OBJECT ORIENTED PROGRAMMING USING

Java

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

JAVA THREADS :

Threads allows a program to operate more efficiently by doing multiple things at the same time.
Threads can be used to perform the complicated tasks in the background without interrupting the main program.

=> How to create a Thread in Java?

There are two ways:
1) Extending Thread class.
2) Implementing a Runnable interface.

1) Extending the Thread class:

We can run Threads in Java by using Thread class, which provides constructors and methods for creating and performing operations on a Thread, which extends a Thread class that can implement Runnable interface.

```
import java.util.*;
public class GFG extends Thread
{
    public void run()
  { System.out.println("Thread started");
  }
    public static void main(String []args)
  {   GFG g = new GFG();

        g.start(); // Involving Thread
} }
```

It can be created by extending the Thread class and overriding its run().

```
public class Main extends Thread
{ public void run()
  { System.out.println("The code is
                              running in a thread");
  }
}
```

=> Create thread by using Runnable interface :

```
import java.util.*;
public class GFG implements Runnable
{
  public void run()
  {
   System.out.println("Thread is Running
                          Successfully");
  }
  public static void main (String[] args)
  { GFG g = new GFG();

    Thread t = new Thread(g);
    t.start();
  }
}
```

Another way to create a Thread is to implement the Runnable interface.

```
public class Main implements Runnable
{ public void run()
   {
     System.out.println("This code is
                        running in a thread");
   }
}
```

=> Create Thread in Java using Thread (String name):

```
import java.util.*;
public class GFG {
 public static void main (String []args)
 {
   Thread t = new Thread ("Hello Geeks");

   t.start();

   String s = t.getName();
   System.out.println (s);
 }
}
```

```
Hello Geeks
```

=> Create Thread Object by using Thread (Runnable r , String name):

```
import java.util.*;
public class GFG implements Runnable
{
```

```
public void run()
{
 System.out.println("Thread is created
                     and running successfully");
}
public static void main(String []args)
{
  Runnable r = new GFG();
  Thread t = new Thread(r, "My Thread");
   // Thread object started
   t.start();
   // getting the thread
   String str = t.getName();
   System.out.println(str);
 }
}
```

```
My Thread
Thread is created and running successfully.
```

Running Threads:

If the class extends the Thread class, the thread can be run by creating an instance of the class and call its start() method.

```
public class Main extends Thread
{
  public void run()
  {
   System.out.println("This code is running
                        in a thread");
  }
public static void main (String[] args)
{
   Main thread = new Main();
   thread.start();
  System.out.println ("This code is
                    outside of thread");
 }
}
```

```
This code is outside of the Thread
This code is running in a thread.
```

If the class implements the Runnable interface, the thread can be run by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method.

```
public class Main implements Runnable
{
  public static void main (String[]args)
 {
   Main obj = new Main();
   Thread thread = new Thread(obj);

   thread.start();
  System.out.println ("This code is
                    outside of thread");
 }
  public void run ()
 {
  System.out.println ("This code is
                    running in a thread");
 }
}
```

Concurrency Problems:

Because threads run at the same time as other parts of the program, there's no way to know in which order the code will run.
When the threads and main program are reading and writing the same variables, the values are unpredictable.
The problems that result from this are called concurrency problems.

A code where the value of the variable amount is unpredictable.

```
public class Main extends Thread
{
   public static int amount = 0;

   public static void main (String[] args)
 {   Main thread = new Main();

     thread.start();
     System.out.println (amount);

     amount++;
     System.out.println(amount);
 }
  public void run()
 {  amount++;
 }
} }    0
       2
```

To avoid concurrency problems, it is best to share as few attributes between the threads as possible.
If attributes need to be shared, one possible solution is to use the isAlive() method of thread to check whether the thread has finished running before using any attributes that thread can change.

Example: Use isAlive() to prevent concurrency problems.

```
public class Main extends Thread
{ public static int amount = 0;

  public static void main (String [] args)
  {   Main thread = new Main();
      thread.start();
      ~~Sys~~ // Wait for the thread to finish
      while (thread.isAlive())
      {
        System.out.println ("Waiting.... ");
      }

   //Update amount and print its value
   System.out.println ("Main:" + amount);
      amount++;
   System.out.println ("Main:" + amount);
  }
  public void run() {
    amount++;
```

```
Waiting...
Main : 1
```