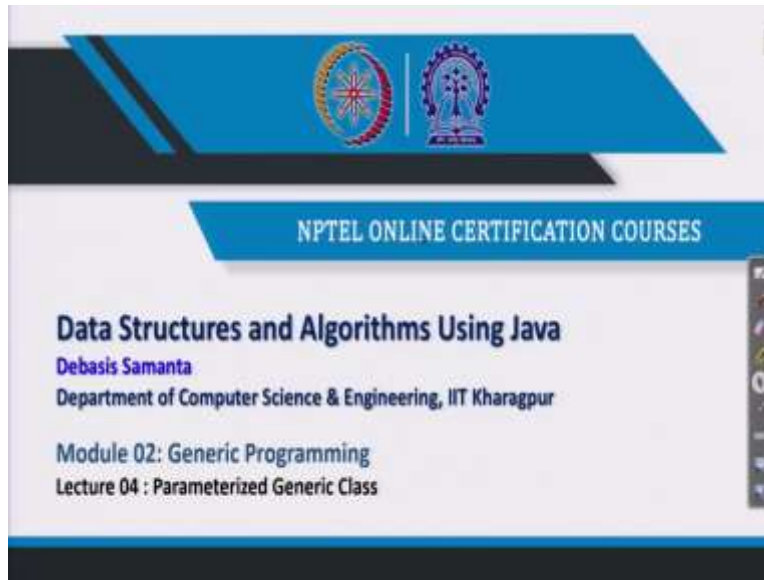


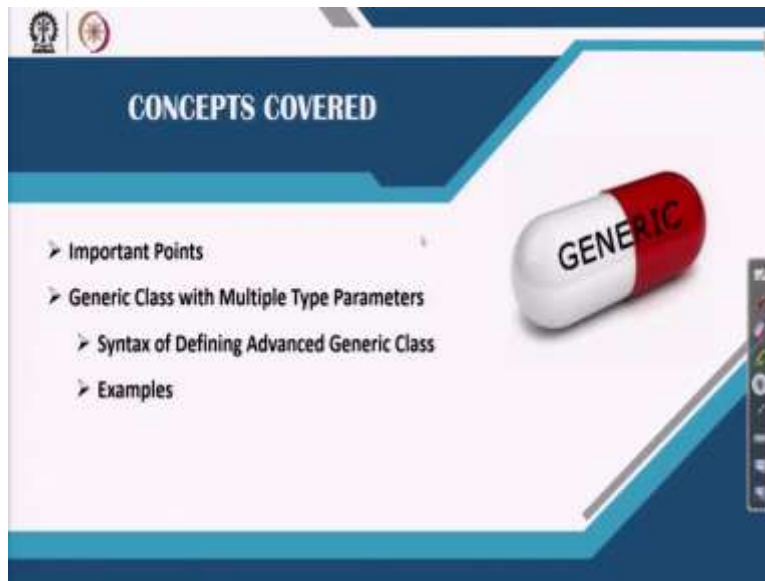
**Data Structures and Algorithms Using Java**  
**Professor Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 04**  
**Parameterized Generic Class**

(Refer Slide Time: 00:33)



We are discussing about generic programming and we have discussed in the last two week, videos how a method can be declared generic, how a class can be declared generic. Next we will discuss about how a class can be declared generic but with multiple type of parameters.

(Refer Slide Time: 00:49)



Before going to start this discussion I want to discuss what are the different points or there are certain points which are very important to note down, so those points will be discussed quickly. And then we will discuss about syntax of defining a generic class with multiple type of parameters, so more than one and will illustrate the concept with examples, so that we can learn this concept.

(Refer Slide Time: 01:21)

**Important Notes**

1. You cannot instantiate an array whose element type is a type parameter. That is following is invalid.

```
T a = new T[5];
```

The reason is that you can't create an array of T is that there is no way for the compiler to know what type of array to actually create.

Now, I just quickly recapitulate certain points which is important to note. The first is it is very good so that it can allow you to take a program in a general ways, not a specific way that means a program can deal with any type of data, but there are few points. If you declare a type as a template this means that you mention that any type it can holds but in that declaration in any parts, in any method you cannot declare a variable of generic type.

