# FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

## Lecture39

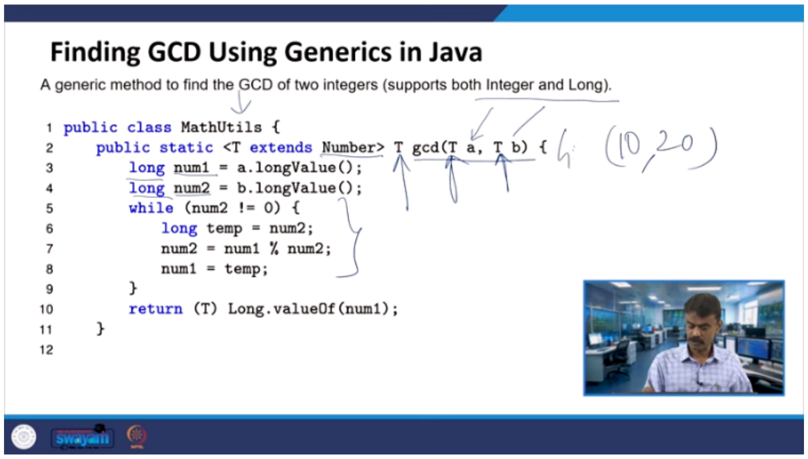
## Lecture 39: Generics in Java [contd.]

Thank you. Welcome to lecture 39 templates and generics. So in the last lecture, we had seen several generic methods in Java, right? So on the continuation, right? So we will see another method which will be calculating GCD, the greatest common divisor of two integers, right?

So let us consider it is going to support both integer and long, right? So now we have the class math utilities. name of the class math utils and t is extending number right interface and we have a method gcd whose return type is t all right. So, a is one number you are going to give. So, let us say whatever be the number and the corresponding is t suppose it is an integer t is integer suppose it is long t is long all right.

So, now you are converting into long all right. So, here number 1 is long and number 2 is long. So, you are converting into long a dot long value Suppose you are giving integer it will be converted into long b dot long value converted into long. So now things are simple if while num 2 not equal to 0 right.

So you are passing let us say I want to find gcd of 10 comma 20 all right. So gcd of 10 comma 20 right. So you are seeing this is number 1 this is number 2 correct. So when it is not equal to 0 right. So number 2 you are assigning to temp correct and then



## Finding GCD Using Generics in Java

A generic method to find the GCD of two integers (supports both Integer and Long).

```java
public class MathUtils {
    public static <T extends Number> T gcd(T a, T b) {
        long num1 = a.longValue();
        long num2 = b.longValue();
        while (num2 != 0) {
            long temp = num2;
            num2 = num1 % num2;
            num1 = temp;
        }
        return (T) Long.valueOf(num1);
    }
```

You are finding out number 1 mod number 2, correct? So, number 1 mod number 2 and then you are assigning whatever be the number you are getting, you are assigning to number 2 and temp you are assigning number 1. So, again you are checking whether number 2, correct? So, number 2 is not equal to 0. Maybe we work it out, GCD of 10 and 20, what will happen, right?

So, number 2, 20 is not 0, true. So while loop is going inside, so now temp is nothing but 20, right? What will be your number 2? Number 2 will be number 1 mod 20, right? 10 mod 20, 10 mod 20 is 10.

So number 2 is becoming 10, and number 1 is now becoming temp, which means 20. I am just swapping initially, right? So now number 2 is 10, which is not 0, right? Number 2 is 10, which is not 0, right? So temp equals number 2, right?

So temp is equal to number 2. So what will happen? Temp is equal to 10, second iteration. And number 2 will become number 1 mod 10. So what is 20 mod 10, correct?

So number 1 is here, 20 mod 10. What is 20 mod 10? 0, right? 20 mod 10 is 0. 2 times 10 is 20, remainder 0.

