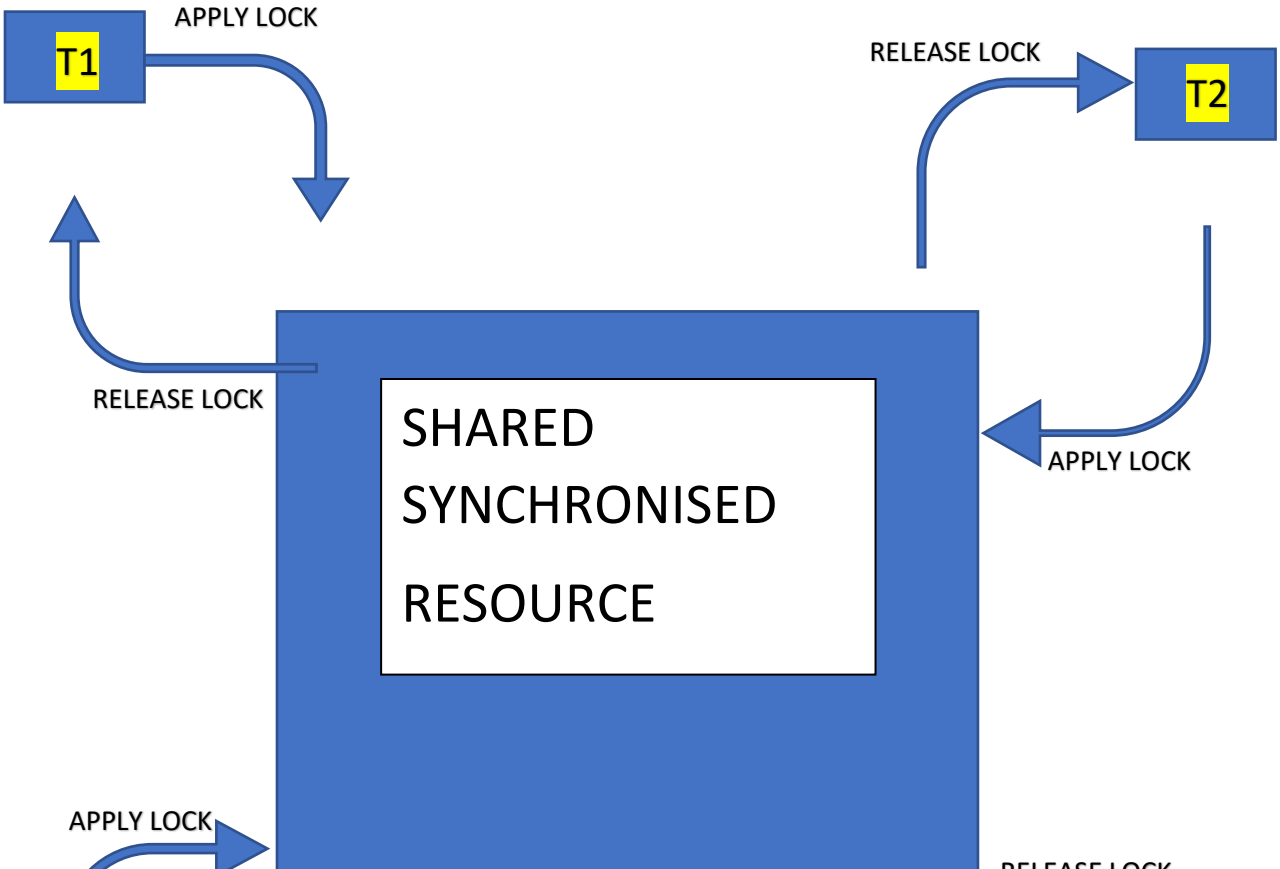
- THE RESOURCE WHICH IS ACCESSED NY MORE THAN ONE THREAD IS CALLED AS SHARED RESOURCE

