Application Program Development

Segment : Similarities of Java and C++

Outcomes

- Understanding C++
- Understanding Java
- Similarities between C++ and Java

C++

- First, both (Java and C++) are very successful and popular programming languages.

- C++ is derived from C and has the features of both procedural and object-oriented programming languages.

- The concept of Objects and Class came into existence with C++.

- C++ encapsulates both- low-level and high-level features.

Java and C++

- Both C++ and Java are similar programming languages in terms of various features.
- These languages are helpful for the programming of various apps, operating systems, browsers, and websites.
- Learning these programming languages is quite simple.
- Moreover, the complexity of learning Java and C++ also have a similar level.
- We will discuss some similarities between the languages first

Similarities between Java and C++

→ Syntax ←

```
#include <iostream>

int main(){
      cout<<"Hello C++";
      return 0;
}
```

```
public class Hello
{

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

Similarities between Java and C++

→Comments←

```cpp
#include <iostream>

int main(){
//Execution begins from main

/*This is will print Hello C++
on Console */
      cout<<"Hello C++";
      return 0;
}
```

```java
public class Hello
{
/* Every Java Program starts with a class
nothing stays outside the class*/

  public static void main(String[] args)
  {//Execution begins from main

/*This is will print Hello C++ on Console
*/
      System.out.print("Hello Java");
  }
}
```

Similarities between Java and C++ →Loops & Conditional statements←

```
#include <iostream>

int main(){
      int a = 5, b = 10;
      if(a>b)
            cout << a;
      else
            cout << b;

      return 0;
}
```

```
public class Hello
{
  public static void main(String[] args)
  {
      int a = 5, b = 10;
      if(a>b)
            System.out.print(a);
      else
            System.out.print(b);
  }
}
```

- All loops such as **while, do while, for** and so on are all same
- All **conditional statements** such as **if, else if, else** and **switch** are all same as well.

Similarities between Java and C++ →Arithmetic & Relational Operators←

- The arithmetic operators are same in both languages.

  +, -, /, *

- The relational operators are also same in both languages

  <, >, <=, >=, !=

Similarities between Java and C++
→Basic Data Types←

- **Java Basic Data Types**

byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
boolean	1 bit
char	2 bytes

- **C/ C++ Basic Data Types**

int	2 or 4 bytes
float	4 bytes
double	8 bytes
boolean	1 byte
char	1 byte

Similarities between Java and C++ →Keywords←

- Many of the keywords are same

break, continue, char, int, double, new, public, private, return, static etc.

- Both have multiple-threading support
  - Both allow executing multiple threads (sub-processes) simultaneously to achieve multi-tasking

Question Time…

Choose the OOP characteristics.

- Inheritance
- Polymorphism
- Encapsulation
- Absraction

Inheritance

Abstraction

Function Overriding

Polymorphism

Encapsulation

Operator Overloading

Modularity

Similarities between Java and C++

→Object Oriented Programming←

- **Object Oriented Programming**
- Both Java and C++ support Object Oriented programming.
- OOPs is a modular approach that allows data to be applied within a specific program area.
- It also provides the re-usability feature to develop productive logic, which means that data is prioritized.
- Classes and objects are supported.
- OOPs characteristics include:
  - **Inheritance**
  - **Polymorphism**
  - **Abstraction**
  - **Encapsulation**

Similarities between Java and C++

→Inheritance←

- Process by which objects of one class can link and share some common properties of objects from another class.

```
// Base class
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n" ;
    }
};

// Derived class
class Car: public Vehicle {
  public:
    string model = "Mustang";
};

int main() {
  Car myCar;
  myCar.honk();
  cout << myCar.brand + " " + myCar.model;
  return 0;
}
```

```
class Vehicle {
  protected String brand = "Ford";
  public void honk() {
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";
  public static void main(String[] args) {

    Car myCar = new Car();
    myCar.honk();

     System.out.println(myCar.brand + " " +
                                                      myCar.modelName);
  }
}
```

Similarities between Java and C++ →Polymorphism←

- Allows us to perform a single action in different ways. It is the process of using a function for more than one purpose.

C++

```
class Animal {
  public:
    void animalSound() {
      cout << "The animal makes a sound \n";
    }
};

class Cat : public Animal {
  public:
    void animalSound() {
      cout << "The cat says: Meo Meo \n";
    }
};
class Dog : public Animal {
  public:
    void animalSound() {
      cout << "The dog says: bow wow \n";
    }
};
```