So remainder 0. Number 2 is 0 now. And what is your number 1? 10 right so now while loop you are coming while number 2 not equal to 0 false right reason is number 2 is already became 0 right and then you are coming out printing the long value of number 1 number 1 is 10 so that means GCD of 10 and 20 is 10 greatest common divisor of 10 and 20 is 10 right we have taken a simple example this is how The loop is working and now what we can do we will go to the main program correct.

So, in the main you have integer right integer a equal to 56 and b equal to 98 right integer a is equal to 56 b equal to 98 and long x this is a long value 1, 2, 3, 4, 5, 6 up to 9 and 9, 8, 7 up to 1 l stands for the long value. So now I am finding GCD of 56 and 98, right? So GCD of AB, I am passing 56 and 98, right? So this is nothing but integer now. Your T is integer.

Capital T is becoming integer. And A and B values are nothing but 56 and 98. So when you are passing this, now this integer value is becoming long. Number one is that A dot long value is converting. It is a function.

It is a method. A is invoking the method, an inbuilt method, long value. That is under number. Correct? And another one is B, long value.

So, both are becoming long. 56 and 98 are now long. And then you are doing the same algorithm that I have explained for 10 and 20. So, then it will return the GCD. Right?

So, similarly, it is a big number. Already it is long. So, now, I mean even the conversion of number 1 and number 2 will become long. And more importantly, what is t now? Long, right?

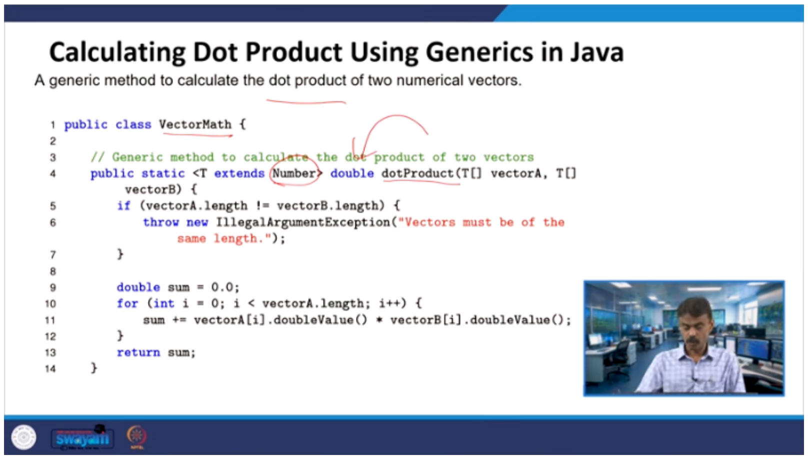
So, t is long. So, here you have long. So, t is long, right? In fact, capital L, that is what we have written here, right? So, it is a long number.

Number will take care. So, capital L, all right? So, this number will take care. So, you are converting into the usual long, all right? And then you are running the algorithm for this big number, x and y. So, we know that that will give you

the gcd so that will be returned right so when i am calling this that will be returned so we will see the code how this is working we will see the code so i will go to the java console and you can see here right so the long value so when you are passing 56 and 98 the gcd of 56 and 98 is 14 right and gcd of 1 2 3 4 5 6 7 8 9 and 987654321 is 9 okay. So, this is how we can calculate and more importantly the concept point of view. So, one place it is becoming integer t becomes integer in another case t is becoming long, right. So, this is how the generic method is working.

Now, we have seen several examples, right. I hope by now you are familiar with the generics. We will see one more program, right. So, interestingly, we are going to see many mathematical programs.

So, here you are going to find out the dot product of two numerical vectors, right. So, this generic method is for the calculation of the dot product of two numerical vectors. So, I have a vector math class, and T is extending the number, right. So here, you have dot product as the name of the method, the return type is double, right? The return type is double. You have an array of T, right? An array of T, and you call it vector A, and an array of T you call vector B. So, two vectors—let us see what the arrays are, whether it is an integer array or a double array. We will see that, right.