For example, this kind of declaration in any method inside the generic class is basically invalid. This is because what actually this is supposed to do, this supposed to do that a is a name of an

object of type T and we want to allocate the memory for this object using new. We want to store 5 different objects into this object arrays. So, a is basically an array of objects whose size is 5, this one.

Now, here actually your compiler will fail to obey this instruction although meaning is like this but it is not possible. This is because compiler cannot allocate the memory at the time of compilation until the actual type of data is known. So, as you have mentioned the template that mean actual type of data is not known, so compiler will not be able to fit.

For example, if say T is for integer then integer needs 16 bits that is fine. So, then if know that it is integer then T is integer, it can work. But suppose T is instantiate for a user defined data type say student which is basically string and floating point values one marks, so it needs may be say 50 bytes.

So, it is not known to at the time of compilation, therefore this is not a valid declaration inside any program, you should not do that, then so this is the one thing that you have to write your program whenever you want to do it.

(Refer Slide Time: 04:20)

**Important Notes**

2. A generic method or any method in a generic class can be declared as static.

**Example 4.1**

```
class GenericClass {
    // Calling generic method to print any data type
    static void display() {
        // ...
    }

    // Calling generic method with generic argument
    void display(T t) {
        // ...
    }

    // Calling generic method with String argument
    void display(String s) {
        // ...
    }

    // Calling generic method with Integer argument
    void display(Integer i) {
        // ...
    }
}
```

Another point that we have learnt about. In a generic class a method if you want to declare it as a generic you can do, that method also can be declared as a static and non-static method, as well as overloaded method all are possible. This example shows how a method can be declared as a

static method as you see this is the main class, we declare is gPrint one method which basically you can pass, here actually we make the method simple but this method is generic and this is a static.

This means that this gPrint method can be called without creating any object of this type. For example, here we after defining this class which includes a generic method and here are the main class, main method which basically use gPrint but this method is called for integer values, this method calls for string and this method is called for floating point values.

So, here the static is basically there we did not create any object of this type. If you create non-static then we have to create first an objects say x generic static x, then x dot gPrint 101 it could be.

But here as we made the declaration static we can have this benefits that objects, without creating an object a method can be called as it is a static. Now, so this is the one and then other things that a method also can be overloaded.

(Refer Slide Time: 06:19)

**Important Notes**

3. A generic method or any method in a generic class can be declared as static.

```
Example 4.2  
class Example4_2 {  
    // See it is to specify class type  
    // of the type of T or Integer,  
    // float etc of the generic class.  
    public static void gPrint(T t) { // A method in the class.  
        System.out.println(t);  
    }  
}  
  
public class Main {  
    public static void main(String args[]) {  
        Example4_2 obj = new Example4_2();  
  
        Example4_2 obj1 = new Example4_2();  
        obj1.gPrint(10);  
  
        Example4_2 obj2 = new Example4_2();  
        obj2.gPrint("10");  
  
        Example4_2 obj3 = new Example4_2();  
        obj3.gPrint(10.5);  
  
        // Printing the value ...  
        printObj: // Printing the obj1  
        printObj: // Printing the obj2  
        printObj: // Printing the obj3  
    }  
}
```

Now, here is another example that how a generic method in a generic class also can be declared. In the previous example there we did not make the class generic, but here in this example we make the class as generic so T. And then the method this one as a generic. So, this means that it, this method can, also can call for this type T or any other template type also it will work.

So, it is generic class that means this class can hold for any type of objects. Inside it a method which is again declare here static but it can call for any types. Now, this is an example here, after declaring this one the generic class which includes a generic method, the driver class to give a demonstration of this.