OBJECT LOCK➔

- THE SYNCHRONISED SHARED RESOURCE IS NON STATIC MEMBER THEN THE LOCK APPLIED ON IT WILL BE OBJECT LOCK

CLASS LOCK➔

- THE SHARED SYNCHRONISED MEMBER ARE ONE OF STATIC TYPE THEN IS CALLED AS CLASS LOCK



AccountMain.java

```
public class account {
	double account_balance;

	account(double balance) {
		this.account_balance = balance;
	}
	public double check_balance() {
		return account_balance;
	}
	public synchronized void deposite(double amount) {
		System.out.println("depositing " + amount + " in account");
		this.account_balance += amount;
		System.out.println("current balance: " + check_balance());
	}
	public synchronized void withdraw(double amount) {
		System.out.println("withdraw " + amount + " from account");
		this.account_balance -= amount;
		System.out.println("current balance " + check_balance());
	}
}
```

WAIT( ) ; ➔

- IT IS A NON-STATIC METHOD PRESENT IN THE THREAD CLASS IT IS USED TO PAUSE THE EXECUTION OF THE CURRENTLY EXECUTING THREAD AND MOVE IT FROM RUNNING PHASE TO WAIT PHASE FORCEFULLY

NOTIFY( ) ; ➔

- IT IS A NON-STATIC METHOD PRESENT IN THE THREAD CLASS IT IS USED TO CALL THE WAITING THREAD TO RESUME ITS EXECUTION IT MOVES THE THREAD FROM WAITING PHASE TO RUNNING PHASE FORCEFULLY

NOTIFYALL( ) ; ➔

- IT IS A NON-STATIC METHOD PRESENT IN THE THREAD CLASS IT IS USED TO CALL THE WAITING THREADS TO RESUME THEIR EXECUTION IT MOVES THE THREADS FROM WAITING PHASE TO RUNNING PHASE FORCEFULLY

Pizza.java

```
public class pizza {
	private int no_of_Pizza;

	public synchronized void orderPizza(int orderedPizza) {
		System.out.println("Ordering " + orderedPizza + " pizzas");

		if (orderedPizza > no_of_Pizza) {
		System.out.println(orderedPizza + " pizzas not available");
		System.out.println("Please wait..!!");

		try {
			this.wait();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

		no_of_Pizza -= orderedPizza;
		System.out.println("Ordered " + orderedPizza + "pizzas
		successfully.");
	   }
	}
	public synchronized void makePizza(int made_Pizza) {

		System.out.println("Making " + made_Pizza + " pizzas");

		no_of_Pizza += made_Pizza;

		System.out.println(no_of_Pizza + " pizzas available");
		this.notify();
	}
}
```

friends.java

```
public class friends extends Thread {
	private pizza pizza;

	public friends(pizza pizza) {
		this.pizza = pizza;
	}
	@Override
	public void run() {
		pizza.orderPizza(5);
	}
}
```

Pizzahut.java

```
public class Pizzahut extends Thread {
     private pizza pizza;
     public Pizzahut (pizza pizza) {
          this.pizza = pizza;
     }
     @Override
     public void run() {
          pizza.makePizza(5);
     I}
}
```

Pizzamain.java

```
public class PIZZAMAIN {
     public static void main(String[] args) {
          pizza pizza = new pizza();
          friends friends = new friends(pizza);
          Pizzahut pizzaHut = new Pizzahut(pizza);
          friends.start();
          pizzaHut.start();
     }
}
```

NOTE: A THREAD WHICH HAS BEEN PUT TO THE WAIT STATE FORCEFULLY WILL NOT RESUME ITS EXECUTION ON ITS OWN

SLEEP( ) ; ➔

- IT IS STATIC METHOD PRESENT IN THE THREAD CLASS
- IT IS SIMILAR TO THE WAIT METHOD I.E. IT IS USED TO PAUSE THE EXECUTION OF CURRENTLY EXECUTING THREAD
- UNLIKE WAIT METHOD THE SLEEP METHOD DOES NOT NEED THE NOTIFY METHOD TO CALL THE THREAD TO RESUME ITS EXECUTION IT ACCEPTS A LONG VARIABLE AS AN ARGUMENT THAT REPERESENTS THE TIME DURATION IN MILISECONDS.