## Calculating Dot Product Using Generics in Java

A generic method to calculate the dot product of two numerical vectors.

```java
public class VectorMath {

    // Generic method to calculate the dot product of two vectors
    public static <T extends Number> double dotProduct(T[] vectorA, T[]
         vectorB) {
        if (vectorA.length != vectorB.length) {
            throw new IllegalArgumentException("Vectors must be of the
                 same length.");
        }

        double sum = 0.0;
        for (int i = 0; i < vectorA.length; i++) {
            sum += vectorA[i].doubleValue() * vectorB[i].doubleValue();
        }
        return sum;
    }
```

So now, both the lengths should be the same. So when I am doing the dot product, suppose I have 1, 2, 3, 4. Another one is 5, 6, 7, 8, right? What is the dot product? 1 multiplied by 5, right?

1 into 5 is 5, plus 2 into 6 is 12. Let us say these are tuples in the vector, right? So, 2 into 6 is 12, plus 3 into 7 is 21, and then 4 into 8 is 32. So, add all this, right? So, this is called the dot product.

So, suppose the lengths are not equal. Assume that one length is 1, 2, 3, and another one is 5, 6, 7, 8. You cannot do it. So, that is what you are checking. If vector A's length is not equal to vector B's length, it throws an exception.

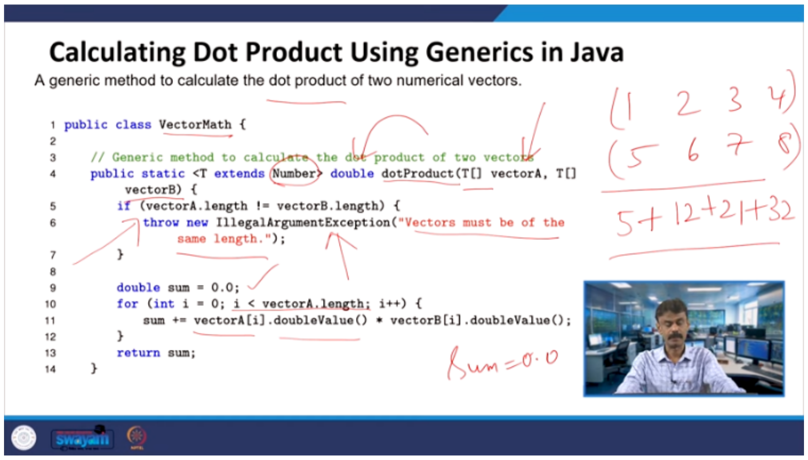
It throws the exception. It is an inbuilt exception: illegal argument exception. So, 'Vectors must be of the same length' will be printed. Assume that you are giving the wrong input. One is 1, 2, 3, and another one is 5, 6, 7, 8.

right, one is having only three components, another is four components. So, they are not, length is not same. So, that will throw an error. Otherwise, right, suppose, right, if it is coming after if means in the else part kind of, right, you do not need to put else, it is understood. That means vector a dot length is equal to vector b dot length, right.

Now, sum is equal to 0.0. Initially, we are initializing and for int i is equal to 0, i less than vector a dot length Any one length. Now both are equal. I plus plus.

Right. So sum is equal to whatever I said. Initially sum is 0. Right. Sum is 0.

So vector A of I. Right. Dot double value. Right. So that means it is converting into double value. Right.



## Calculating Dot Product Using Generics in Java

A generic method to calculate the dot product of two numerical vectors.

```java
public class VectorMath {

    // Generic method to calculate the dot product of two vectors
    public static <T extends Number> double dotProduct(T[] vectorA, T[]
         vectorB) {
        if (vectorA.length != vectorB.length) {
            throw new IllegalArgumentException("Vectors must be of the
                 same length.");
        }

        double sum = 0.0;
        for (int i = 0; i < vectorA.length; i++) {
            sum += vectorA[i].doubleValue() * vectorB[i].doubleValue();
        }
        return sum;
    }
```

$(1 \quad 2 \quad 3 \quad 4)$

$(5 \quad 6 \quad 7 \quad 8)$

$5+12+21+32$

$Sum = 0.0$

The number. Under number. It will be converting into double value. Right. For example here.