(Refer Slide Time: 07:29)

**Important Notes**

3. A generic method or any method in a generic class can be declared as static.

```
class GenericClassT {
    // This is a generic class type
    T arr;
    GenericClassT(T arr) {
        // An object of type T is received
        // constructor of the generic class
        this.arr = arr;
    }
    // This is a generic method
    public void static print(T arr) {
        // A method of the class
        System.out.println(arr);
    }
}

class GenericClassDemo {
    public void main(String args) {
        GenericClassT<Integer> g1 = new GenericClassT<Integer>(new Integer[] { 1, 2, 3, 4, 5 });
        GenericClassT<String> g2 = new GenericClassT<String>(new String[] { "a", "b", "c" });
        GenericClassT<Double> g3 = new GenericClassT<Double>(new Double[] { 1.1, 2.2, 3.3 });

        // This is the main method
        print(g1); // Printing the array a
        print(g2); // Printing the array b
        print(g3); // Printing the array c
    }
}
```

So, main method and here you see how we create the three different executions, one this is a for an array of integer objects, this is another (instan) execution for string and this is another execution for double values. Then the print method can be called for any type because it made generic this for string and this is for double. So, this way the static as well as generic class also can work. So, these are the three points that we have to discuss it.

(Refer Slide Time: 08:32)

**Important Notes**

4. In parameter type, you can not use primitives type like int, char, double, etc. Only class can be referred as the template data.

```
class GenericClass {
public:
    GenericClass(T t) {
        data = t;
    }
    void PrintData() {
        cout << t << endl;
    }
};

int main() {
    GenericClass obj1(1);
    GenericClass obj2(2);
    GenericClass obj3(3);
    GenericClass obj4(4);
    GenericClass obj5(5);
    GenericClass obj6(6);
    GenericClass obj7(7);
    GenericClass obj8(8);
    GenericClass obj9(9);
    GenericClass obj10(10);
    return 0;
}
```

And another point also we can just here that a method can be declared as overloaded, I will come to that example but before going to this, one again important point I am just (( ))(8:37) it again and again. That in parameter type that is for the template, in parameter type that is for template T, you should always call this objects for only type of objects but not of other type call primitives like this one. That means here you cannot create int or char or double like, this is basically as you see.

(Refer Slide Time: 09:17)

**Important Notes**

4. In parameter type, you can not use primitives type like int, char, double, etc. Only class can be referred as the template data.

```
class GenericClass { // The <T> to specify class type
    T obj; // An object of type T is declared
    GenericClass(T obj) { // An object of type T is declared
        this.obj = obj;
    }
}

class GenericClass2 {
    public void testMethod() {
        GenericClass2 obj1 = new GenericClass2(1); // OK
        GenericClass2 obj2 = new GenericClass2(2); // OK
        GenericClass2 obj3 = new GenericClass2(3); // OK
        GenericClass2 obj4 = new GenericClass2(4); // OK
        GenericClass2 obj5 = new GenericClass2(5); // OK
    }
}
```

Here in this example this is not okay error because of this one, it cannot be. Here also this is not okay because of this one, this is not valid, not legitimate. On the other hand this is okay because of this, this is also okay and this is also okay because this. So, this is the one important point that you should know and you should remember when you work with generic class.

(Refer Slide Time: 10:06)

**Important Notes**

5. A generic class can be defined with multiple type parameters.

```
class GenericClass { // The <T, U> to specify class type
    T obj1; // An object of type T is declared
    U obj2; // An object of type U is declared
}

class GenericClass2 {
    GenericClass2(T obj1, U obj2) {
        this.obj1 = obj1;
        this.obj2 = obj2;
    }
}
```

Now, I will discuss about another point which basically multiple type parameters while discussing generic class. Now, this is an example that you can note, this is a generic class



