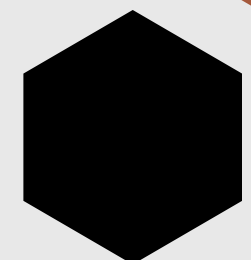
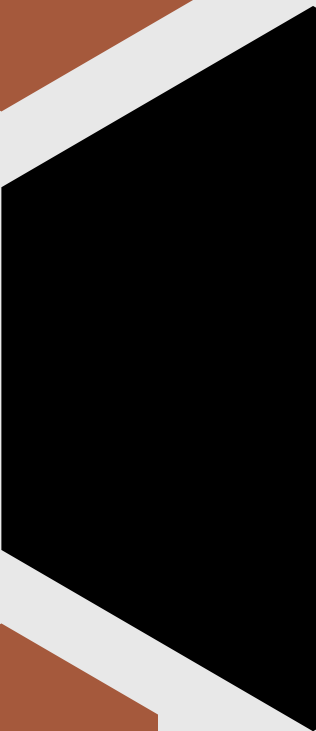
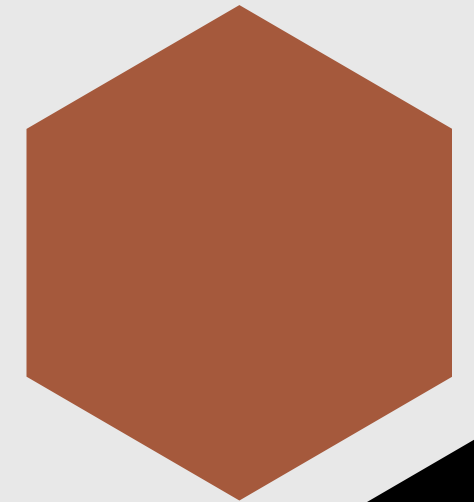
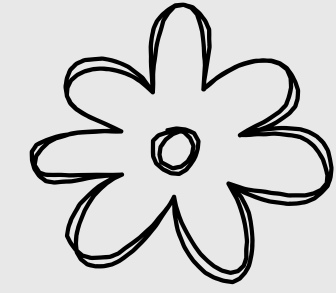


MULTI-THREADING



ARAVINDHAN
20384105



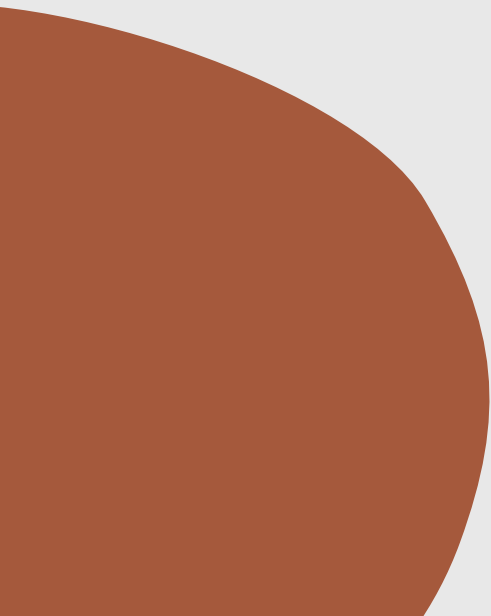
MULTI-PROCESSING

Multiprocessing is used to create a more reliable system



MULTI-THREADING

Multithreading is used to create threads that run parallel to each other



CREATING AND RUNNING A THREAD

```
class Multi extends Thread
{
    public void run()
    {
        System.out.println("thread is running...");
    }

    public static void main(String args[])
    {
        Multi t1=new Multi();
        t1.start();
    }
}
```

```
thread is running...
```

The start() method of Thread class is used to start a newly created thread.

EXECUTING MULTIPLE THREADS

```
class Multi extends Thread
{
    public void run()
    {
        System.out.println("thread is running...");
    }
}

class Example extends Thread
{
    public void run()
    {
        System.out.println("thread2 is running...");
    }
}

public class Main
{

    public static void main(String args[])
    {
        Multi t1 = new Multi();

        Example t2 = new Example();

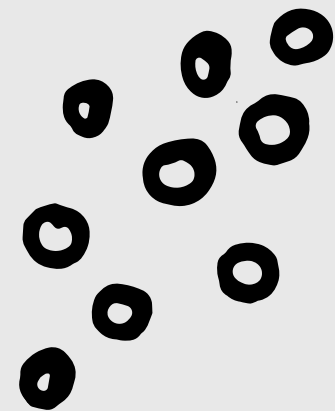
        t1.start();
        t2.start();

    }
}
```

```
thread is running...
thread2 is running...
```