It is nothing but 1 into. It is 1.0. Right. It is nothing but 1.0. Even though you are giving 1.

That is what this method will do. So, meaning 1.0 is getting multiplied with the 5.0 vector by dot double value, right? So first is sum, that means sum is becoming 5.0 first when you are doing this 0 plus this is 5.0. Next, when i becomes 1, this component will be multiplied, that is what you are doing, and then i becomes 2, 3 into 7, i becomes 4, 4 into 8. I have just taken an example, and then it will return sum, correct. So now we will see in the main what all the vectors we are using, okay?

So now, I hope you understood, right? Return sum means this particular method, correct? The name of the method is dot product, right? This is returning, alright? This is returning sum. So now, in the same class, it is a driver class, in fact, you have the main method. So, in the main method, you have two vectors: one vector A, which is an integer that contains 1, 2, 3, and another one is vector B, which is also an integer array, right? 4, 5, 6, fine. And then you have two double arrays, vector C and vector D, okay.

So we have four arrays. So two are integer arrays, vector A and vector B, and vector C and vector D are nothing but two double arrays. So now let us call the function, right. Let us call the method. So, System.out.println and I am printing dot product int vectors.

So the dot product you are calling by passing vector A and vector B. So now this t is becoming what? Integer. So the t is becoming an integer. So when I put the notation like this, when I put the notation like this, it is nothing but an integer array. t is becoming an integer.

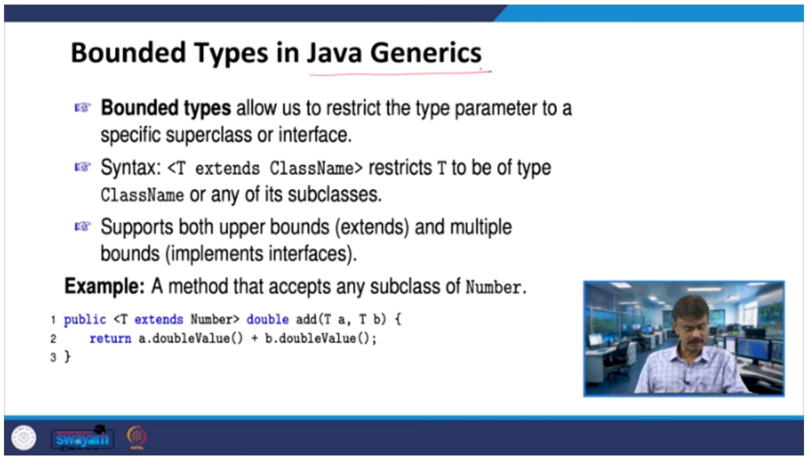
So, integer array. So what are you doing? You are passing vector A. So your local variable, you have vector A. And you are passing an integer. Vector B and here local variable is also vector B, right? So, vector A and vector B you are passing.

So now both are the same length. So this is ruled out and it is coming over here as usual, right? So what it will do: 1 into 4, in fact 4.0, right? You are converting plus 2 into 5, 10.0, right? Then 3 into 6, 18.0, so 18 plus 10, 28, 28 plus 4, 32.0 is your first output, right? So dot product int vectors is 32, right. Next one is you are passing dot product, you are passing vector C and vector D, right? So when you are passing vector C and vector D, what will happen?

It is copying vector C will be copied into local variable vector A of this method and vector D will be copied into vector B and then again the product right and this will be executed both are same length. So this will be executed 1.1 into 4.4 plus 2.2 into 5.5 plus 3.3 into 6.6 right. So, let us see because you need some calculation. So, that particular output you will get. So, let us consider in the program.