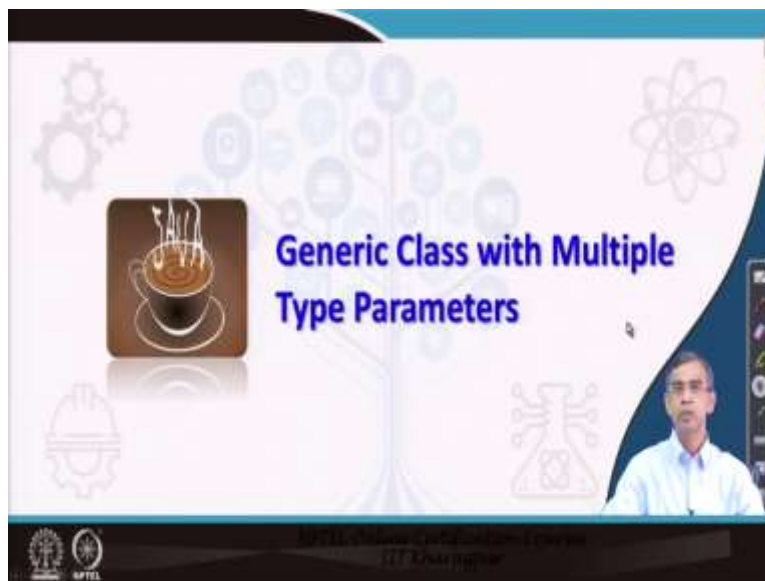
discussion but here we discuss in this example two different templates. So, T and V that means any two type can be holds here.

Now, rest of the things are similar but as we have to deal with two types so there will be two different what is called the data to be considered T type and V type so objective one it can be a single value or it can be a array of elements, no issue this is also either single or any arrays it can be, but we have to discuss.

And then one constructor needs to be declare which basically initialize the elements for this class. So, these are the basically constructor that can initialize. So, you have learned about same way of extending the concept for more than one type of parameter.

The same concept can be again extended 2, 3, 4 any type, so here list of types should be mentioned separated by comma, so T, U, V, W, T, comma, U, comma, V, comma W, like any types, any numbers. So, for Java development programming is concerned there is no limit, maximum up to 256 number of arguments that you can be passed, beyond 256 the Java Virtual Machine will not allow you to do that.

(Refer Slide Time: 12:07)



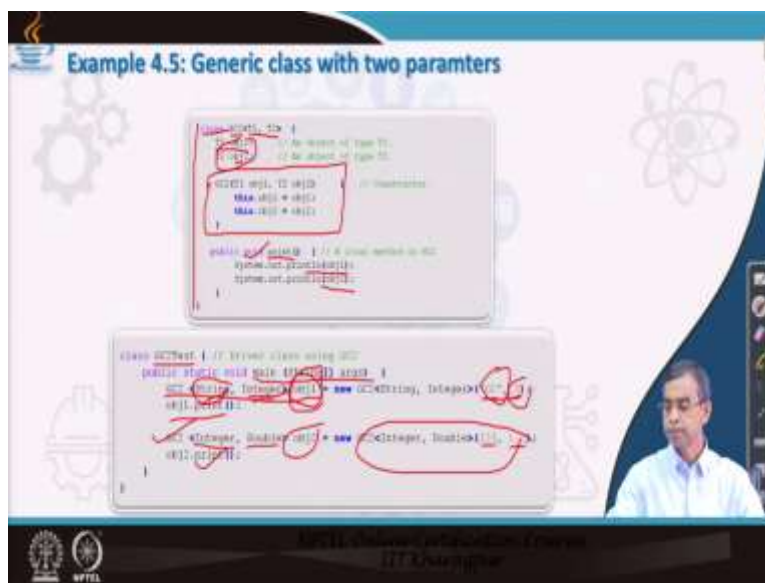
Now, let us have an example about this generic class with multiple type parameters.

(Refer Slide Time: 12:13)



So, what in summary we can say that, we can create a generic class with only one type or any type of parameters, so that more than one type of parameters that can be managed by your generic program.

(Refer Slide Time: 12:34)



Now, let us illustrate this concept with an example. The example is very simple, easy to understand, little bit you have to closely watch the program. In this program, we declare a generic class, the name of the class is GC2, we pass two templates T1 and T2 indicating that two

different types that needs to be consider. It can be integer, it can be string, it can be string, it can be integer or anything absolutely no issue, any order also not an issue.

So, this is the user constructor basically who will initialize, these are the two data that it is in this class and this is the one method, this method will print the value or elements stored in these two objects, so print object **one**, print object **two**. So, this is a very simple one declaration about generic class, you can note that similar concept but only one added types.

