Theory (Friend Function)

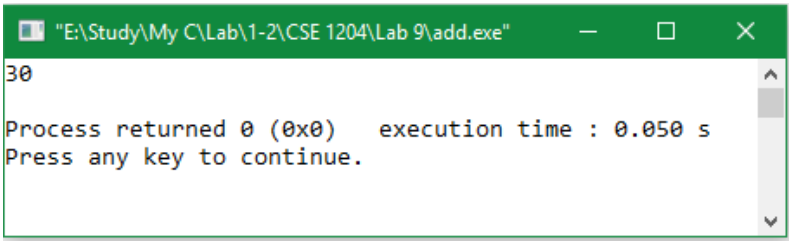
Friend function is a special feature of C++ language. A ***friend*** is not a member of a class but still has the access to its private elements. Two reasons that friend functions are useful have to do with operator overloading and the creation of certain types of I/O functions. A friend function is defined as a regular, non-member function. However, inside the class declaration for which it will be friend, its prototype is also included, prefaced by the keyword ***friend***

Code

```
#include<iostream>
using namespace std;
class A {
public:
   int x,y;
   void get_x(int a, int b) {
      x = a;
      y = b;
   }
   friend int sum(A ob1);
};
int sum(A ob1) {
   return ob1.x+ob1.y;
}
int main()
{
   A ob;
   ob.get_x(10,20);
   cout<<sum(ob)<<endl;

   return 0;
}
```

Output



Comment

Here the sum of two number was executed by using a friend function sum(). Sum() function was not a member of class A but still it had the access to the private member of the class.

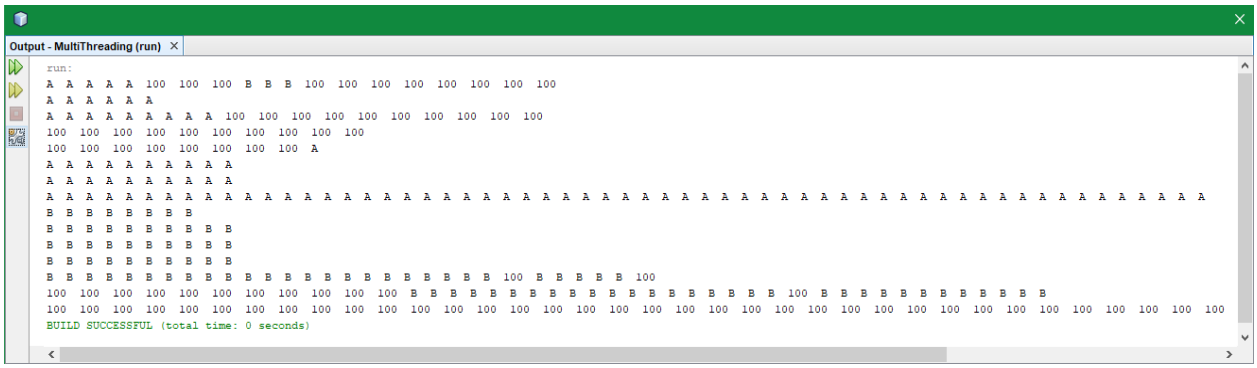
Theory (Multithreading)

Multithreading is a specialized form of multitasking. In a thread-based multitasking environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously. For instance, a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads.

Code

```
package multithreading;
class Test implements Runnable {
   char c;
   public Test(char s) {
      c = s;
   }
   public void run() {
      int i;
      for (i=0 ; i<100 ; i++) {
         System.out.println(c+"  ");
      }
   }
}
class Test1 implements Runnable {
   int i1;
   public Test1(int i) {
      i1 = I; }
   public void run() {
      int i;
      for (i=0 ; i<100 ; i++) {
         System.out.println(i1+"  ");
      }
   }
}
public class MultiThreading {
   public static void main(String[] args) {
      Test t1 = new Test('A');
      Test t2 = new Test('B');
      Test1 t3 = new Test1(100);
      Thread thread1 = new Thread(t1);
      Thread thread2 = new  Thread(t2);
      Thread thread3 = new Thread(t3);
      thread1.start();
      thread2.start();
      thread3.start();
   }
}
```

Output



Comment

Here Test class was implemented from Runnable class to print run the thread at most 100 times and for that in main function three new threads were declared and they were started using start method. And this is how the multithreading works.