Let us go to the program and we will see here you have the same. So, first one I have written 32.0. This is what I wrote because it is easier to calculate for me. Next one is a numerical. 1.1 into 4.4 plus 2.2 into 5.5.

plus 3.3 into 6.6 right you verify you cross verify this you should get 38.72 okay so now the important thing is so here you have right the method you can see t one time it is becoming integer another time it is becoming double so we have seen several programs right with respect to java generics Right. So we had seen methods as well as classes. Right. So now we have the advanced type called bounded types.


## Bounded Types in Java Generics

- **Bounded types** allow us to restrict the type parameter to a specific superclass or interface.
- Syntax: `<T extends ClassName>` restricts `T` to be of type `ClassName` or any of its subclasses.
- Supports both upper bounds (extends) and multiple bounds (implements interfaces).

**Example:** A method that accepts any subclass of `Number`.

```
public <T extends Number> double add(T a, T b) {
    return a.doubleValue() + b.doubleValue();
}
```

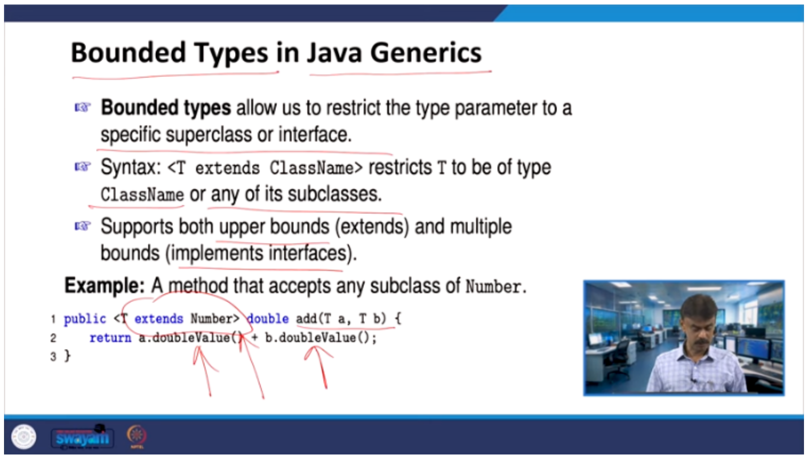
Right. What do you mean by bounded types in Java generics? So bounded types, this allow us to restrict the type parameter to a specific superclass or interface. So we are moving from the subclass to superclass or subclass to interface. So when we are using this, we have used several times.

So this particular syntax is saying that so we are restricted to be a type class name, particular class name or any of its subclasses. Whereas the bounded types, so when I am talking about the bounded types which is supporting both upper bounds and multiple bounds. Upper bounds in the sense extends. T extends class name, right? So that means we are going to a type parameter to a specific superclass or interface, right?

And multiple bounds. So multiple bounds means it is implementing interfaces, right? So that means bounded types which is supporting both upper bounds and the multiple bounds. So, we will see an example right. So, here we have one example t extends number let us assume we are adding right two entities a is under t and b is under t. So, t may be anything right this we know if it is a integer or it may be a double it may be a float.

So, we are converting into double and we are adding both right. So, that is what we are doing over here. So, let us see right some more example all right. So here this is a simple example. Correct.

So the syntax T extends number. So this is restricting T to be of type that particular class name. The class name is nothing but add. Right. Or it may be of one of its subclass.


## Bounded Types in Java Generics

- **Bounded types** allow us to restrict the type parameter to a specific superclass or interface.
- Syntax: `<T extends ClassName>` restricts `T` to be of type `ClassName` or any of its subclasses.
- Supports both upper bounds (extends) and multiple bounds (implements interfaces).

**Example:** A method that accepts any subclass of `Number`.

```java
public <T extends Number> double add(T a, T b) {
    return a.doubleValue() + b.doubleValue();
}
```

Any of its subclass. Correct. So. Let us see how, I mean, this bounded types is being supported, right? Both extends and implements, right?

So, when I write 'T extends number', right? So, you are restricted to T, which extends number, correct? So, that means I can use the superclass or the interface of number, right? So, that is the meaning. So, let us go slightly advanced.