Now, the main class showing the execution of this class is GC2Test which includes the main method. Now, here you see we create GC2 objects of type object **one**, the name of the object you can say a, b, c whatever it is there, in this case I just did it for obj**one** no issue. But here you see for this class, for this object obj**one** the first element that means this one is basically of type string, the second this one is of type integer.

Now, when we call this we have to instantiated passing values for initialization, we pass string value GC and integer value 9. This means that this one is an object is created whose two fields are one is string and another is integer value. And the print method, this print method is called here for this little print GC and 9 one by one.

Again in the next execution we create another instance obj2 but here instance is first type is integer, second is double and we passed the values this one as an integer and double and we print it so it will print 123 and 1.2 in this case.

(Refer Slide Time: 15:39)

**Example 4.5: Generic class with two paramters**

```
class GCD<T1, T2> {
    T1 obj1; // An object of type T1
    T2 obj2; // An object of type T2

    GCD(T1 t1, T2 t2) { // constructor
        this.obj1 = obj1;
        this.obj2 = obj2;
    }

    public void print() { // A class method to print
        System.out.println(obj1);
        System.out.println(obj2);
    }
}

class GCDTest { // A driver class using GCD
    public static void main (String[] args) {
        GCD<String, Integer> obj1 = new GCD<String, Integer>("10", 3);
        obj1.print();

        GCD<Integer, Double> obj2 = new GCD<Integer, Double>(11, 1.1);
        obj2.print();
    }
}
```

So, what we have understood here, here is that a generic class with two type of parameters are defined, they can be called for any types whatever it is there string-string, integer-double, integer-string, string-double whatever it is absolutely no problem, not only this.

In addition to any type defined by user like student, book, person also can be placed here and it will work. You can test the same program using another defined class of your own and then run it, you will see it will work for whatever the data type, not only the standard data type like integer, double, string also for user-defined data type. So, this basically works for, if it works for two type I should say that it should also work for any number of types.

(Refer Slide Time: 16:48)

The slide displays the following Java code:

```
class GC2T1 {
    T1 obj1; // An object of type T1
    T2 obj2; // An object of type T2

    GC2T1(T1 t1, T2 t2) { // Constructor
        obj1 = t1;
        obj2 = t2;
    }

    public void print() { // A class method to do
        System.out.println(obj1);
        System.out.println(obj2);
    }
}

class GC2T2 { // A class using GC2T1
    public static void main (String[] args) {
        GC2T1 obj1 = new GC2T1("1", 1);
        obj1.print();

        GC2T1 obj2 = new GC2T1("2", 2);
        obj2.print();
    }
}
```

In that case only the number of types of parameter that you want to mention, you have to write it here and that is all, so this is basically the idea that it will work for you.

(Refer Slide Time: 17:10)

The slide displays the following Java code:

```
public class PairData <T, V> {
    // Declaration of generic type T and V
    // This is a class with two generic parameters
    // Constructor
    public PairData(T t, V v) {
        x = t;
        y = v;
    }

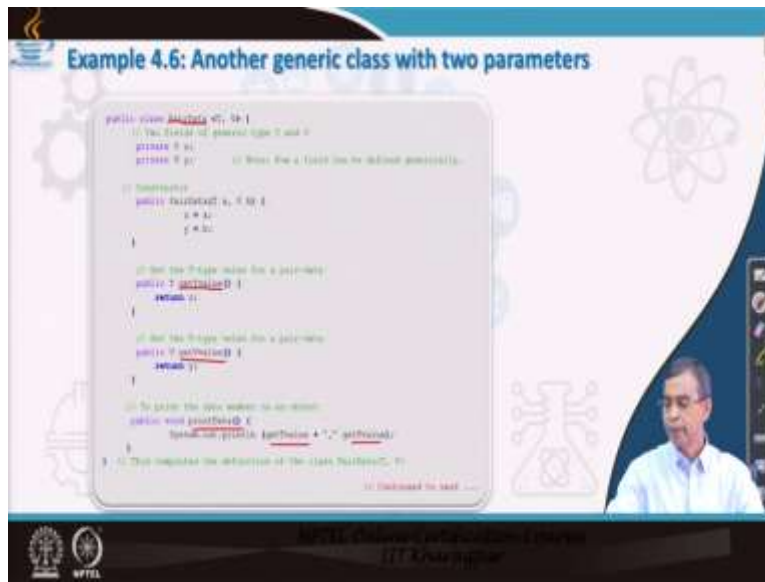
    // Get the T type value for a pair data
    public T getTValue() {
        return x;
    }

    // Get the V type value for a pair data
    public V getVValue() {
        return y;
    }

    // To print the pair value on screen
    public void printPair() {
        System.out.println("PairData: " + x + ", " + y);
    }
} // This completes the definition of the class PairData(T, V)
```

