Week – 1

Aim: To learn Basic programs in C++ and Java.

1) **Write a CPP program for accepting user inputs and display it.**

Description:
In C++ to access the output stream, we need to use the ‘std’ namespace. As it is by default private, we need to access it through the scope resolution operator.

Program:
```
#include<iostream>
int main() {
    std::cout << "Hello world!" << std::endl;
}
```
Output:
```
Hello world!
```

2) **Write a CPP program for GCD, PRIME NUMBER, Multiplication Table, roots of quadratic equation**

Description:
In C++ programming “using namespace std” header allows us to directly access the std namespace methods. There are 4 method in std namespace they are: cout, cin, cerr and clog.

a) **GCD.cpp:**
```
#include <iostream>
using namespace std;
int main() {
        int n1, n2, GCD;
        cout << "Enter two numbers: ";
        cin >> n1 >> n2;
        if ( n2 > n1) {
        int temp = n2;
        n2 = n1;
        n1 = temp;
    }
    for(int i = 1; i <=  n2; i++) {
            if (n1 % i == 0 && n2 % i ==0) {
                    GCD = i;
            }
     }
     cout << "The GCD of " << n1 << " and " << n2 << " is " << GCD;
     return 0;
}
```

Output:

```
Enter the two numbers : 14 49
The GCD of 14 and 49 is 7
```

b) **PRIME.cpp:**

```
#include<iostream>
using namespace std;
int main() {
        int n,i,count = 0;
        cout << "Enter the number : ";
        cin >> n;
        for(i=2; i<n; i++) {
      if (n % i == 0)
         count ++;
        }
        cout << n <<" is a ";
  if(count != 0)
    cout << "COMPOSITE number";
  else
    cout << "PRIME number";
        return 0;
}
```

Output:

```
Enter the number : 7
7 is a PRIME number
```

c) **MULTIPLICATION.cpp:**

```
#include<iostream>
using namespace std;
int main(){
  int a, i, n;
  cout << "Enter table format : ";
  cin >> a;
  cout << "Enter end of table : ";
```

```
      cin >> n;
  for(i = 0; i <= n; i++){
    cout << a << " X " << i << " = " << a*i << endl;
  }
}
```

Output:

```
Enter table format : 2
Enter end of table : 8
2 X 0 = 0
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
```

d) ROOTS.cpp

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
  float a, b, c, x1, x2, d, r, i;
  cout << "Enter coefficients a, b and c: ";
  cin >> a >> b >> c;
  d = b*b - 4*a*c;
  if (d > 0) {
    x1 = (-b + sqrt(d)) / (2*a);
    x2 = (-b - sqrt(d)) / (2*a);
    cout << "Roots are real and different." << endl;
    cout << "x1 = " << x1 << endl;
    cout << "x2 = " << x2 << endl;
  }
  else if (d == 0) {
    cout << "Roots are real and same." << endl;
    x1 = -b/(2*a);
    cout << "x1 = x2 =" << x1 << endl;
  }
  else {
    r = -b/(2*a);
    i = sqrt(-d)/(2*a);
    cout << "Roots are complex and different."  << endl;
    cout << "x1 = " << r << " + (" << i << ") i" << endl;
    cout << "x2 = " << r << " - (" << i << ") i" << endl;
  }
  return 0;
}
```

Output:

```
Enter coefficients a, b and c: 1 2 3
Roots are complex and different.
x1 = -1 + (1.41421) i
x2 = -1 - (1.41421) i
```

3) Write a JAVA program for accepting user inputs and displaying them.

Description:

In java, we need to write them in a class and those methods or attributes can be accessed through the class which is having the main class. Also the name of the program should be same as the main class name.

Program:

```
class Hello {
   public static void main(String args[]) {
      System.out.println("Hello Java!");
   }
}
```

Output:

```
Hello Java!
```

4) Write a JAVA program for GCD, PRIME NUMBER, Multiplication Table

Description:

In order to give inputs in java, we use some special in-built classes that allow us to reduce the code and also reuse them. All these classes are under the Java class, we need to access them through "." Operator.

a) GCD.java:

```
import java.util.Scanner;

public class GCD {
    public static void main(String[] args) {
        int n1, n2, temp, GCD=1;
        Scanner a = new Scanner(System.in);
        System.out.println("Enter the n1 and n2 values : ");
        n1 = a.nextInt();
        n2 = a.nextInt();
        for(int i=1;i<=n1 && i<=n2;i++) {
            if( n1%i == 0 && n2%i == 0 )
            GCD = i;
        }
        System.out.printf("GCD of %d and %d is %d",n1 ,n2 ,GCD);
    }
}
```

Output:

```
Enter the n1 and n2 values :
12
56
GCD of 12 and 56 is 4
```

b) **PRIME.java:**

```
import java.util.Scanner;

public class Prime {
    public static void main(String args[]) {
        int i,m=0,flag=0;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer : ");
        int n = sc.nextInt();
        if(n==0||n==1)      {
            System.out.println(n+" is not prime number");
        }
        else {
            for(i=2;i<n;i++){
                if(n%i==0) {
                      System.out.println(n+" is COMPOSITE number");
                      flag=1;
                      break;
                }
            }
            if(flag==0)  {
                System.out.println(n+" is PRIME number");
                }
            }
        }
    }
```

Output:

```
Enter an integer : 43
43 is prime number
```

c) **MulTable.java**

```
import java.util.Scanner;

public class MulTable {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter table format : ");
                int n = sc.nextInt();
                System.out.print("Enter end of table : ");
                int m = sc.nextInt();
                for(int i = 1; i <= m; ++i) {
                        System.out.printf("%d * %d = %d \n", n, i, n * i);
                }
        }
}
```

Output:

```
Enter table format : 4
Enter end of table : 9
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
```

5) Write the usage of the User-defined functions in CPP and JAVA for GCD, Prime numbers, roots of quadratic equation

Description:

Similar to C programming in C++ also we import the header files in the same fashion but the header files in C++ are different from the C language.

a) GCDfun.cpp:

```
#include <iostream>
using namespace std;
int GCD(int a,int b) {
        if ( b > a) {
        int temp = b;
        b = a;
        a = temp;
        }
        if (b == 0)
        return a;
        else
        return GCD(b, a % b);
}
int main() {
        int n1, n2, gcd;
        cout << "Enter two numbers: ";
        cin >> n1 >> n2;
        gcd = GCD(n1,n2);
        cout << "The GCD of " << n1 << " and " << n2 << " is " << gcd;
        return 0;
}
```

Output:

```
Enter two numbers: 12 132
The GCD of 12 and 132 is 12
```

b) PrimeFun.cpp:

```
#include<iostream>
using namespace std;
void Prime(int n) {
        int count = 0,i;
        for(i=2; i<n; i++) {
      if (n % i == 0)
         count ++;
        }
  if(count != 0)
    printf("%d is a COMPOSITE number\n",n);
  else
```

```
        printf("%d is a PRIME number\n",n);
    }
    int main() {
            int n;
            cout << "Enter the number : ";
            cin >> n;
            Prime(n);
    }
```

Output:

```
Enter the number : 17
17 is a PRIME number
```

c) **RootsFun.cpp:**

```
#include <iostream>
#include <cmath>
using namespace std;

void Roots(int a,int b,int c) {
        int d;
        double r,i,x1,x2;
  d = b*b - 4*a*c;
  if (d > 0) {
    x1 = (-b + sqrt(d)) / (2*a);
    x2 = (-b - sqrt(d)) / (2*a);
    cout << "Roots are real and different." << endl;
    cout << "x1 = " << x1 << endl;
    cout << "x2 = " << x2 << endl;
  }
  else if (d == 0) {
    cout << "Roots are real and same." << endl;
    x1 = -b/(2*a);
    cout << "x1 = x2 =" << x1 << endl;
  }
  else {
    r = -b/(2*a);
    i = sqrt(-d)/(2*a);
    cout << "Roots are complex and different."  << endl;
    cout << "x1 = " << r << " + (" << i << ") i" << endl;
    cout << "x2 = " << r << " - (" << i << ") i" << endl;
  }
}
```

```
int main() {
  float a, b, c, x1, x2, d, r, i;
  cout << "Enter coefficients a, b and c: ";
  cin >> a >> b >> c;
        Roots(a,b,c);
        return 0;
}
```

Output:

```
Enter coefficients a, b and c: 1 3 5
Roots are complex and different.
x1 = -1 + (1.65831) i
x2 = -1 - (1.65831) i
```

d) GCDfun.java:

```
import java.util.Scanner;

public class GCDfun {
        public static void main(String[] args) {
                Scanner a = new Scanner(System.in);
                System.out.println("Enter the n1 and n2 values : ");
                int n1 = a.nextInt();
                int n2 = a.nextInt();
                System.out.printf("GCD of %d and %d is %d\n",n1 ,n2
,findGCD(n1,n2));
        }
        static int findGCD(int a, int b) {
                if (b == 0)
                        return a;
                return findGCD(b, a % b);
        }
}
```

Output:

```
Enter the n1 and n2 values :
13
26
GCD of 13 and 26 is 13
```

e) PrimeFun.java:

```
import java.util.Scanner;
```

```
public class Prime {
        void prime(int n) {
                int i,m=0,flag=0;
                if(n<=1) {
                        System.out.println(n+" is not prime number");
                }
                else {
                  for(i=2;i<n;i++){
                    if(n%i==0) {
                        System.out.println(n+" is COMPOSITE number");
                                flag=1;
                                break;
                    }
                  }
                  if(flag==0)  {
                        System.out.println(n+" is PRIME number");
                  }
                }
        }
        public static void main(String args[]) {
                Prime p = new Prime();
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter an integer : ");
                int n = sc.nextInt();
                p.prime(n);
        }
}
```

Output:

```
Enter an integer : 5
5 is PRIME number
```

6) Write the Operations of Arrays in CPP and JAVA

Description:

In C++ we declare the arrays in the same fashion but in Java it is completely different from C and C++. It is declared as: data_type[] array_name;.

a) ArraySum.cpp:

```
#include <iostream>
using namespace std;
int main() {
        int n, s = 0, I = 0;
        cout << “Enter the size of the array : “;
        cin >> n;
        int arr[n];
        cout << “Enter the elements in the array : “;
        while(I < n) {
                cin >> arr[i];
                s += arr[i];
                i++;
        }
        i=0;
        cout << “The sum of “;
        while(I < n) {
                printf(“%d “,arr[i]);
                i++;
        }
        cout << “ is : “ << s;
        return 0;
}
```

Output:

```
Enter the size of the array : 3
Enter the elements in the array : 1 2 3
The sum of 1 2 3  is : 6
```

b) Arrays.java:

```
import java.util.Scanner;

public class Arrays {
        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                int[] ele;
                int n,v,s = 0;
                I(“Enter the No. of elements : “);
                n = input.nextInt();
                ele = new int[n];
                I(“Enter the “+n+” elements values : “);
                for(int I = 0;I < n;i++) {
                        v = input.nextInt();
                        ele[i] = v;
                }
                for(int a : ele) {
                        s += a;
                }
                System.out.println(“Sum of “+n+” elements : “+s);
        }
}
```

Output:

```
Enter the No. of elements : 5
Enter the 5 elements values : 1 2 3 4 5
Sum of 5 elements : 15
```

Conclusion:

In this week I had learned about:

- How to write a Basic C++ and Java program and compile them on a software.
- How to give inputs from user and display it in C++ and Java.
- How to perform different mathematical operations using C++ and Java.
- How to perform function declarations and function calls in C++ and Java.
- How to declare, initiate and access elements of array using C++ and Java.

Week – 2

Aim: To know about Classes, objects, and Constructors.

1. **Write a CPP program for creating a class, object, and initializing class member variables, including scope resolution.**

   Description:
   In C++ we can declare classes, which differs us from C programming. In order to access the methods, attributes, and constructors we need to create objects using the class name while declaring.

   Class.cpp:

```
#include <iostream>
using namespace std;
class Name {
  public:
  string name;
  void printname() {
    cout << "Name is : " << name;
  }
};
int main() {
  Name obj;
  obj.name = "Ravi";
  obj.printname();
  return 0;
}
```

   Output:

```
Name is : Ravi
```

2. **Write a JAVA program for creating a class, object, and initializing class member variables.**

   Description:
   While creating objects for the base class, in C++ by default the entities inside the class are private whereas in java it is public. So, it is necessary to initiate an access modifier to change the class state.

   Class.java:

```
public class Class {
  String Name;
  String Course;
  int Age;
  public Class(String name, String course,int age) {
    Name = name;
    Course = course;
    Age = age;
  }
  public String getName() {
    return Name;
  }
    public String getCourse() {
            return Course;
    }
    public int getAge() {
            return Age;
    }
    public static void main(String[] args) {
    Class s1 = new Class("Ravi","CSE",54);
            System.out.println(s1.getName());
            System.out.println(s1.getCourse());
            System.out.println(s1.getAge());
  }
}
```

Output:

```
Ravi
CSE
54
```

3. **Write the program using the constructor in CPP and JAVA.**

Description:
Constructors are used to create/allocate memory for the object in a program. By default, these will be invoked while declaring the objects for a specific class. These can be user-defined also.

a) **Constructor.cpp:**

```
#include <iostream>
using namespace std;
class Employee {
	public:
	Employee() {
		cout<<"By Default Constructor Invoked"<<endl;
	}
};
int main() {
	Employee e1;
	Employee e2;
	return 0;
}
```

Output:

```
By Default Constructor Invoked
By Default Constructor Invoked
```

b) **Constructor.java:**

```
class Box {
	double width;
	double height;
	double depth;
	Box() {
		System.out.println("Constructing Box");
		width = 100;
		height = 100;
		depth = 100;
	}
	double volume() {
		return width * height * depth;
	}
}
class BoxDemo {
	public static void main(String args[]) {
		Box mybox1 = new Box();
		Box mybox2 = new Box();
		double vol;
		vol = mybox1.volume();
		System.out.println("Volume of the first box is " + vol);
		vol = mybox2.volume();
		System.out.println("Volume of the second box is " + vol);
	}
}
```

Output:

```
Constructing Box
Constructing Box
Volume of the first box is 1000000.0
Volume of the second box is 100000.0
```

4. Write a program using a different type of constructor calls in CPP.

Description:

While invoking the constructor we can pass arguments in different ways i.e., arguments of different datatypes, number of arguments, and sequence.

Constructor2.cpp:

```
#include <iostream>
using namespace std;
class Employee {
        public:
        int id;
        string name;
        float salary;
        Employee(int i, string n, float s) {
                id = i;
                name = n;
                salary = s;
        }
        void display() {
                cout << id << " " << name <<" "<< salary <<endl;
        }
};

int main() {
        Employee e1 = Employee(25, "Ravi", 80000);
        Employee e2 = Employee(50, "Kiran", 50000);
        e1.display();
        e2.display();
        return 0;
}
```

Output:

```
25 Ravi 80000
50 Kiran 50000
```

5. Write a program using the Copy constructor, and destructor in CPP

Description:
In C++ the constructors can be copied in 2 ways, one is a deep copy another one is a shallow copy. When the program terminates the destructor will be invoked which means the destructor will de-allocate the memory for the objects.

a) **CopyConstr.cpp:**

```
#include <iostream>
using namespace std;
class A {
        public:
        int x;
        A(int a) {
                x = a;
        }
        A(A &i) {
                x = i.x;
        }
};

int main() {
        A a1(20);
        A a2(a1);
        cout << "Variable in object a1 : " << a1.x << endl;
        cout << "Variable in copied object a2 : " << a2.x;
        return 0;
}
```

Output:

```
Variable in object a1 : 20
Variable in copied object a2 : 20
```

b) **Destuctor.cpp:**

```
#include <iostream>
using namespace std;
class Employee {
        public:
        Employee() {
                cout<<"Constructor Invoked"<<endl;
        }
        ~Employee() {
                cout<<"Destructor Invoked"<<endl;
        }
};
int main() {
        Employee e1;
        Employee e2;
        return 0;
}
```

Output:

```
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked
```

Conclusion:

In this week I had learned about:

- How to create classes, and objects and initialize the objects under a class in C++ and java.
- How to give various methods in a class and access them through objects in C++ and java.
- Understanding the purpose of constructor and destructor in C++ and java.
- How to use the constructor with and without parameters in C++ and java.
- How to copy the constructors of one object to another object in C++.

Week – 3

Aim: To get familiar with overloading, inline, static functions, and command-line arguments in C++ and java.

1. Write a program showing Method Overloading in CPP, JAVA

Description:
In C++ and java, we create the same method with different parameters such that the method is accessed based on the parameters.

a) MethodOL.cpp:

```
#include<iostream>
using namespace std;
class PrintData {
        public:
        void print(int a) {
                cout << a << " is a Integer.\n";
        }
        void print(string a) {
                cout << a << " is a String.\n";
        }
        void print(double a) {
                cout << a << " is a Float.\n";
        }
};
int main() {
        PrintData d;
        d.print(5);
        d.print("ravi");
        d.print(5.54);
        return 0;
}
```

Output:

```
5 is a Integer.
ravi is a String.
5.54 is a Float.
```

b) MethodOL.java:

```
public class MethodOL {
        public void disp(String a) {
                System.out.println(a+" is a String");
        }
        public void disp(char a) {
                System.out.println(a+" is a Character");
        }
        public void disp(int a) {
                System.out.println(a+" is an Integer");
        }
        public void disp(double a) {
                System.out.println(a+" is a Float");
        }
}

class Display {
        public static void main(String args[]) {
                MethodOL obj = new MethodOL();
                obj.disp("Ravi");
                obj.disp(5);
                obj.disp(5.54);
                obj.disp('R');
        }
}
```

Output:

```
Ravi is a String
5 is an Integer
5.54 is a Float
R is a Character
```

2. **Write a program using Static keywords, static methods static blocks usage.**

Description:

In C++ and Java, we create the variable method and blocks with a static keyword such that only one copy of a member is created for the entire class shared by all objects of that class.

a) **StaticKeyword.cpp:**

```
#include <iostream>
#include <string>
using namespace std;
void Static ()
```

```
{
        static int a = 0;
        cout << a << " ";
        a++;
}
int main()
{
        for (int i=0; i<5; i++)
                Static();
        return 0;
}
```

Output:

```
0 1 2 3 4
```

b) StaticMethod.cpp:

```
#include<iostream>
using namespace std;
class Static
{
        public:
                static void print()
                {
                cout<<"Hello CSE-A";
                }
};
int main()
{
        Static::print ();
}
```

Output:

```
Hello CSE-A
```

c) StaticBlock.cpp:

```
class Static{
   static int a;
   static
   {
      a= 777;
      System.out.println("static block is invoked");
   }
}
```

```
class StaticBlock{
   public static void main(String args[])
   {
      System.out.println(Static.a);
   }
}
```

Output:

```
777
static block is invoked
```

3. Write a program using Inline functions and friend function in CPP

Description:

In C++ we create a method or constructor using the keyword inline such that the whole function is substituted at the inline function call. We create friend keyword to access private or protected items of the class by friend class.

a) Inline.cpp:

```
#include <iostream>
using namespace std;
inline int factorial(int s)
{
   If(s!=1)
   return s*factorial(s-1);
   else
   return 1;
}
int main()
{
   cout << "The 6! is: " << factorial(6) << "\n";
   return 0;
}
```

Output:

```
The 6! Is: 720
```

b) Friend.cpp:

```
#include <iostream>
using namespace std;
class A {
private:
   int a;
public:
   A() { a = 0; }
   friend class B;
};

class B {
private:
   int b;

public:
   void Value(A& x)
   {
      cout << "a=" << x.a;
   }
};

int main()
{
   A x;
   B y;
   y.Value(x);
   return 0;
}
```

Output:

```
a=0
```

4. **Write the program showing Operator overloading, and Assignment Operator overloading using CPP.**

Description:

In C++ and Java, we create a method such that it has a special keyword operator and consists of an operator symbol between one or more objects.

a) OperatorOverloading.cpp:

```
#include<iostream>
using namespace std;
class Complex {
private:
   int rp, imp;
public:
   Complex(int r = 0, int i = 0) {rp= r;   imp = i;}
   Complex operator + (Complex const &obj) {
       Complex res;
       res. rp = rp + obj. rp;
       res. imp = imp + obj. imp;
       return res;
   }
   void print() { cout << rp << " + i" << imp << '\n'; }
};

int main()
{
   Complex c1(1, 2), c2(5, 4);
   Complex c3 = c1 + c2;
   c3.print();
}
```

Output:

```
6 + i6
```

b) AssignmentOverloading.cpp:

```
#include <iostream>
using namespace std;
class Distance {
  private:
    int feet;
    int inches;
  public:
    Distance() {
      feet = 0;
      inches = 0;
    }
    Distance(int f, int i) {
      feet = f;
      inches = i;
    }
    void operator = (const Distance &D ) {
      feet = D.feet;
      inches = D.inches;
    }
    // method to display distance
    void displayDistance() {
      cout << "F: " << feet <<  " I:" <<  inches << endl;
    }
};

int main() {
  Distance D1(11, 10), D2(5, 11);
  cout << "First Distance : ";
  D1.displayDistance();
  cout << "Second Distance :";
  D2.displayDistance();
  D1 = D2;
  cout << "First Distance :";
  D1.displayDistance();
  return 0;
}
```

Output:

```
First Distance : F: 11 I:10
Second Distance :F: 5 I:11
First Distance :F: 5 I:11
```

5. Write the CPP program using Command line arguments, Manipulators, string Math header files

Description:

In C++ we create programs with header files such as string, cmath, and different manipulators to manipulate the output and access different methods in the header file.
We create a program that handles command-line arguments using the main function.

a) Command.cpp:

```
#include<iostream>
int main (int argc,char *argv[]){
  if (argc == 2) {
    cout<<"The argument supplied is "<<argv[1]<< "\n";
  }
  else if (argc > 2) {
    cout<<"To many arguments supplied.\n";
  }
  else {
    cout<<"One argument expected.\n";
  }
}
```

Output:

1. $ ./main a
   The argument supplied is a
2. $ ./main a b
   To many arguments supplied.
3. $./main
   One argument expected.

b) **Manipulator.cpp:**

```
#include <iostream>
#include <iomanip>
#include <istream>
#include <sstream>
#include <string>
using namespace std;
int main()
{
   double A = 100;
   double B = 2001.5251;
   double C = 201455.2646;
   cout << hex << left << showbase << nouppercase;
   cout << (long long)A << endl;
   cout << setbase(10) << right << setw(15) << setfill('_') << showpos << fixed << 
setprecision(2);
   cout << B << endl;
   cout << scientific << uppercase<< noshowpos << setprecision(9);
   cout << C << endl;

   istringstream str("          Programmer");
   string line;
   getline(str >> std::ws, line);
   cout << line << endl;
   cout << "only a test" << flush;
   cout << "\n";
   cout << "b" << ends;
   cout << "c" << endl;
   return 0;
}
```

Output:

```
0x64
________+2001.53
2.014552646E+05
Programmer
Only a test
abc
```

c) **String.cpp:**

```
#include<string>
int main(){
string a;
string b;
cin>>a;
cin>>b;
string c = a + b;
cout<<c;
string d = a.append(b);
cout << d;
cout << "The length of d : " << d.size();
cout << "First letter of d :" << d[0];
return 0;
}
```

Output:

```
Ravi kiran varma
Ravi kiran varma
Ravi kiran varma
The length of d : 15
First letter of d : R
```

d) **Math.cpp:**

```
#include <cmath>
#include <iostream>
int main(){
 cout << sqrt(64)<<endl;
 cout << round(2.6)<<endl;
 cout << log(10)<<endl;
 return 0;
}
```

Output:

```
8
3
0.693147
```

CONCLUSION:

In this week I had learned about:

- How to use method overloading in C++ and java.
- Usage of static keywords, static methods, and static blocks.
- How to make inline functions using C++.
- How to use operator overloading and assignment overloading in C++.
- Usage of command-line arguments, manipulators, string and math header files.

WEEK 4

Aim: To study different types of inheritance and polymorphism.

1) **Write a program using Inheritance & types of Inheritance in CPP, JAVA.**

a) **Description:**
In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only, we need to access it through the scope resolution operator.

SingleInherit.cpp:
```
#include<iostream>
using namespace std;
 class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle\n";
    }
};
class Car : public Vehicle {
};
int main()
{
  Car obj;
    return 0;
}
```
Output:
```
This is a Vehicle
```

b) **Description:**
Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one **sub class** is inherited from more than one **base classes**, we need to access it through the scope resolution operator.

<u>MultipleInherit.cpp</u>:

```
#include<iostream>
using namespace std;
class Vehicle {
public:
        Vehicle()
        {
        cout << "This is a Vehicle\n";
        }
};
class FourWheeler {
public:
        FourWheeler()
        {
        cout << "This is a 4 wheeler Vehicle\n";
        }
};
class Car : public Vehicle, public FourWheeler {
};
int main()
{
car obj;
        return 0;
}
```

<u>Output:</u>

```
This is a Vehicle
This is a 4 wheeler Vehicle
```

c) **<u>Description:</u>**

In this type of inheritance, a derived class is created from another derived class， we need to access it through the scope resolution operator.

<u>MultilevelInherit.cpp:</u>

```
#include<iostream>
using namespace std;
class Vehicle
{
public:
        Vehicle()
        {
        cout << "This is a Vehicle\n";
        }
};
class fourWheeler: public Vehicle
{ public:
        fourWheeler()
        {
        cout << "Objects with 4 wheels are vehicles\n";
        }
};
class Car: public fourWheeler {
public:
        Car()
        {
        cout << "Car has 4 Wheels\n";
        }
};
int main()
{
Car obj;
        return 0;
}
```

Output:

```
This is a Vehicle
Objects with 4 wheels are vehicles
Car has 4 Wheels
```

d) Description:

In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class， we need to access it through the scope resolution operator.

HierarchicalInherit.cpp:

```
#include<iostream>
using namespace std;
class Vehicle {
public:
        Vehicle() {
        cout << "This is a Vehicle\n";
        }
};
class Car: public Vehicle {
};
class Bus: public Vehicle {
};
int main() {
        Car obj1;
        Bus obj2;
        return 0;
}
```

Output:

```
This is a Vehicle
This is a Vehicle
```

e) **Description:**

Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance, we need to access it through the scope resolution operator.

HybridInherit.cpp:

```
#include<iostream>
using namespace std;
class Vehicle {
public:
        Vehicle() {
        cout << "This is a Vehicle\n";
        }
};
class Fare {
        public:
        Fare() {
                cout << "Fare of Vehicle\n";
        }
};
class Car : public Vehicle {
};
class Bus : public Vehicle, public Fare {
};
int main() {
        Bus obj2;
        return 0;
}
```

Output:

```
This is a Vehicle

Fare of Vehicle
```

f) **Description:**

In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class B.

SingleInherit.java:

```
class A {
 int a, b;
 void display() {
      System.out.println("Inside class A values ="+a+" "+b);
 }
}
class B extends A {
 int c;
 void show() {
      System.out.println("Inside Class B values="+a+" "+b+" "+c);  }
     }
class SingleInheritance {
 public static void main(String args[]) {
            B obj = new B();
            obj.a=10;
            obj.b=20;
            obj.c=30;
            obj.display();
            obj.show();
 }
}
```

Output:

```
Inside class A values =10 20

Inside Class B values=10 20 30
```

g) **Description:**

In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support classes. In java, we can achieve multiple inheritances only. In the image below, Class C is derived from iinterfaces A and B.

MultipleInherit.java:

```
import java.io.*;
import java.lang.*;
import java.util.*;
interface one {
        public void print_geek();
}
interface two {
        public void print_for();
}
interface three extends one, two {
        public void print_geek();
}
class child implements three {
        @Override public void print_geek()
        {
                System.out.println("o");
        }

        public void print_for() { System.out.println("pp"); }
}
public class Main {
        public static void main(String[] args)
        {
                child c = new child();
                c.print_geek();
                c.print_for();
                c.print_geek();
        }
}
```

Output:

```
oppo
```

h) **Description:**

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived class B, C and D.

HierarchicalInherit.java:

```
class A {
	public void print_A() { System.out.println("Class A"); }
}
class B extends A {
	public void print_B() { System.out.println("Class B"); }
}
class C extends A {
	public void print_C() { System.out.println("Class C"); }
}
class D extends A {
	public void print_D() { System.out.println("Class D"); }
}
public class Test {
	public static void main(String[] args)
	{
		B obj_B = new B();
		obj_B.print_A();
		obj_B.print_B();
		C obj_C = new C();
		obj_C.print_A();
		obj_C.print_C();
		D obj_D = new D();
		obj_D.print_A();
		obj_D.print_D();
	}
}
```

Output:

```
Class A

Class B
Class A
Class C
Class A
Class D
```

i) **Description:**

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.

MultilevelInherit.java:

```
import java.io.*;
import java.lang.*;
import java.util.*;
class one {

public void print_geek()

{

        System.out.println("o");

}

}

class two extends one {

public void print_for() { System.out.println("p"); }

}
class three extends two {
        public void print_geek()
        {

        System.out.println("ps");

        }
}
```

```
public class Main {
        public static void main(String[] args)
        {
                three g = new three();
                g.print_geek();
                g.print_for();
                g.print_geek();
}
}
```

Output:

```
o

p

ps
```

2) Write a program using Method Overriding & super keyword in CPP, JAVA.

a) **MethodOR.cpp:**

Description:

It is the redefinition of base class function in its derived class with the same signature i.e return type and parameters. **:** Overriding is needed when the derived class function has to do some added or different job than the base class function.

```
#include<iostream>
using namespace std;
class BaseClass
{
public:
        virtual void Display()
        {
                cout << "\nThis is Display() method"
                        " of BaseClass";
        }
        void Show()
        {
                cout << "\nThis is Show() method "
                        "of BaseClass";
        }
};
class DerivedClass : public BaseClass
{
public:
        // Overriding method - new working of
        // base class's display method
        void Display()
        {
                cout << "\nThis is Display() method"
                        " of DerivedClass";
        }
};
int main()
{
        DerivedClass dr;
        BaseClass &bs = dr;
        bs.Display();
        dr.Show();
}
```

Output:

```
This is Display() method of DerivedClass
This is Show() method of BaseClass
```

b) MethodOR.java:

Description:

Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to *override* the method in the super-class.

```
class Parent {
        void show()
        {
                System.out.println("Parent's show()");
        }
}
class Child extends Parent {
        @Override
        void show()
        {
                System.out.println("Child's show()");
        }
}
class Main {
        public static void main(String[] args)
        {
                Parent obj1 = new Parent();
                obj1.show();
                Parent obj2 = new Child();
                obj2.show();
        }
}
```

Output:

```
Parent's show()
Child's show()
```

c) **Super.CPP:**

Description:

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.

Code:

```
class Person
{
        Person()
        {
                System.out.println("Person class Constructor");
        }
}
class Student extends Person
{
        Student()
        {
                                super();
        System.out.println("Student class Constructor");
        }
}
class Test
{
        public static void main(String[] args)
        {
                Student s = new Student();
        }
}
```

Output:

```
Person class Constructor
Student class Constructor
```

Super.java:

Description:

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.

Code:

```
 class Person
{
          Person()
          {
                    System.out.println("Person class Constructor");
          }
}
class Student extends Person
{
          Student()
          {
                    super();
          System.out.println("Student class Constructor");
          }
}
class Test
{
          public static void main(String[] args)
          {
                    Student s = new Student();
          }
}
```

Output:

```
Person class Constructor
Student class Constructor
```

3) Write a program of Type casts in CPP and JAVA, “this” & “final” keywords in CPP and JAVA, access specifiers in CPP and JAVA.

a) typeCast.cpp:
Description:

Typecasting refers to the conversion of one data type to another in a program. Typecasting can be done in two ways: automatically by the compiler and manually by the programmer or user. Type Casting is also known as Type Conversion.

Code:

```
#include <iostream>
using namespace std;
int main ()
{
   short x = 200;
   int y;
   y = x;
   cout << " Implicit Type Casting " << endl;
   cout << " The value of x: " << x << endl;
   cout << " The value of y: " << y << endl;
 int num = 20;
   char ch = 'a';
   int res = 20 + 'a';
   cout << " Type casting char to int data type ('a' to 20): " << res << endl;
     float val = num + 'A';
   cout << " Type casting from int data to float type: " << val << endl;
   return 0;
}
```

Output:

```
The value of x: 200
The value of y: 200
Typecasting char to int data type ('a' to 20): 117
Typecasting from int data to float type: 85
```

b) **typeCast.java:**

Description:

Type Casting is done during the program design time by the programmer. Typecasting also refers to Narrow Conversion. Because in many cases, We have to Cast large datatype values into smaller datatype values according to the requirement of the operations.

Code:

```
import java.io.*;
public class GFG {
        public static void main(String[] args)
        {
                int a = 3;
                double db = (double)a;
                System.out.println(db);
                int db1 = (int)db;
        System.out.println(db1);
        }
}
```

Output:

```
3.0
3
```

a) **this.java:**

Description:

'this' is a reference variable that refers to the current object.

Code:

```
class A
{
        B obj;
        A(B obj)
        {
                this.obj = obj;
                obj.display();
        }
        }
class B
{
        int x = 5;
        B()
        {
                A obj = new A(this);
        }

        void display()
        {
                System.out.println("Value of x in Class B : " + x);
        }

        public static void main(String[] args) {
                B obj = new B();
        }
}
```

Output:

```
Value of x in Class B : 5
```

b) final.cpp:

Description:

Sometimes you don’t want to allow derived class to override the base class’ virtual func

Code:

```
#include <iostream>
using namespace std;
class Base
{
public:
        virtual void myfun() final
        {
                cout << "myfun() in Base";
        }
};
class Derived : public Base
{
        void myfun()
        {
                cout << "myfun() in Derived\n";
        }
};
int main()
{
        Derived d;
        Base &b = d;
        b.myfun();
        return 0;
}
```

Output:

```
prog.cpp:14:10: error: virtual function ‘virtual void Derived::myfun()’
   void myfun()
       ^
prog.cpp:7:18: error: overriding final function ‘virtual void Base::myfun()’
   virtual void myfun() final
```

c) **final.java:**

Description:

When a variable is declared with the **final keyword,** its value can’t be modified, essentially, a constant. This also means that you must initialize a final variable. If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but the internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from the or final collection.

Code:

```
class GFG {
        public static void main(String[] args)
        {
                int arr[] = { 1, 2, 3 };
                for (final int i : arr)
                        System.out.print(i + " ");
        }
}
```

Output:

```
1 2 3
```

d) accesSpecifiers.cpp:

Description:

Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

Code:

```
#include<iostream>
using namespace std;
class Circle
{
        public:
                double radius;

                double compute_area()
                {
                        return 3.14*radius*radius;
                }
        };
int main()
{
        Circle obj;

        obj.radius = 5.5;
        cout << "Radius is: " << obj.radius << "\n";
        cout << "Area is: " << obj.compute_area();
        return 0;
}
```

Output:

```
Radius is: 5.5

Area is: 94.985
```

e) **accesSpecifiers.java:**

Description:

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor, variable, method, or data membe

Code:

```
package p1;
class A
{
private void display()
        {
                System.out.println("GeeksforGeeks");
        }
}
class B
{
public static void main(String args[])
        {
                A obj = new A();
                // Trying to access private method
                // of another class
                obj.display();
        }
}
```

Output:

```
Compile time error
```

4) **Write a program of Static Polymorphism using inheritance – C++.**

StaticPoly.CPP:

Description:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism, a person at the same time can have different characteristics.

Code:

```
#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    void func(int x)
    {
            cout << "value of x is " << x << endl;
    }
    void func(double x)
    {
            cout << "value of x is " << x << endl;
    }
    void func(int x, int y)
    {
            cout << "value of x and y is " << x << ", " << y << endl;
    }
};
int main() {
    Geeks obj1;
    obj1.func(7);
    obj1.func(9.132);
    obj1.func(85,64);
    return 0;
}
```

Output:

```
value of x is 7
value of x is 9.132
value of x and y is 85, 64
```

5) Write a program of Dynamic Polymorphism in CPP JAVA.

a) **DynamicPoly.cpp:**

Description:

the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

Code:

```
#include <bits/stdc++.h>
using namespace std;
class base
{
public:
        virtual void print ()
        { cout<< "print base class" <<endl; }
        void show ()
        { cout<< "show base class" <<endl; }
};
class derived:public base
{
public:
        void print () //print () is already virtual function in derived class, we could
also declared as virtual void print () explicitly
        { cout<< "print derived class" <<endl; }
        void show ()
        { cout<< "show derived class" <<endl; }
};
```

```
int main()
{
        base *bptr;
        derived d;
        bptr = &d;

        //virtual function, binded at runtime (Runtime polymorphism)
        bptr->print();

        // Non-virtual function, binded at compile time
        bptr->show();

        return 0;
}
```

Output:

```
print derived class
show base class
```

b) DynamicPoly.java

Description:

It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

code:

```
class Parent {
        void Print()
        {
                System.out.println("parent class");
        }
}
class subclass1 extends Parent {
        void Print() { System.out.println("subclass1"); }
}
class subclass2 extends Parent {
        void Print()
        {
                // Print statement
                System.out.println("subclass2");
        }
}
class GFG {
        public static void main(String[] args)
        {
                Parent a;
                a = new subclass1();
                a.Print();
                a = new subclass2();
                a.Print();
        }
}
```

Output:

```
subclass1
subclass2
```

Conclusion:

In this week I had learned about:

- Different types of inheritance in C++ and java.
- Method overriding and usage of super keyword in C++ and java.
- Different models of typecasting in C++ and java.
- Usage and differences of this and final keywords in C++ and java.
- Programs on static and dynamic polymorphism.

Week – 5

Aim: To learn about multiple inheritances in C++ and Java.

1. Write a CPP program to show abstract classes and pure abstract classes.

Description 1:

An abstract class is, conceptually, a class that cannot be instantiated and is usually implemented as a class that has one or more pure virtual (abstract) functions.

A pure virtual function is one that must be overridden by any concrete (i.e., non-abstract) derived class. This is indicated in the declaration with the syntax " = 0" in the member function's declaration.

a) Program:

```
#include<iostream>
using namespace std;
class A
{
   public:
      int x;
      virtual void test() = 0;
};
class B : public A
{
   public:
      B(int y)
      {
         x = y;
      }
      void test()
      {
         std::cout<<"Hello I am virtual running in derived class and the number is
"<<x<<endl;
      }
};
```

```
int main(void)
{
   B obj(4);
   obj.test();
   return 0;
}
```

Output:

```
Hello I am virtual running in derived class and the number given 4
```

Description 2

An abstract class is one in which there is a declaration but no definition for a member function. The way this concept is expressed in C++ is to have the member function declaration assigned to zero.

b) Program :

```
#include<iostream>
using namespace std;
class A
{
   public:
      virtual void test() = 0;
};
class B : public A
{
   public:
      void test()
      {
         std::cout<<"Hello I am virtual running in derived class "<<endl;
      }
};

int main(void)
{
   B obj;
   obj.test();
   return 0;
}
```

Output:

```
Hello I am virtual running in the derived class
```

2. **Write a program using Abstract classes in java, the final keyword.**

Description:
A class that is declared with the abstract keyword is known as an abstract class in java. It can have abstract and non-abstract methods (method with the body).
An abstract method should be overwritten. But, once a method is written with the final keyword we can't overwrite it.

1. **Program:**

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end u
ser
class Rectangle extends Shape{
        void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
    void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method
s.draw();
}
}
```

Output:

drawing circle

2. **Program:**

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
  }
}
```

Output:

Compile time error.

3. **Write a program using interfaces in java.**

Description:

In java, we need to write them in a class and those methods or attributes can be accessed through the class which is having the main class. Also the name of the program should be same as the main class name.

Program:

```
interface printable
{
  void print();
}
class A6 implements printable
{
  public void print()
  {
     System.out.println("Hello");
  }
  public static void main(String args[])
```

```
    {
        A6 obj = new A6();
        obj.print();
    }
}
```

Output:

```
Hello
```

CONCLUSION:

The main 2 points from this I have learned is that abstract classes have features that can’t be instantiated by themselves but, can be done by their derived classes. Interfaces in java can be considered as templates, these templates can be used in how many ever classes we want.

Week – 6

Aim: To study packages and templates in Java.

1) Write a Java Program to call and access packages from the outside folder, Outside of the drive

Description:

Java package is a mechanism of grouping similar types of classes, interfaces, and sub-classes collectively based on functionality. Creating a package in Java is a very easy task. Choose a name for the package and include a package command as the first statement in the Java source file.

Compare.java:

```
package MyPackage;
public class Compare {
 int num1, num2;
 Compare(int n, int m) {
  num1 = n;
  num2 = m;
 }
 public void getmax(){
  if ( num1 > num2 ) {
   System.out.println("Maximum value of two numbers is " + num1);
  }
  else {
   System.out.println("Maximum value of two numbers is " + num2);
  }
 }
 public static void main(String args[]) {
  Compare current[] = new Compare[3];


  current[1] = new Compare(5, 10);
  current[2] = new Compare(123, 120);
  for(int i=1; i < 3 ; i++) {
   current[i].getmax();
  }
 }
}
```

Demo.java:

```
import MyPackage.Compare;


public class Demo{
  public static void main(String args[]) {
    int n=10, m=10;
    Compare current = new Compare(n, m);
    if(n != m) {
      current.getmax();
    }
    else {
      System.out.println("Both the values are same");
    }
  }
}
```

Compilation:

```
C:\Users\RaviV\Desktop\VSC\Java> javac -d . Compare.java
```

Output:

```
Maximum value of two numbers is 10

Maximum value of two numbers is 123

Both the values are same
```

2) Write a CPP program using Template functions & classes in C++.

Description:

In C++ we created templates in functions and classes such that we can use different classes with only one class code and functions with any methods.

a) TempFunc.cpp:

```
Template<class T>
Void swap(T &a,T &b){
T temp = a;
a = b;
b = temp;
}
Template<class T1 class T2>
Float Avg(T1 a,T2 b){
Float avg =(a+b)/2.0;
Return 0;
}
Int main(){
Float a;
a = Avg(5,2.4);
printf("The avg is %f\n",a);
int x=5,y=7;
swap(x,y);
cout<<x<<endl<<y;
}
```

Output:

```
The avg is 3.7
7
5
```

b) TempClass.cpp:

```
#include <iostream>
using namespace std;
template <class T>
class Number {
    private:
      T num;
    public:
      Number(T n) : num(n) {}

    T getNum() {
      return num;
    }
};

    int main() {
    Number<int> numberInt(7);
    Number<double> numberDouble(7.7);
    cout << "int Number = " << numberInt.getNum() << endl;
    cout << "double Number = " << numberDouble.getNum() << endl;
    return 0;
}
```

Output:

```
int number = 7
double number = 7.7
```

Conclusion:

In this week I had learned about:

- How to import methods in a package to another java program.
- How to create a user-defined package in java.
- How to save a user-defined package in a particular location of our system.
- How to create the template for class and function.
- How to declare template class.

Week – 7

Aim: To get familiar with exception handling.

1) Write a program for exception handling in java tries catch finally blocks.

Description:
we are implementing try and catch block to handle the exception. The error code written is in try block and catch block handles the raised exception. The finally block will be executed on every condition.

Program:

```
class ExceptionTest
{
	public static void main(String[] args)
	{
		int a = 40, b = 4, c = 4;
		int result;
		try
		{
			result = a / (b-c);
		}
		catch (ArithmeticException ae)
		{
			System.out.println("Cannot divided by zero."+ae);
		}
		finally
		{
			System.out.println("finally block");
		}
		result = a / (b+c);
		System.out.println("Result: "+result);
	}
}
```

Output:

```
Cannot divided by zero
Finally block
Result: 5
```

2) Write a program for nested try catch blocks.

Description:

When a try catch block is present in another try block then it is called the nested try catch block. Each time a try block does not have a catch handler for a particular exception, then the catch blocks of parent try block are inspected for that exception, if match is found that that catch block executes.

If neither catch block nor parent catch block handles exception then the system generated message would be shown for the exception, similar to what we see when we don’t handle exception.

Program:

```
class NestingDemo{
      public static void main(String args[]){
        //main try-block
        try{
        //try-block2
          try{
             //try-block3
             try{
                 int arr[]= {1,2,3,4};

                 System.out.println(arr[10]);
             }catch(ArithmeticException e){
                 System.out.print("Arithmetic Exception");
                 System.out.println(" handled in try-block3");
             }
          }
          catch(ArithmeticException e){
            System.out.print("Arithmetic Exception");
            System.out.println(" handled in try-block2");
            }
        }
        catch(ArithmeticException e3){
          System.out.print("Arithmetic Exception");
          System.out.println(" handled in main try-block");
        }
        catch(ArrayIndexOutOfBoundsException e4){
          System.out.print("ArrayIndexOutOfBoundsException");
          System.out.println(" handled in main try-block");
        }
        catch(Exception e5){
          System.out.print("Exception");
          System.out.println(" handled in main try-block");
         }
      }}
```

Output:

```
ArrayIndexOutOfBoundsException handled in main try-block
```

3) write a program using User defined exceptions.

Description:

Java user-defined exception is a custom exception created and throws that exception using a keyword ‘throw’. It is done by extending a class ‘Exception’. An exception is a problem that arises during the execution of the program. In Object-Oriented Programming language, Java provides a powerful mechanism to handle such exceptions.

Program:

```
class SampleException{
public static void main(String args[]){
try{
throw new UserException(400);
}
catch(UserException e){
System.out.println(e) ;
}
}
}
class UserException extends Exception{
int num1;
UserException(int num2) {
num1=num2;
}
public String toString(){
return ("Status code = "+num1) ;
}
}
```

Output:

Keyword ‘throw’ is used to create a new Exception and throw it to catch block

Conclusion:

In this week I have learned about following concepts:

1. I have learned about the concept exception handling with java.
2. The core advantage of exception handling is to maintain the normal flow of the application.
3. It really helped me to write a program on Nested try catch blocks and user defined exceptions.

Week – 8

Aim: To Understand and learn the concept of Threads.

1. **Write a program of Creating Threads By Class and Runnable interface.**

Description:
In java using Runnable interface that is to be implemented by a class whose instances are intended to be executed by a thread. There is no need of subclassing a Thread when a task can be done by overriding only run() method of Runnable.

Program:
```
class Mythread implements Runnable {
    public void run() {
            System.out.print("Thread has been created using Runnable
interface...!");
    }
}
public class Threads1 {
    public static void main(String[] args) {
            Thread t = new Thread(new Mythread());
            t.start();
    }
}
```

Output:
```
Thread has been created using Runnable interface...!
```

2. **Write a program of Multi-threading in JAVA.**

Description:
In Java Multithreading refers to a process of executing two or more threads simultaneously for maximum utilization of the CPU. A thread in Java is a lightweight process requiring fewer resources to create and share the process resources.

Program:

```
public class Threads2 extends Thread{
    public void run() {
            for(int i=1;i<5;i++)
            System.out.println(i);
    }
    public static void main(String[] args) {
            Threads2 t1 = new Threads2();
            Threads2 t2 = new Threads2();
            t1.start();
            t2.start();
    }
}
```

Output:

```
Test case-1:                Test case-2:
1                           1
2                           2
1                           3
2                           1
3                           2
4                           3
3                           4
4                           4
```

3. **Write a program of Synchronization in JAVA.**

Description:

in java Synchronization is the capability to control the access of multiple threads to any shared resource. In the Multithreading concept, multiple threads try to access the shared resources at a time to produce inconsistent results. The synchronization is necessary for reliable communication between threads.

Program:

```
class mythread1 extends Thread {
    Symbol t = new Symbol();
    mythread1(Symbol t){
            this.t=t;
    }
    public void run() {
            t.symbol("#");
    }
}
class mythread2 extends Thread {
    Symbol t = new Symbol();
    mythread2(Symbol t){
            this.t=t;
    }
    public void run() {
            t.symbol("@");
    }
}
class Symbol {
    synchronized void symbol(String a){
            for(int i = 0;i < 4;i++)
            System.out.print(a+" ");
    }
}
public class Threads3 {
    public static void main(String[] args) {
            Symbol t = new Symbol();
            mythread1 t1 = new mythread1(t);
            t1.start();
            mythread2 t2 = new mythread2(t);
            t2.start();
    }
}
```

Output:

Test case-1

```
# # # # @ @ @ @
```

Test case-2

```
$ $ $ $ % % % %
```

Conclusion:

In this week I had learned about:

- How Thread works in programming.
- How to Create Threads using Java.
- How to write a Java program using multiple threads.
- How to implement Runnable interface in Java.
- How to handle threads in Java using synchronize keyword.

Week – 9

Aim: To Understand and learn the concept of Event Handling and AWT.

1. **Write a program by Usage of Frames and different AWT components like Buttons, List, Text Boxes Radio Button, Choice, Combo Box Classes.**

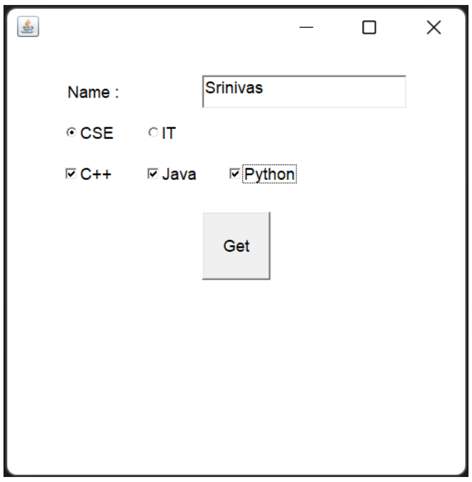
   Description:
   In java AWT components are platform-dependent i.e., components are displayed according to the view of operating system. AWT is heavy weight i.e., its components are using the resources of underlying operating system (OS).

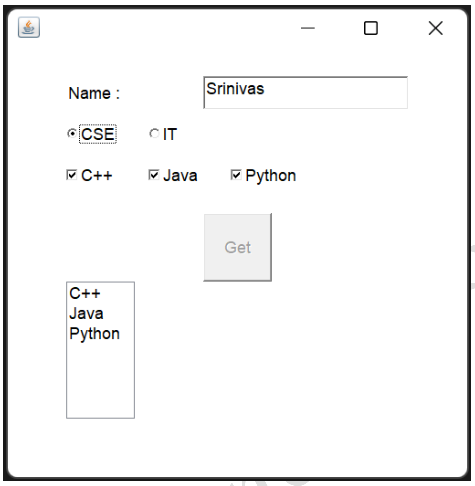
   Program:

```
import java.awt.*;
import java.awt.event.*;
class comps extends Frame implements ActionListener{
    Label l1;
  Button b;
  List l;
    int n;
  Checkbox cpp,j,p,m,f;
  CheckboxGroup g;
  TextField t;
  comps(){
            l1=new Label("Name :");
            l1.setBounds(50, 50, 50, 25);
            t = new TextField();
    t.setBounds(150, 50, 150, 25);
            g = new CheckboxGroup();
    m = new Checkbox("CSE",g,false);
    m.setBounds(50, 80, 50, 25);
            f = new Checkbox("IT",g,false);
    f.setBounds(110, 80, 50, 25);
    cpp = new Checkbox("C++");
    cpp.setBounds(50, 110, 50, 25);
```

```
            j = new Checkbox("Java");
      j.setBounds(110, 110, 50, 25);
            p = new Checkbox("Python");
      p.setBounds(170, 110, 50, 25);
            b = new Button("Get");
      b.setBounds(150, 150, 50, 50);
            l = new List(3);
            l.setBounds(50,200, 50, 100);
      add(l1);add(b);add(cpp);add(m);add(f);add(t);add(cpp);add(j);add(p);
            b.addActionListener(this);
      setSize(350,350);
      setLayout(null);
      setVisible(true);
    }
      public void actionPerformed(ActionEvent e) {
            if(e.getSource()==b){
                  if (cpp.getState()==true){
                        l.add("C++");
                  }
                  if (j.getState()==true){
                        l.add("Java");
                  }
                  if (p.getState()==true){
                        l.add("Python");
                  }
                  add(l);
                  b.setEnabled(false);
            }
      }
}
public class AWT9 {
      public static void main(String[] args) {
            comps a = new comps();
      }
}
```

Output:





```
C++
Java
Python
```

2. Write a program of Different Layouts.

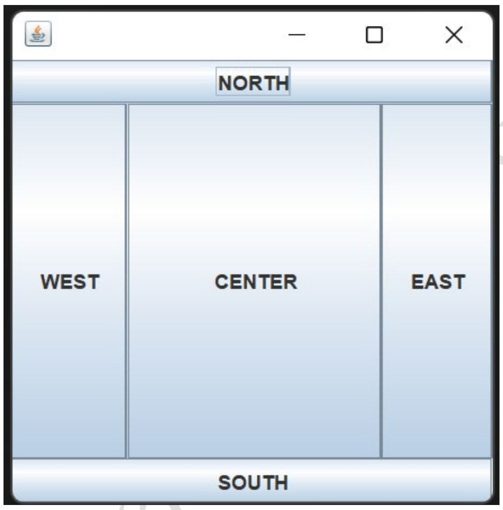
Description:

Android Layout is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen. Generally, every application is a combination of View and ViewGroup.

Program:

```
import java.awt.*;
public class Border {
    Frame f;
    Border() {
            f = new Frame();
            Button b1 = new Button("NORTH");
            Button b2 = new Button("SOUTH");
            Button b3 = new Button("EAST");
            Button b4 = new Button("WEST");
            Button b5 = new Button("CENTER");
            f.add(b1, BorderLayout.NORTH);
            f.add(b2, BorderLayout.SOUTH);
            f.add(b3, BorderLayout.EAST);
            f.add(b4, BorderLayout.WEST);
            f.add(b5, BorderLayout.CENTER);
            f.setSize(300, 300);
            f.setVisible(true);
    }
    public static void main(String[] args) {
            new Border();
    }
    }
```

Output:



3. Write a program by Using Action Listeners like performing calculator operations, interest calculation.

Description:

GUI helps in user interactions using some graphics. It primarily consists of a set of classes and methods that are required for creating and managing the GUI in a simplified manner such as buttons, windows, frame, text field.
I have provided the Java code for the calculator which uses Action listener interface for Event Handling.

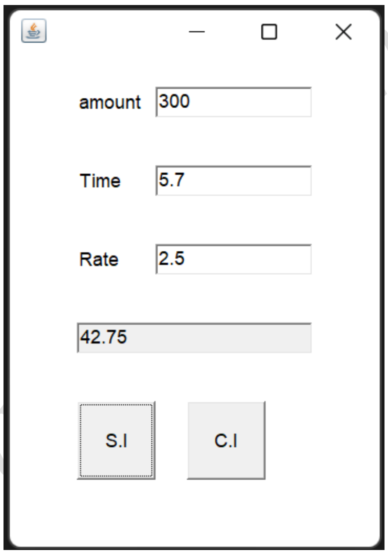
Program:

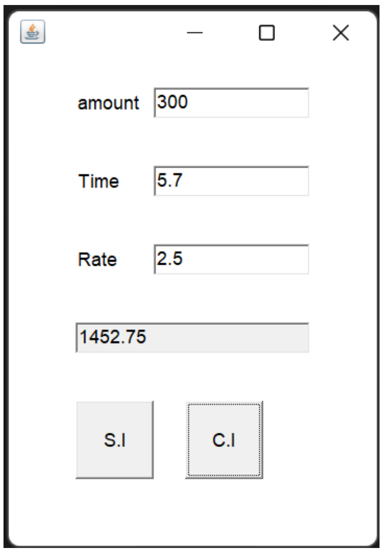
```
import java.awt.*;
import java.awt.event.*;
public class AWT6 extends Frame implements ActionListener{
```

```
Label l1,l2,l3;
TextField tf1,tf2,tf3,tf4;
Button b1,b2;
AWT6(){
        l1=new Label("amount");
        l1.setBounds(50, 50, 50, 20);
        l2=new Label("Time");
        l2.setBounds(50, 100, 50, 20);
        l3=new Label("Rate");
        l3.setBounds(50, 150, 50, 20);
        tf1=new TextField();
        tf1.setBounds(100,50,100,20);
        tf2=new TextField();
        tf2.setBounds(100,100,100,20);
        tf3=new TextField();
        tf3.setBounds(100,150,100,20);
        tf4=new TextField();
        tf4.setBounds(50,200,150,20);
        tf4.setEditable(false);
        b1=new Button("S.I");
        b1.setBounds(50,250,50,50);
        b2=new Button("C.I");
        b2.setBounds(120,250,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        add(l1);add(l2);add(l3);
        add(tf1);add(tf2);add(tf3);add(tf4);add(b1);add(b2);
        setSize(250,350);
        setLayout(null);
        setVisible(true);
}
public void actionPerformed(ActionEvent e) {
        String s1=tf1.getText();
        String s2=tf2.getText();
        String s3=tf3.getText();
```

```
            float p = Float.parseFloat(s1);
            float t = Float.parseFloat(s2);
            float r = Float.parseFloat(s3);
            float s = 0;
            if(e.getSource()==b1){
                s = (p*t*r)/100;
            }else if(e.getSource()==b2){
                s = p*(1+r/100)*t-p;
            }
            String result=String.valueOf(s);
            tf4.setText(result);
        }
        public static void main(String[] args) {
            new AWT6();
        }
    }
```

Output:





Conclusion:

In this week I had learned about:

- This module explored the fundamentals of graphics and howthey are used in Java.

- I have begun the module by becoming acquainted with the JavaAWT (Advanced Windowing Toolkit) and, more specifically, the Graphics class.
- I have learned how to use the Graphics class to draw graphicsprimitives and text.
- In these modules, I have been introduced to images and how they fit into Java applets.
- I had wrapped up the module by building a slide show appletthat demonstrated how to load and draw images.

Week – 10

Aim: To Understand and learn the concept of Listener class.

1. **Write a program of Mouse Listener**

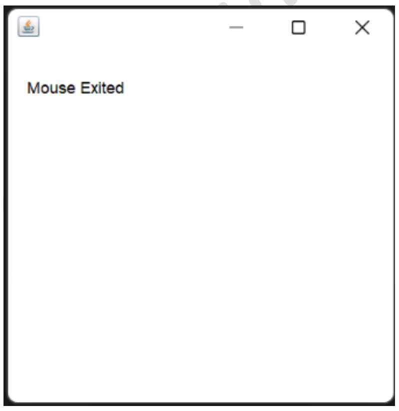
Description:
MouseListener and MouseMotionListener is an interface in java.awt.event package . Mouse events are of two types. MouseListener handles the events when the mouse is not in motion. While MouseMotionListener handles the events when mouse is in motion.

Program:

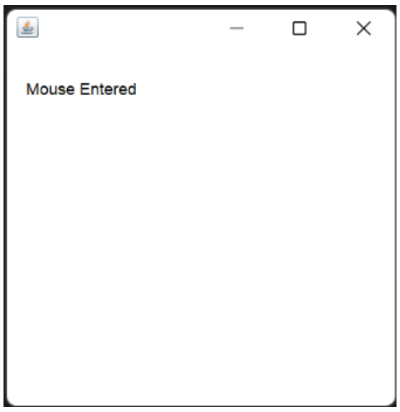
```
import java.awt.*;
import java.awt.event.*;
public class AWT10 extends Frame implements MouseListener{
  Label l;
  AWT10(){
    addMouseListener(this);
            l=new Label();
    l.setBounds(20,50,100,20);
    add(l);
    setSize(300,300);
    setLayout(null);
    setVisible(true);
  }
  public void mouseClicked(MouseEvent e) {
    l.setText("Mouse Clicked");
  }
```

```
    public void mouseEntered(MouseEvent e) {
      l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
      l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
              l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
      l.setText("Mouse Released");
    }
      public static void main(String[] args) {
              new AWT10();
      }
  }
```

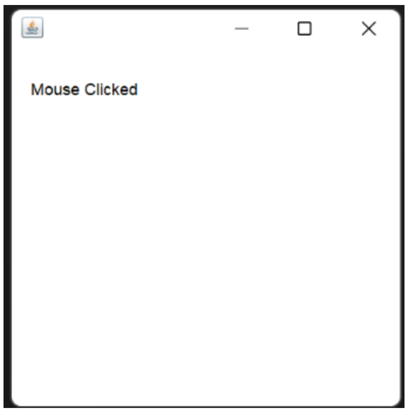
Output:



Mouse Exited



Mouse Entered



Mouse Clicked

2. **Write a program of Keyboard listener.**

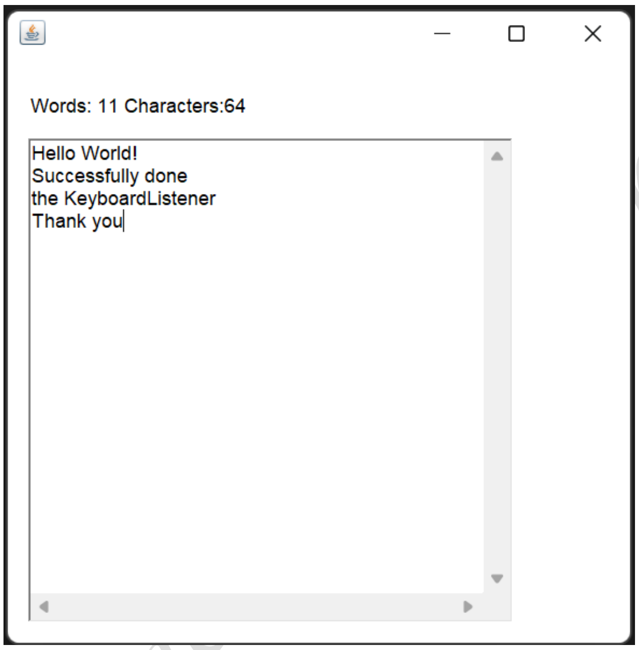
Description: The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The class that is interested in processing a keyboard event either implements this interface (and all the methods it contains) or extends the abstract KeyAdapter class (overriding only the methods of interest). Type an uppercase 'A' by pressing and releasing the Caps Lock key, and then pressing the A key. You should see the following events: key-pressed (Caps Lock), key-pressed (A), key typed ('A'), key-released (A). Note that Caps Lock is not listed as a modifier key.

Program:

```
import java.awt.*;
import java.awt.event.*;
public class AWT11 extends Frame implements KeyListener {
  Label l;
  TextArea area;
  AWT11() {
    l = new Label();
    l.setBounds (20, 50, 200, 20);
    area = new TextArea();
    area.setBounds (20, 80, 300, 300);
    area.addKeyListener(this);
    add(l);
            add(area);
    setSize (400, 400);
    setLayout (null);
    setVisible (true);
  }
  public void keyPressed(KeyEvent e) {}
  public void keyReleased (KeyEvent e) {
    String text = area.getText();
    String words[] = text.split ("\\s");
    l.setText ("Words: " + words.length + " Characters:" + text.length());
  }
```

```
    public void keyTyped(KeyEvent e) {}
    public static void main(String[] args) {
      new AWT11();
    }
  }
```

Output:



```
Words: 11 Characters:64

Hello World!
Successfully done
the KeyboardListener
Thank you
```

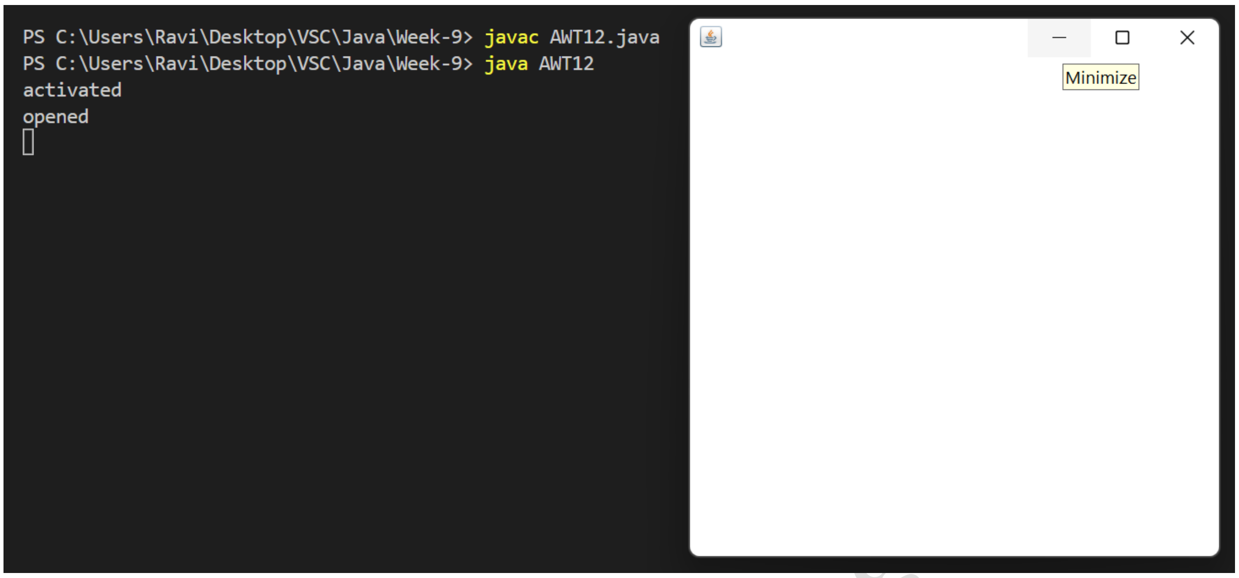
3. Write a program of Window listener

Description: The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. public interface WindowListener extends EventListener.

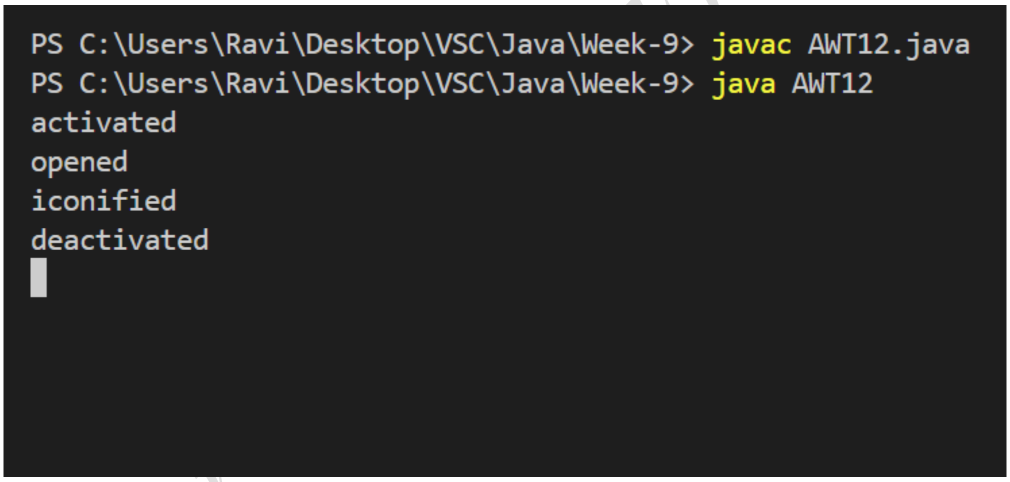
Program:

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class AWT12 extends Frame implements WindowListener {
  AWT12() {
   addWindowListener(this);
   setSize (400, 400);
   setLayout (null);
   setVisible (true);
  }
   public static void main(String[] args) {
   new AWT12();
   }
   public void windowActivated (WindowEvent arg0) {
   System.out.println("activated");
   }
   public void windowClosed (WindowEvent arg0) {
   System.out.println("closed");
   }
   public void windowClosing (WindowEvent arg0) {
   System.out.println("closing");
   dispose();
   }
   public void windowDeactivated (WindowEvent arg0) {
   System.out.println("deactivated");
   }
   public void windowDeiconified (WindowEvent arg0) {
   System.out.println("deiconified");
   }
   public void windowIconified(WindowEvent arg0) {
   System.out.println("iconified");
   }
   public void windowOpened(WindowEvent arg0) {
   System.out.println("opened");
   }
   }
```

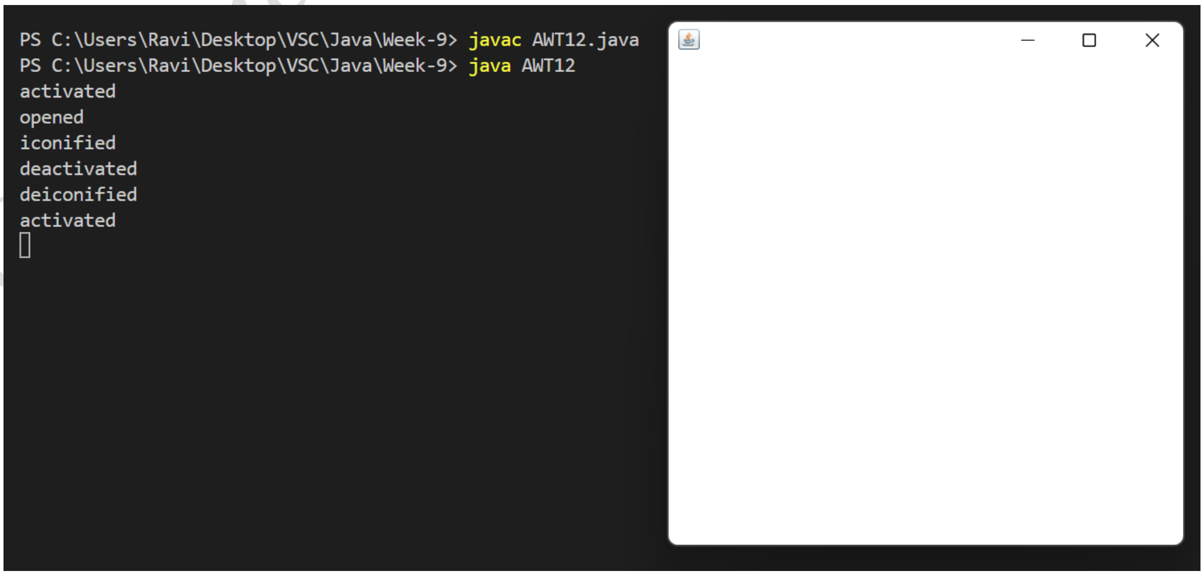
Output:



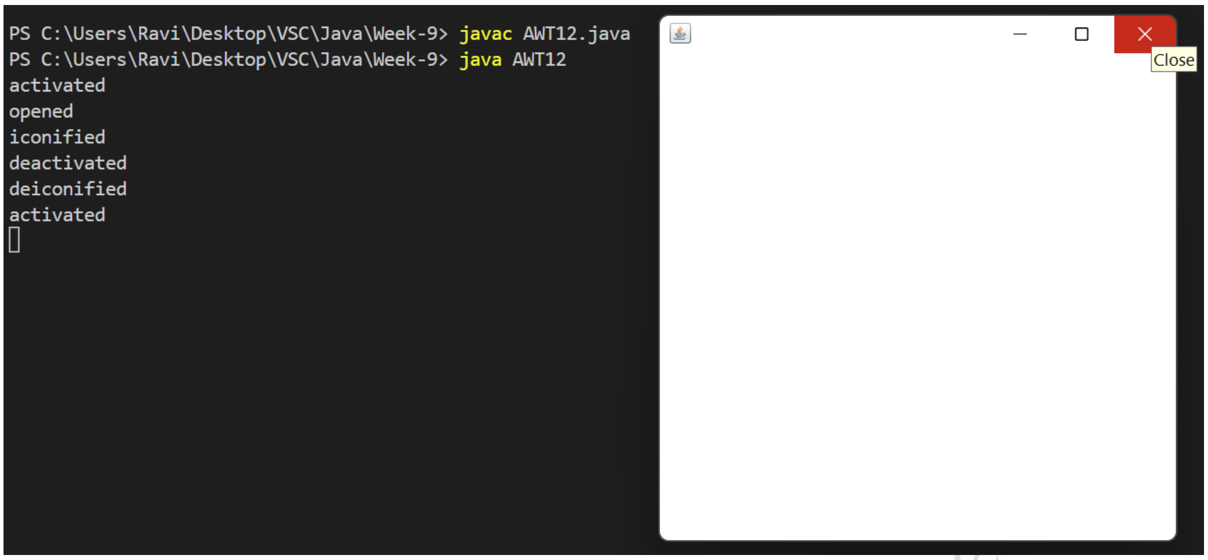
```
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> javac AWT12.java
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> java AWT12
activated
opened
```



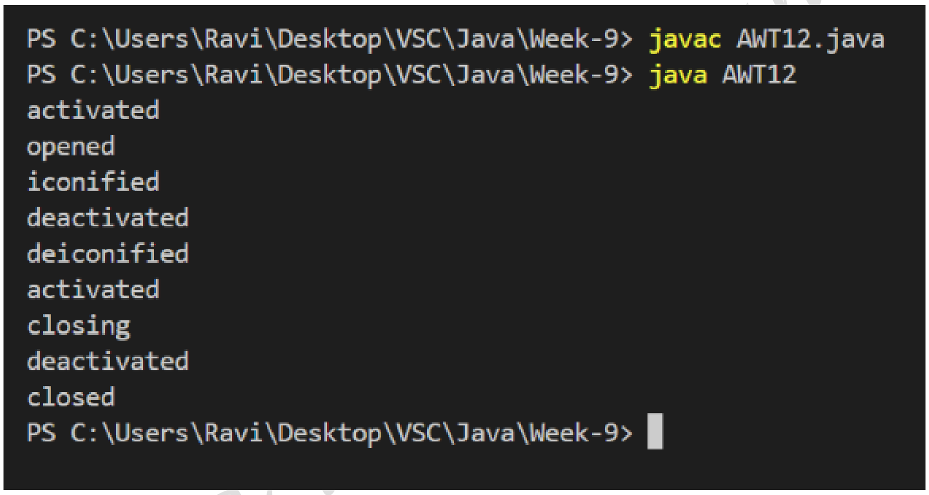
```
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> javac AWT12.java
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> java AWT12
activated
opened
iconified
deactivated
```



```
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> javac AWT12.java
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> java AWT12
activated
opened
iconified
deactivated
deiconified
activated
```



```
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> javac AWT12.java
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> java AWT12
activated
opened
iconified
deactivated
deiconified
activated
```



```
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> javac AWT12.java
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9> java AWT12
activated
opened
iconified
deactivated
deiconified
activated
closing
deactivated
closed
PS C:\Users\Ravi\Desktop\VSC\Java\Week-9>
```

Conclusion:

In this week I had learned about:

- An event listener in Java is designed to process some kind of event it "listens" for an event, such as a user's mouse click or a key press, and then it responds accordingly.
- An event listener must be connected to an event object that defines the event.
- graphical components like a JButton *or* JTextField are known as *event sources*.
- This means that they can generate events (called *event objects*), such as providing a *JButton* for a user to click, or a JTextField in which a user can enter text.
- The event listener's job is to catch those events and do something with them.

WEEK 11

Aim: To Understand the use and get familiar with applets.

1) Write a program Placing different AWT component on APPLET.

Description:

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

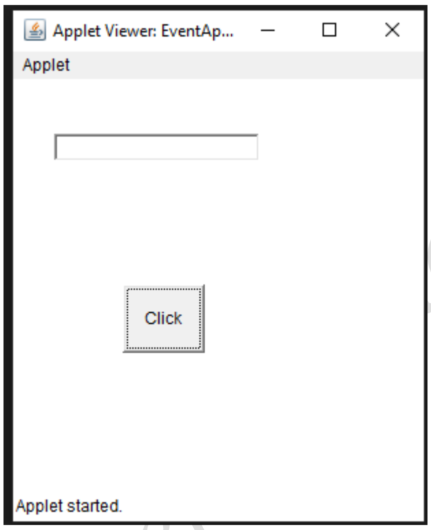
An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.