```
int main() {
  Animal myAnimal;
  Cat myCat;
  Dog myDog;

  myAnimal.animalSound();
  myCat
.animalSound();
  myDog.animalSound();
  return 0;
}
```

Similarities between Java and C++ →Polymorphism←

- Allows us to perform a single action in different ways. It is the process of using a function for more than one purpose.

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}
class Cat extends Animal {
  public void animalSound() {
    System.out.println("The cat says: Meo Meo");
  }
}
class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

Java

```
class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();
// Create a Animal object
    Animal myCat = new Cat();
// Create a Cat object
    Animal myDog = new Dog();
// Create a Dog object
    myAnimal.animalSound();
    myCat.animalSound();
    myDog.animalSound();
  }
}
```

Similarities between Java and C++ →Abstraction←

- Includes representing essential features without including the background details.

C++

```
#include <iostream>
using namespace std;
class Summation {
private:
      // private variables
      int a, b, c;
public:
      void sum(int x, int y)
      {
            a = x;
            b = y;
            c = a + b;
            cout<<"Sum of the two number is : "<<c<<endl;
      }
};
```

```
int main()
{
      Summation s;
      s.sum(5, 4);
      return 0;
}
```

Similarities between Java and C++ →Abstraction←

- Includes representing essential features without including the background details.

Java

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}
class Cat extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The cat says: Meo Meo");
  }
}
```

```
class Main {
  public static void main(String[] args)
{
    Cat myCat = new Cat();
    myCat.animalSound();
    myCat.sleep();
  }
}
```

Similarities between Java and C++ →Encapsulation←

- Process of combining data and functions into a single unit.

C++

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

class EncapsulationExample {
 private:
  // we declare a as private to hide it from outside
     int a;
public:
        // set() function to set the value of a
        void set(int x)
        {
                a = x;
        }
        // get() function to return the value of a
        int get()
        {
                return a;
        }
};
```

```
// main function
int main()
{
        EncapsulationExample e1;

        e1.set(10);

        cout<<e1.get();
        return 0;
}
```

Similarities between Java and C++ →Encapsulation←

- Process of combining data and functions into a single unit.

```
public class Person {
  private String name;

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
// Set the value of the name variable to "John"
    myObj.setName("John");
    System.out.println(myObj.getName());
  }
}
```

Java

Similarities between Java and C++ →Static Keyword←

- Static keyword is used for almost the same purpose in both C++ and Java
- **Static Data Members:**
  - Static data members can be defined in both languages.
  - Like C++, static data members in Java are class members and shared among all objects.
- **Static Member Methods:**
  - Static member functions can be defined in both languages with same restirictions.
    - They can only call other static methods.
    - They must only access static data.
    - They can't access **this** or **super**.
    - Like C++, static data members and static methods can be accessed without creating an object. They can be accessed using the class names.

- Easy access of static members is possible, without creating some objects.

Similarities between Java and C++

→ForEach←

- C++ and Java, both uses the foreach loop is used to quickly iterate over the elements of a container (array, vectors, etc.) without performing initialization, testing, or increment/decrement.

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

int main()
{
    int arr[] = { 10, 20, 30, 40 };

    for (auto x : arr)
    cout<<x<<" ";
}
```

```
public class Main {
    public static void main(String[] args)
    {
        int arr[] = { 10, 20, 30, 40 };

            for (var x : arr)
            System.out.print(x+" ");
    }
}
```

Similarities between Java and C++ →Templates VS Generics←

- Generics in Java and the similar in C++ named Template come in handy, when we want to write a code in large scale projects where our program works regardless of the type of data is being passed.

- Function can be templated or generic types as well.

- Both languages support to pass the types at the class level as well.

```
// CPP program to illustrate Templates
#include <iostream>
#include <string.h>
using namespace std;

template <class T>
class TempClass {
		T value;
public:
		TempClass(T item)
		{ value = item; }
		T getValue()
		{ return value; }
};

int main()
{
class TempClass<string>* String =
		new TempClass<string>("Generics vs Templates");
		cout << "Output Values: " << String->getValue()
				<< "\n";
		class TempClass<int>* integer = new
TempClass<int>(9);
		cout << "Output Values: " << integer->getValue();
}
```

```
// Java program to illustrate Generics
public class GenericClass<T> {
		private T value;
		public GenericClass(T value)
		{ this.value = value; }
		public void showType()
		{
		System.out.println("Type:" +
		value.getClass().getSimpleName());
				System.out.println("Value: " + value);
		}
		public static void main(String[] args)
		{
		GenericClass<String> Str = new
	GenericClass<String>("Generics vs Templates");

		GenericClass<Integer> integer = new
						GenericClass<Integer>(9);

				Str.showType();
				integer.showType();
		}
	}
```