Now, we have learned about generic class with two parameters and this is another example explaining the same thing little bit in a different way, let us check the program, this is another generic class with two parameters, we define these two parameters T, V so this is the name of the generic class PairData, we declare as a private no issue, you can declare public, protected whatever it is no issue here.

(Refer Slide Time: 17:43)

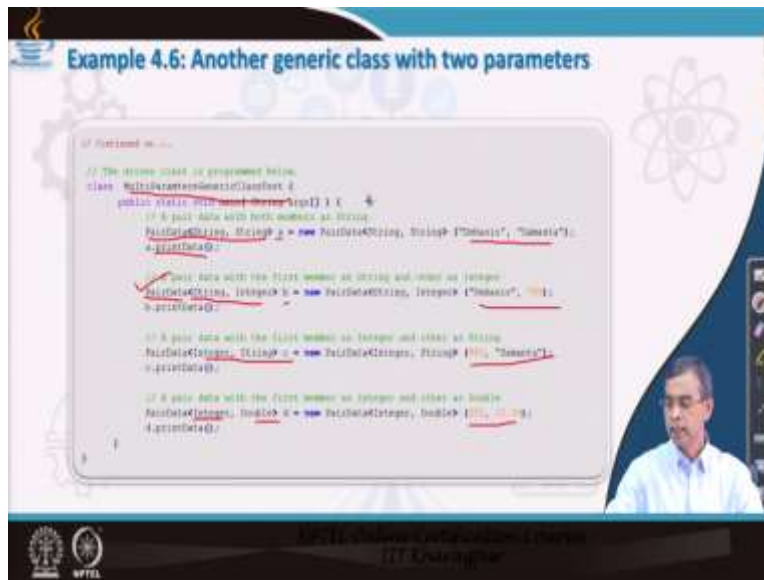


Now, we declare T of type one, V is of another type, they may be same type no issue, but okay, fine and this is a constructor, we pass the value for this type of object T, another value for the type of object V, A and B are the two objects of type T and V. We pass it while we create this generic objects and then instantiate it. So, it is basically x order value it will pass.

Now, this is the one method T getValue and V getValue, so here you note it the getValue and this getValue are the two methods overloaded, this method will return x this data, on the other hand this method will return y, so two methods are there, alternatively okay fine. So, these are the method which basically return the values of x and y which is there for the generic elements.

And printData as we see here is a method which basically print the x component of it and y the y component of it. So, it is basically print the elements which are stored in the generic objects defined by this generic class PairData. So, this program again simple but I repeated intentionally so that you can understand it again with the different view of it, that is okay. Now, let us see how the same program can be utilized in your main.

(Refer Slide Time: 19:34)



So, the driver class it looks like this, again let us see the driver class here. So, it is multi-parameter generic class test demo. Now, here in this example as we see, we create an object, a is the name of the object which includes two elements, the first element is string, the second element is string and then we just pass these are the values of the objects to this a.

So, x is here and y of this object store this value and this print will basically print these are the two elements. Now, in the second example here again we call the objects say let it be b for the type first is of string type, second is of integer type x and y and instantiated this one. The next one here c object is created, first field is integer, second is string we pass it, in the next last one it is integer and double pass it.