Sleepdemo.java

```
public class SLEEP {
      public static void main(String[] args) {
         String message = "This is the magic of sleep()";
         char[] messageArray = message.toCharArray();
         for (int i = 0; i < message.length(); i++) {
         System.out.print(messageArray[i]);
         try {
          Thread.sleep(500);
          } catch (InterruptedException e) {
          e.printStackTrace();
          }
       }
    }
}
```

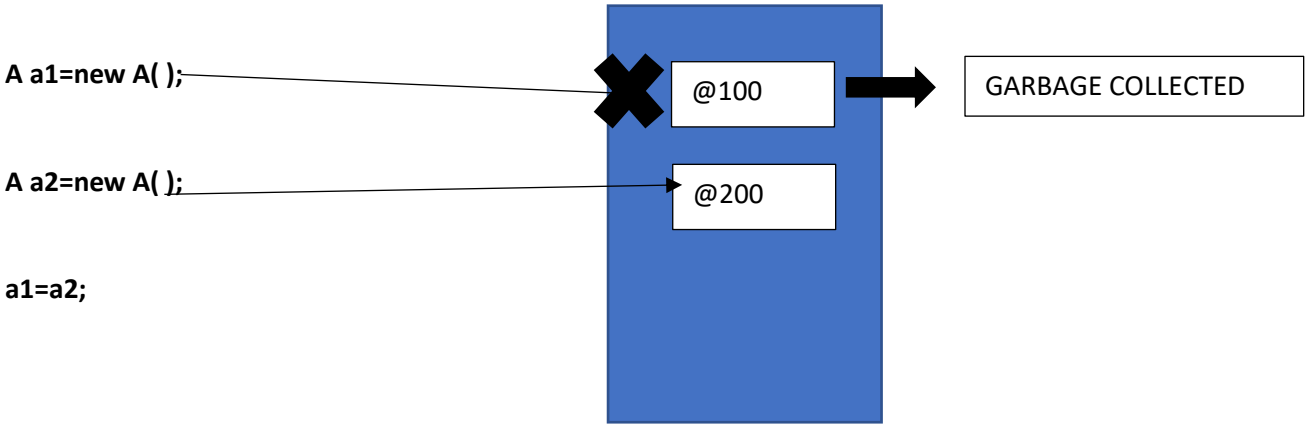
DEMON THREADS➔

- THE THREADS CREATED USING THREAD CLASS ARE RUNNABLE INTERFACE ARE KNOWN AS USER DEFINED THREADS SUCH THREADS ANEED TO BE CALLED FOR EXECUTION
- IN JAVA THERE ARE SOME PREDEFINED THREADS THAT ARE CALLED FOR EXECUTION IMPLICTLY BY THE **JVM** SUCH THREADS ARE KNOWN AS DEMON THREADS
- DEMON THREADS ARE OF LOW PRIORITY
- THE MOST IMPORTANT DEMON THREAD IS GRABAGE COLLECTION