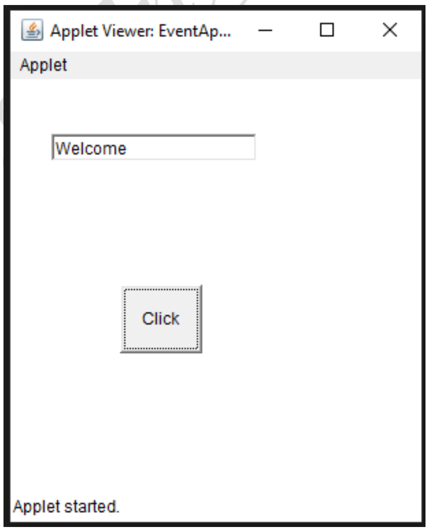
Applets are used to make the website more dynamic and entertaining.

Program:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
    Button b;
    TextField tf;
    public void init(){
            tf=new TextField();
            tf.setBounds(30,40,150,20);
            b=new Button("Click");
            b.setBounds(80,150,60,50);
            add(b);add(tf);
            b.addActionListener(this);
            setLayout(null);
    }
    public void actionPerformed(ActionEvent e){
            tf.setText("Welcome");
    }
}
/*
<applet code="EventApplet.class" width="300" height="300">
</applet>
*/
```

Output:





2) Write a program of Interest calculator using Applet

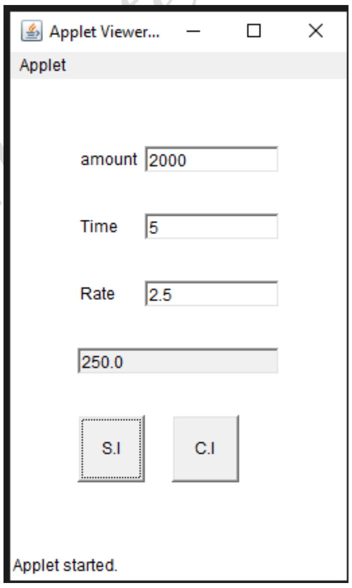
Description:
Applets helps in user interactions using some graphics. It primarily consists of a set of classes and methods that are required for creating and managing the Applets in a simplified manner such as buttons, windows, frame, text field.
I have provided the Java code for the calculator which uses Action listener interface for Event Handling.

Program:

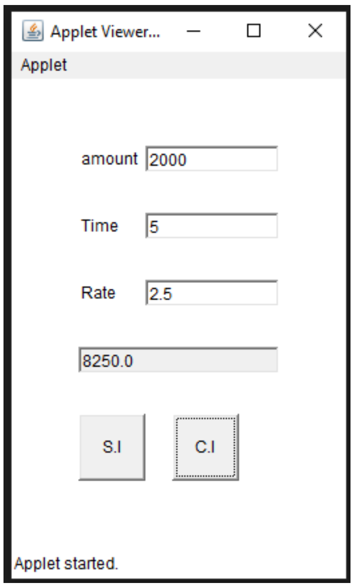
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class InterestApplet extends Applet implements ActionListener{
    Label l1,l2,l3;
    TextField tf1,tf2,tf3,tf4;
    Button b1,b2;
    public void init() {
            l1=new Label(“amount”);
            l1.setBounds(50, 50, 50, 20);
            l2=new Label(“Time”);
            l2.setBounds(50, 100, 50, 20);
            l3=new Label(“Rate”);
            l3.setBounds(50, 150, 50, 20);
            tf1=new TextField();
            tf1.setBounds(100,50,100,20);
            tf2=new TextField();
            tf2.setBounds(100,100,100,20);
            tf3=new TextField();
            tf3.setBounds(100,150,100,20);
            tf4=new TextField();
            tf4.setBounds(50,200,150,20);
            tf4.setEditable(false);
            b1=new Button(“S.I”);
            b1.setBounds(50,250,50,50);
            b2=new Button(“C.I”);
            b2.setBounds(120,250,50,50);
            b1.addActionListener(this);
            b2.addActionListener(this);
            add(l1);add(l2);add(l3);
            add(tf1);add(tf2);add(tf3);add(tf4);add(b1);add(b2);
            setLayout(null);
    }
    public void actionPerformed(ActionEvent e) {
```

```
                String s1=tf1.getText();
                String s2=tf2.getText();
                String s3=tf3.getText();
                float p = Float.parseFloat(s1);
                float t = Float.parseFloat(s2);
                float r = Float.parseFloat(s3);
                float s = 0;
                if(e.getSource()==b1){
                        s = (p*t*r)/100;
                }else if(e.getSource()==b2){
                        s = p*(1+r/100)*t-p;
                }
                String result=String.valueOf(s);
                tf4.setText(result);
        }
}
/*
<applet code="InterestApplet.class" width="250" height="350">
</applet>
*/
```

Output:



```
Applet Viewer...
Applet
amount 2000
Time 5
Rate 2.5
250.0
S.I  C.I
Applet started.
```



```
Applet Viewer...
Applet
amount 2000
Time 5
Rate 2.5
8250.0
S.I  C.I
Applet started.
```

3) Write a program of Graphics class in Applet usage

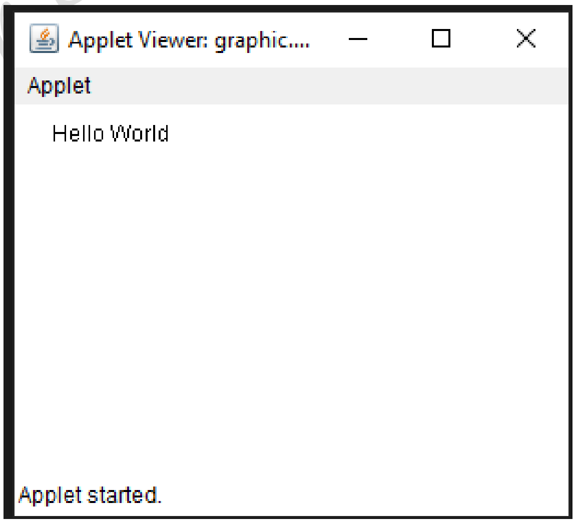
Description:

Graphics class includes methods for drawing many different types of shapes, from simple lines to polygons to text in a variety of fonts.
All the drawing methods have arguments representing end points, corners, or starting locations of a shape as values in the applet's coordinate system.

Program:

```
import java.applet.Applet;

import java.awt.Graphics;

public class graphic extends Applet {

    public void paint(Graphics g) {

            g.drawString("Hello World", 20, 20);

    }

}
```

Output:



```
Hello World

Applet started.
```

Conclusion:

In this week I have learned about:

- How to execute applet programs on cmd.
- Difference between AWT and Applets, and how to perform AWT using Applets.
- Drawing many different types of shapes, from simple lines to polygons to text in a variety of fonts using Graphics class.
- How to use Applets using HTML and decrease the functions of AWT.

Week-12

Aim: To understand and execute different interfaces and their types.

1) Write a program of Collections, Comparator and Iterator interfaces.

Description:

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have.
Comparator interface lets us sort a given collection any number of different ways. Also, this interface can be used to sort any instances of any class.
Iterator enables you to cycle through a collection, obtaining or removing elements. ListIterator extends Iterator to allow bidirectional traversal of a list and the modification of elements.

Program:

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.Iterator;

import java.util.List;

class Student {

    String Name;
```

```
int Age;
public Student(String Name, Integer Age) {
        this.Name = Name;
        this.Age = Age;
}
public String getName() { return Name; }
public void setName(String Name) { this.Name = Name; }
public Integer getAge() { return Age; }
public void setAge(Integer Age) { this.Age = Age; }
@Override
public String toString() {
        return "Customer{" + "Name=" + Name + ", Age=" + Age + '}';
}
static class CustomerSortingComparator
implements Comparator<Student> {
        @Override
        public int compare(Student customer1,Student customer2) {
                int NameCompare = customer1.getName().compareTo(
                customer2.getName());
                int AgeCompare = customer1.getAge().compareTo(
                customer2.getAge());
                return (NameCompare == 0) ? AgeCompare : NameCompare;
        }
}
public static void main(String[] args) {
        List<Student> al = new ArrayList<>();
        Student obj1 = new Student("Ajay", 27);
        Student obj2 = new Student("Sneha", 23);
        Student obj3 = new Student("Simran", 37);
        Student obj4 = new Student("Ajay", 22);
        Student obj5 = new Student("Ajay", 29);
        Student obj6 = new Student("Sneha", 22);
        al.add(obj1);al.add(obj2);al.add(obj3);al.add(obj4);al.add(obj5);
        al.add(obj6);
        Iterator<Student> custIterator = al.iterator();
        System.out.println("Before Sorting:\n");
        while (custIterator.hasNext()) {
            System.out.println(custIterator.next());
        }
        Collections.sort(al,new CustomerSortingComparator());
        System.out.println("\n\nAfter Sorting:\n");
        for (Student customer : al) {
            System.out.println(customer);
        }
    }
}
```

Output:

```
Before Sorting:

Customer{Name=Ajay, Age=27}
Customer{Name=Sneha, Age=23}
Customer{Name=Simran, Age=37}
Customer{Name=Ajay, Age=22}
Customer{Name=Ajay, Age=29}
Customer{Name=Sneha, Age=22}
```

```
After Sorting:

Customer{Name=Ajay, Age=22}
Customer{Name=Ajay, Age=27}
Customer{Name=Ajay, Age=29}
Customer{Name=Simran, Age=37}
Customer{Name=Sneha, Age=22}
Customer{Name=Sneha, Age=23}
```

2) Write a program of List Collections classes.

Description:

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Program:

```
import java.util.*;
class Collection{
    public static void main(String args[]){
            System.out.println("....Using Arrays....");
            ArrayList al1=new ArrayList();
            al1.add("Jack");
            al1.add("Tyler");
            Iterator itr1=al1.iterator();
            while(itr1.hasNext()){
                    System.out.println(itr1.next());
            }
```

```
		System.out.println("\n....Using Linked List....");
		LinkedList<String> al2 = new LinkedList<String>();
		al2.add("Rachit");
		al2.add("Rahul");
		al2.add("Rajat");
		Iterator<String> itr2 = al2.iterator();
		while(itr2.hasNext()){
			System.out.println(itr2.next());
		}
	}
}
```

Output:

```
....Using Arrays....
Jack
Tyler

....Using Linked List....
Rachit
Rahul
Rajat
```

3) Write a program of Map Interface & Map Interface based Collection classes.

Description:

Maps are perfect to use for key-value association mapping such as dictionaries. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys.

Program:

```
import java.util.*;
```

```
class MapEx {
    public static void main(String args[]){
            Map<Integer,String> map=new HashMap<Integer,String>();
            map.put(100,"Amit");
            map.put(101,"Vijay");
            map.put(102,"Rahul");
            for(Map.Entry m:map.entrySet()){
            System.out.println(m.getKey()+" "+m.getValue());
            }
    }
}
```

Output:

```
100 Amit
101 Vijay
102 Rahul
```

Conclusion:

In this week I had learned about:

- Different interfaces and their types especially about Collection interface.
- Map Interface & Map Interface based Collection classes.
- Comparator, Iterator interfaces and how to use them in a generic program.
- Different types of lists in collections i.e., ArrayList, LinkedList, Vector, and Stack.
- Different types of Sets in collections i.e., HashSet and LinkedHashSet.