So, what we have done here, the generic class that we have declared as a pair data for two types of elements and they can be executed at the time of running that means dynamically passing different value. So, this is a great example the way generic class supports strongly for any type of objects generically and more significantly dynamically, so during the program it can be. So, this way the generic class with two parameters as we have explained here it works for us.

(Refer Slide Time: 21:42)

```
Example 4.7: Generic class with method overloading

// Student.h
class Student {
public:
    Student(); // Name of the student
    Student(int, int, int); // Marks of three subjects
    Student(int, int, int, int); // Marks of four subjects
    Student(int, int, int, int, int); // Marks of five subjects
    Student(int, int, int, int, int, int); // Marks of six subjects
    Student(int, int, int, int, int, int, int); // Marks of seven subjects
    Student(int, int, int, int, int, int, int, int); // Marks of eight subjects
    Student(int, int, int, int, int, int, int, int, int); // Marks of nine subjects
    Student(int, int, int, int, int, int, int, int, int, int); // Marks of ten subjects
};

// Student.cpp
#include "Student.h"
using namespace std;

// Constructor to create student's object
Student::Student() {
    cout << "Student's name: ";
    System.out.println("Name: " + name);
    System.out.println("Score 1: " + score1 + " " + score2 + " " + score3);
}

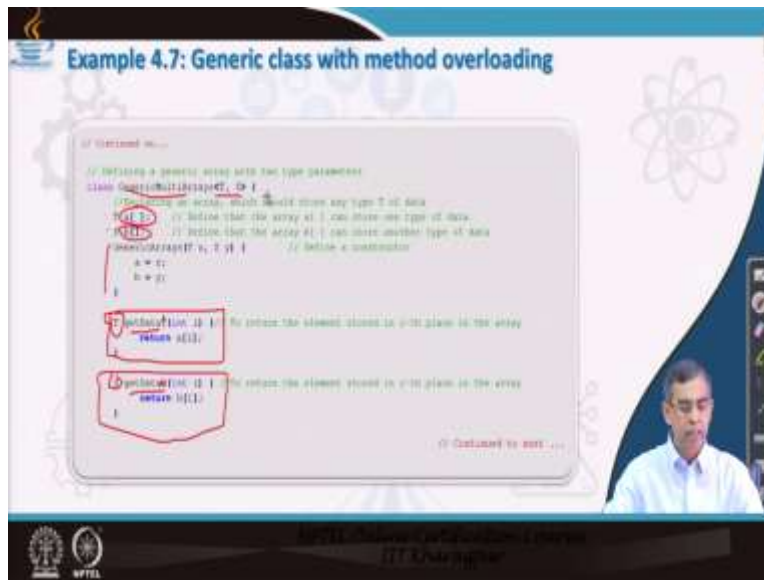
// End of the class Student.
// continued to next ...
```

Now, method overloading as I told you, it also works for you and this example takes short what is called the demo of method overloading. Here what we are doing you see we declare one user-defined class, the name of the class is student which has two fields, the name field and marks, marks composed of, the marks of three subjects as a integer like and this is a constructor to initialize the object of this type.

And this is the print method, print student and here how this print student method will work basically printing the different element name print and then score the total of the score value will be printed here, that is all. So, it will be name and then whatever the values is stored there it will be printed the total. So, this is one simple declaration of a user-defined class. Having this declaration we want to see how the methods in it can we defined in an overloaded fashion.



(Refer Slide Time: 23:10)



And here is an example you can we can repeat it for the two type parameters. So, this basically define a generic class with two template type T and S. So, T here we have defined that it can store an array of objects, b also is an array of objects of type S, here is an array of object of type T, a is an array of object type T , b is an array of object type S.

This type at the time of writing program we do not know, it can be anything. And this is the way of initialization, we pass x and y are the reference and then it will be initialized. GetData here you see as in the previous case it is a overloaded version, two overloaded version of the same program, but here it will basically GetData T okay fine we can return the a and then GetData S return the S, it is exactly not overloaded, the two methods are there.