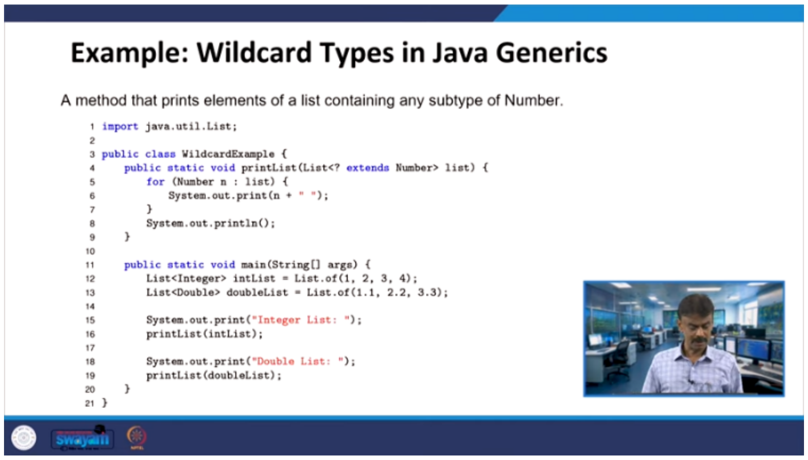
Wildcard types in Java generics. Right, so wildcard we can use the question mark for more flexible generic methods, right? Or we can have the class definition by allowing an unknown type to be specified with the question mark, right? So far, we have not seen the question mark. So, you can see in this particular example, the question mark extends number, right? So, it is of any type, right? Unbounded wildcard question mark, right? And when I write 'question mark extends number', which is nothing but an upper bound. Now, I am talking about upper bound, right? Extends T, which is nothing but an upper-bound wildcard or any subtype of T, right? So, assume that Let us assume when we are using this question mark, right? So, it is as such.

Previously, we talked about T, right? Previously, we talked about T, right? So, when I have this, we are writing 'extends number', right? The question mark extends number. So, your number is nothing but T.

Assume that the number is nothing but T. That is what we are telling. So that means this can be any subtype of T. That means any subtype of number and we call this as a upper bound wildcard. And suppose you are writing one more syntax question mark super T. So that is any super type of T and which is called lower bounded wildcard. So in this example this is upper bounded wildcard. This is an example of the upper bounded wildcard.

So question mark extends number. So question mark extends T. T is a number now, right? Which is nothing but upper bounded wildcard. Any subset of the number, okay? So similarly, you can use this syntax.

Question mark super T, right? So let us see this particular example. The same syntax I am using, right? You have public class with the wildcard example, right? This is a class.



## Example: Wildcard Types in Java Generics

A method that prints elements of a list containing any subtype of Number.

```java
import java.util.List;

public class WildcardExample {
    public static void printList(List<? extends Number> list) {
        for (Number n : list) {
            System.out.print(n + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        List<Integer> intList = List.of(1, 2, 3, 4);
        List<Double> doubleList = List.of(1.1, 2.2, 3.3);

        System.out.print("Integer List: ");
        printList(intList);

        System.out.print("Double List: ");
        printList(doubleList);
    }
}
```

All right. Here you have a void print list. List your question mark extends number. Right. Question mark extends number.

Question mark extends T means upper bound wildcard any subset of T. T is a number. Right. That is the meaning. So it is extending this. And then your list.

All right. Your list. Let us say name of the array. Correct. For number N and list.

So now n and list are going to point out. The same identity as HashMap. Right. And then you are printing in. So that means we are going to take our list.

Correct. So on one blank space. Correct. So the function is over here. Your method is over here.

Start from here to here. Now this is a driver class. While code example is a driver class. You have void main. Right.