HANDLING A THREAD




```
2
3 class Multi extends Thread
4 {
5     public void run()
6     {
7         System.out.println("thread is running...");
8     }
9 }
10
11
12
13 class Example extends Thread
14 {
15
16     public void run()
17     {
18         System.out.println("thread2 is running...");
19     }
20 }
21
22 public class Main
23 {
24
25
26     public static void main(String args[])
27     {
28         Multi t1 = new Multi();
29
30         Example t2 = new Example();
31
32         t1.start();
33         t2.start();
34
35         System.out.println("hello");
36
37     }
38 }
39
```

```
thread is running...
hello
thread2 is running...
```



THREAD METHODS

- **start()** - Starts the thread. 
- **getState()** - It returns the state of the thread.
- **setName()** - It sets name for the thread.
- **getName()** - It returns name for the thread.
- **setPriority()** - It sets the priority for the thread.
- **getPriority()** - It returns the priority of the thread.
- **sleep()** - Stop the thread for the specified time.
- **join()** - Stop the current thread until the called thread gets terminated.
- **isAlive()** - Check if the thread is alive.

THREAD PRIORITY

.....



Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority.



The `setPriority(int newPriority)` method throws `IllegalArgumentException` if the value `newPriority` goes out of the range, which is 1 (minimum) to 10 (maximum).



3 CONSTANTS DEFINED IN THREAD CLASS

PUBLIC STATIC INT MIN_PRIORITY

The value of MIN_PRIORITY
is 1.

PUBLIC STATIC INT NORM_PRIORITY

Default priority of a thread
is 5 (NORM_PRIORITY).


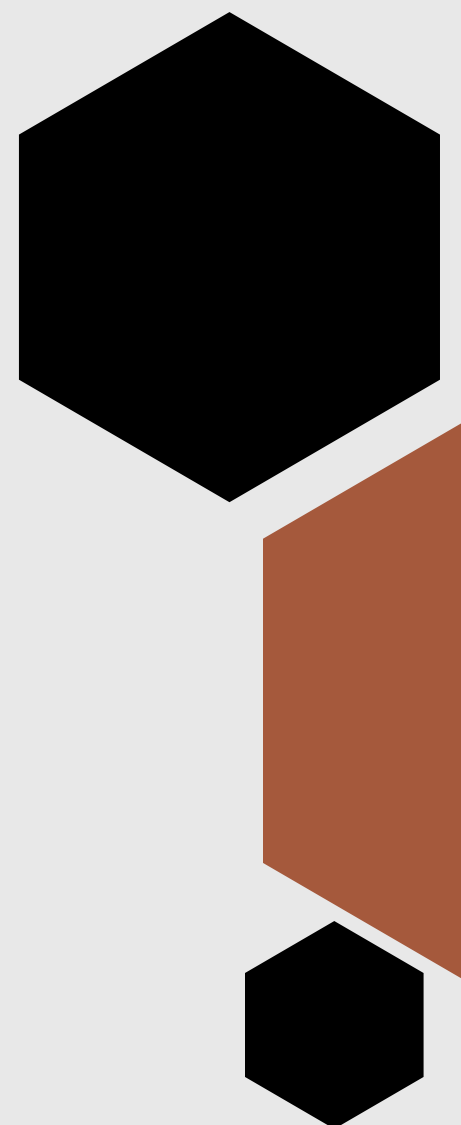

PUBLIC STATIC INT MAX_PRIORITY

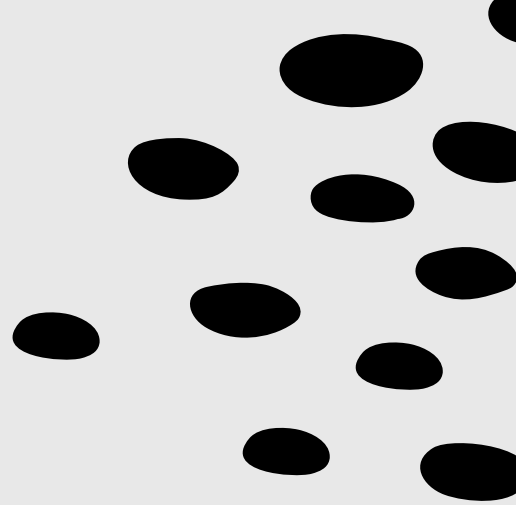
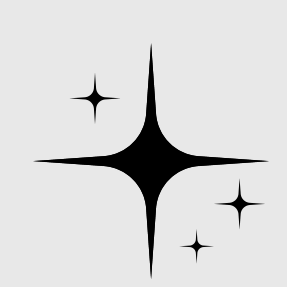
The value of
MAX_PRIORITY is 10.



SYNCHRONIZATION

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.





IMPLEMENTING A RUNNABLE INTERFACE

Java runnable is an interface used to execute code on a concurrent thread.

The runnable interface has an undefined method `run()` with void as return type, and it takes in no arguments.

To create a thread using runnable interface,