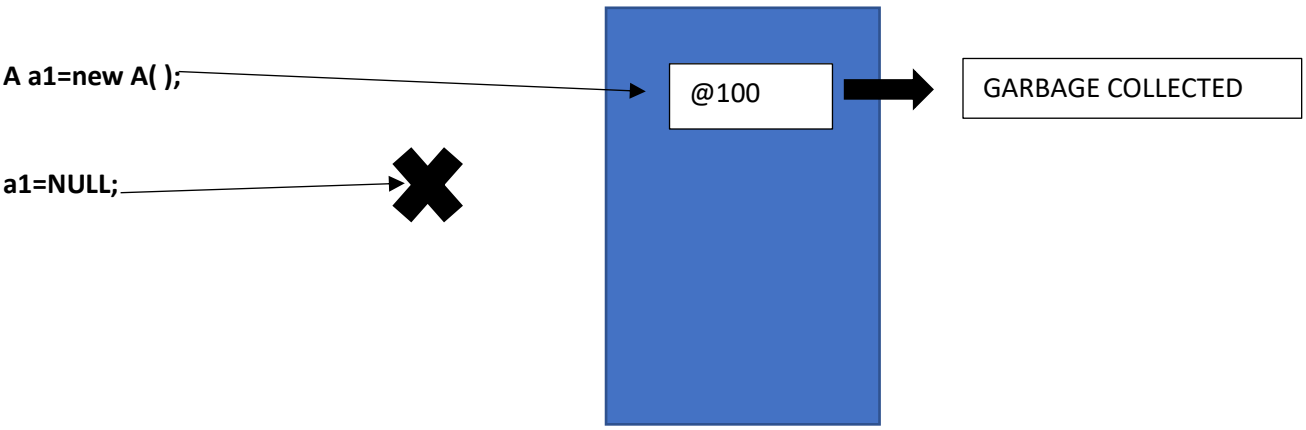
GARBAGE COLLECTION➔

- THE GARBAGE COLLECTION DEMON THREAD IS RESPONSIBLE TO REMOVE UNWANTED OBJECTS FROM THE MEMORY
- IT IS CALLED IMPLICTILY BY THE **JVM** WHENEVER REQUIRED
- IT IS CALLED FOR EXECUTION IN TWO SITUATIONS
    I. AFTER DEREFERING AN OBJECT
    II. AFTYER NULLIFIYING AN OBJECT

DEREFERING ➔



NULLIFYING ➔



ADVANTAGES OF GARBAGE COLLECTION➔

- IT MAKES JAVA EFFICIENT IN MEMORY MANAGEMENT
- BECAUSE OF GARBAGE COLLECTION DEMON THREAD THE PROGRAMMER DOES NOT NEED TO TAKE CARE OF MEMORY MANAGEMENT
  NOTE:THE USED DEFINED THREADS CAN BE MADE TO BEHAVE ASA DEMON THREAD WITH THE HELP OF SETDEMON ( ) ;
- IT CAN BE DONE BY PASSING "TRUE" AS AGRUMENT TO THIS METHOD

Stop() class/Interface - Thread (java.lang.Thread)
Modifier- public
Return type-void
Header - 1 public final void stop(),
2 public final vold stop
Argument → The Throwable object to be thrown
member-non-static
Exception - Security Exception - This exception comer When the current thread can't change this thread;

notify()
class / Interface - Object (java.lang.Object)
Modifier- public
Return type - void
Header - public final void notify ( )
Argument- None.
Member - Non-static
Exception-illegalMonitorStateException

notifyALL()
Class / Interface - Object (Java.Lang.Object)

Modifier - public
Return type- wold
Header- public final void notify All()
Argument - None
Member - non-static
Exception-illegalMonitorStateException

sleep()
class/Interface- Java lang package.
modifier- public
Return type- void
Header -1-public static vold sleep (long milliseconds)
2-public static void sleep (long milliseconds,int nanoseconds)
Argument-1-millisecond
2-nanoseconds(0 to 999999)
Member non-static
Exception - None

Current Thread ()
class/Interface :- Theuad (java.lang. Thread)
modifier- public
Returntype-it return currently executing Thread.
Header- public static thread current thread ()
Argument - None
Member - non-static
Exception - None.

start ()-
class / Interface :- Thread (Java.lang.Thread)
Modifier :- public
Retuin type- void
Header - public void start()
Argument -None
Member - non-static
Exception-none

wait()
class / Interface - Object (java.lang.Object)
Modifier - public
Return type-void
Header - Public final void wait()
Argument :- long

Member - non-static
Exception-interruptedException

run()
class / Interface - Thread (java.lang.Thread)
Modifier - public
Return type- vold
Header - public void run()
Avigument - None
Member - non-static.
Exception - None.

// **MULTITHREADING ENDS //**