You have int list. This is an object. So this object under list. Right. So this object is under list.

of integer right so this already i have imported java utility import list so list of integer you are having one object int list and the int list is nothing but one two three four all right so what we are doing now your list will be nothing but list of integer your list is nothing but list of integer list your extending number so list will become the list of integers wildcard is a wildcard type right and rest of them are similar your nn list is now pointing to the base address or you call it as identity ashmap and 1234 will be printed with this right syntax system dot out dot printer then it is printing 1234 right and then you have double list another object called double list under list of double right so now you can understand that question mark Alright. What it is showing? Question mark extends number.

In one case I am using integer. Another case is I am using double wildcard type. Okay. So and then I have a list of 1.1, 2.2 and 3.3. Double list.

So now I am calling the function. So integer list will be printed. And a print list method I am calling by passing int list. So when I am passing this int list. Now, list is nothing but the integer list 1, 2, 3, 4.

So, now we are mapping to n. So, base address or you call it as identity hash map n and list are same. And then you are printing n. I will print 1, 2, 3, 4. The first output will be 1, 2, 3, 4. I am again calling print list by passing double list. Right.

By passing double list. So double list here you have. Right. So here you have double list. Correct.

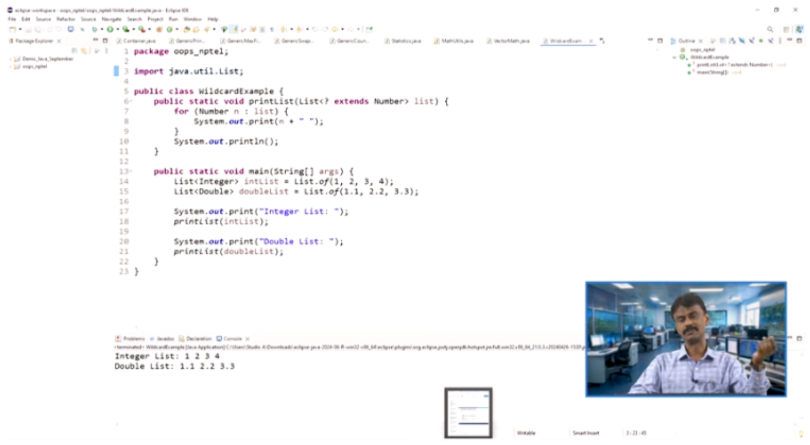
Passing a double list. That means I am passing 1.1, 2.2, and 3.3. Correct. So the list is now becoming a double list. That means both identity hashmaps are the same.

List and double list. And here in line number 5. Correct. In line number 5. So line number 5.

You have number n colon list. So, that means the identity hashmap of n list and double list are all the same. So, what is n now? 1.1 second case, and I am calling this 1.1, 2.2, and 3.3. Anyway, we are asking to print here itself.

So, we should get these two outputs. One output is 1, 2, 3, 4. Another output is 1.1, 2.2, and 3.3. Let us run the code, right. This is what we expected.

1, 2, 3, 4. Yes, we are right. And another one is 1.1, 2.2, and 3.3. So, you can do some manipulations. For example, the dot product we had seen, right.



```java
 1 package oops_nptel;
 2
 3 import java.util.List;
 4
 5 public class WildcardExample {
 6     public static void printList(List<? extends Number> list) {
 7         for (Number n : list) {
 8             System.out.print(n + " ");
 9         }
10         System.out.println();
11     }
12
13     public static void main(String[] args) {
14         List<Integer> intList = List.of(1, 2, 3, 4);
15         List<Double> doubleList = List.of(1.1, 2.2, 3.3);
16
17         System.out.print("Integer List: ");
18         printList(intList);
19
20         System.out.print("Double List: ");
21         printList(doubleList);
22     }
23 }
```

```
Integer List: 1 2 3 4
Double List: 1.1 2.2 3.3
```

So, like this, I mean, you can work it out with the help of the wild type, right. Wildcat type, all right. So, this is what we have done, right. So, it is still enforcing type safety, right? That is very important. In one case, we have used an integer, and another one is a double, right. So, the data type safety is still there.

So, these are all the comparisons in C++ we have templates right and in the case of Java you call it as generics. So, templates are processed that during the compile time right. So, which is creating separate instances for all the types right suppose you are using integer or double or character all right so you have separate instances so we can able to do that whether it is a class or method you can do that and we had seen many flexibility in fact we had seen several cases in particular serial ability and all we worked out all right only difficulty is it will create the larger binaries right so otherwise it is offering lot of flexibility And problem with C++ template is a lack of type safety right.

So, errors can occur. So, when you are converting double to integer, integer to double right or the multiple classes that you are using or multiple data types that you are using right. I mean to say multiple templates you are using. So, in that case so errors may be detected and it will be instantiate rather than the declaration right. So, these are all about C++ template.

Whereas, in the case of Java generics right. So, this is also enforced at compile time, but I mean that will be eased out when you are going in the runtime right. So, that will be taken care. So, compile time we may have a problem, but it will be eased out when you are at the runtime and it is also ensuring the type safety. And it is catching the exception or type errors during compile time.

So here it is less flexible than C++ template. right because you do not have any template specialization right. So, there are some advantages in C++ we had seen right the multiple right multiple classes or multiple templates right. So, these are all some of the advantages we had seen in C++ whereas that is not available in Java. So, templates in C++ so, they are powerful for code reuse right.

but lack in runtime type safety right and it is more flexible and in the case of generics in Java. So, this is ensuring the type safety during compilation time right, but it is lacking some flexibility due to type erasure. So, but both the cases right both the cases I mean generics or templates. So, we have enjoyed that reusability of the code right. So, you can write

let us say template class or template methods. So, whatever be the type whatever be the data type. So, we are able to right reuse it right. So, and then both techniques are very efficient right and we have used right multiple types in both the cases whether it is a templates in C++ or generics in Java. Yeah, each with unique trade-offs.

So, that means they have advantages as well as disadvantages when we compare both of these, right? So, now we have seen generics in Java, right? So, what we can do is we can also talk about generics in Python, right? So, this is the first time we are learning Python, right? This is the 39th lecture, I believe.

Alright, so this is the first time we are talking about Python. As you know, it is an interpreter, right? So, it is an interpreted language, which is a high-level language known for its readability and versatility. Alright, so mainly people use it for machine learning, deep learning, data science, or web development. Python is being used here. Because when we are talking about generics, right? So, we will see the very basic codes as well, right? Before learning or before I talk about generics. We will run some basic code and then we will talk about generics. So, here the theory says it is dynamically typed, right?

So, the types—I mean, the data types—right? So, they are inferred at runtime rather than specified explicitly, right? So, suppose you are using A, A equals 5, alright. So, like int A equals 5, let us say in C++ or Java, it will be automatically understood. That is what this particular line is saying, alright.

So, let us start with Python 3.5 and then type hints and annotations were introduced to improve type clarity and enable generic programming. So, alright. So, before concluding this lecture. So, maybe we can see the small code in Python, right.

I will start with the addition of two numbers. So, let us write this is a Python simple, right. We always start with addition of two numbers are all over. So, define add of a comma b, right. It is a function and then return a plus b, right.

So, define add of a comma b one colon return a plus b. So, now in the result you are calling this function, right, add 3 comma 4. You are passing 3 for A and passing 4 for B. So, 3 plus 4, I have to get 7, right. So, this one example. We will do it in today's class and then I will wind up.

Yeah. So, we will run the code. All right. Python interpreter you are seeing. So, here you go.

The same program what I have explained. Correct. So, when you look at the result add a comma b return a plus b. You are calling the function and putting in result and we are seeing what is result. Right. You can see the output is 7.

So, we are passing 3 and 4. and we are getting the output 7 ok. So, like few basics we will see. So, here we have defined the function right inside the function you are writing the algorithm addition of two numbers return a plus b and then you are calling this the function call you call it as a function call right. So, this is the basic python I am introducing we will see some basic code and then

I will talk about generics in the next class. So, with this, I am concluding today's lecture. Thank you all.