But if we do not include T here it could work for you also, so no issue int I, int j we have mentioned here because ith element in the array can be return whatever it is there, same. But here the return type is different and then without this T GetData it could work for you also, and overloaded can be done, but here overloaded, not overloaded anything is there but we can do it.

Anyway, so this basically define a method generic class with two overloaded method and I want to declare one more method reversing because we are here dealing with array, there may be one task that reversing the elements, objects in this array as well as in this array, so two different method is required because say it is one type, it is another type. So, two different methods may be there but we will consider the same method like in a generic way.

(Refer Slide Time: 25:24)

The slide displays the following Java code:

```
// Continued on...  
  
// Overloaded method in the generic class  
with printData(T[] T) { // A generic method to print the contents of array T.  
    for(int i = 0; i < T.length(); i++)  
        System.out.println(toString(i) + " "); //Print the contents of T  
        System.out.println(); // Print a new line  
    }  
  
with printData(S[] S) { // The overloaded generic method to print elements of S  
    for(int i = 0; i < S.length(); i++)  
        S[i].println(); // Print the contents of S  
        System.out.println(); // Print a new line  
    }  
  
// Continued on next ...
```

And in the next you see the reverse, so this is another printData and printData these are the two methods, are overloaded method here it is basically print the elements which is stored here of type T and S type T. So, we have declared generic class with two types, array of objects of type T and S and method also we have declared and for the getting any value and then printing any value.

(Refer Slide Time: 26:04)

The slide displays the following Java code:

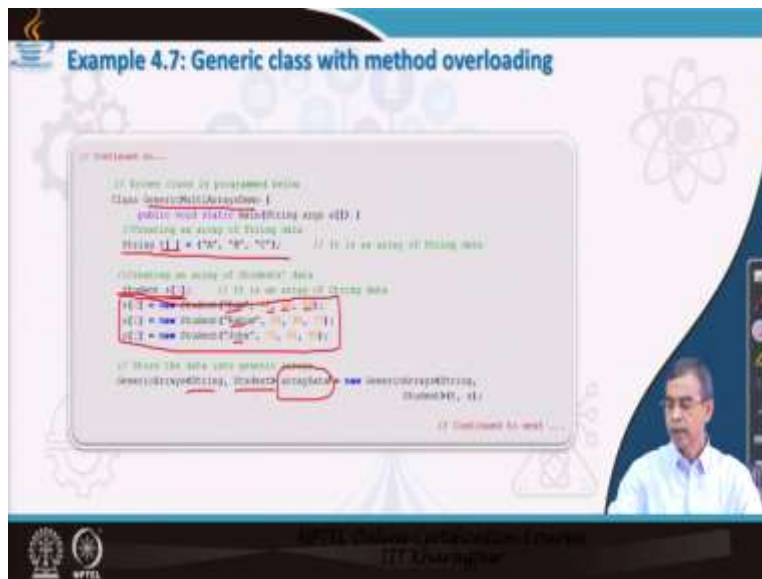
```
// Continued on...  
  
// The additional method  
with reverse(T[] T) { //Generic method to reverse the order of elements in T  
    for (int i = 0, j = T.length()-1; i < j; i++)  
        while (i < j) {  
            temp = T[i];  
            T[i] = T[j];  
            T[j] = temp;  
            i++; j--;  
        }  
    }  
  
with reverse(S[] S) //Generic method to reverse the order of elements in S  
    for (int i = 0, j = S.length()-1; i < j; i++)  
        while (i < j) {  
            temp = S[i];  
            S[i] = S[j];  
            S[j] = temp;  
            i++; j--;  
        }  
    }  
  
// End of the definition of class GenericMultiElement  
  
// Continued on next ...
```

And now another method also we can declare regarding the reversing of elements. So, here this is a reverse array is a method, this is a reverse array the same method name but argument is T

that mean this method will work for a collection of type T, this method will work for the collection of type S.

The logic is basically straightforward to reverse the ordering of the elements. So, this complete the definition of a generic class with multiple types as well as overloaded methods in it. Now, having this declaration now let us see how the main method will work.

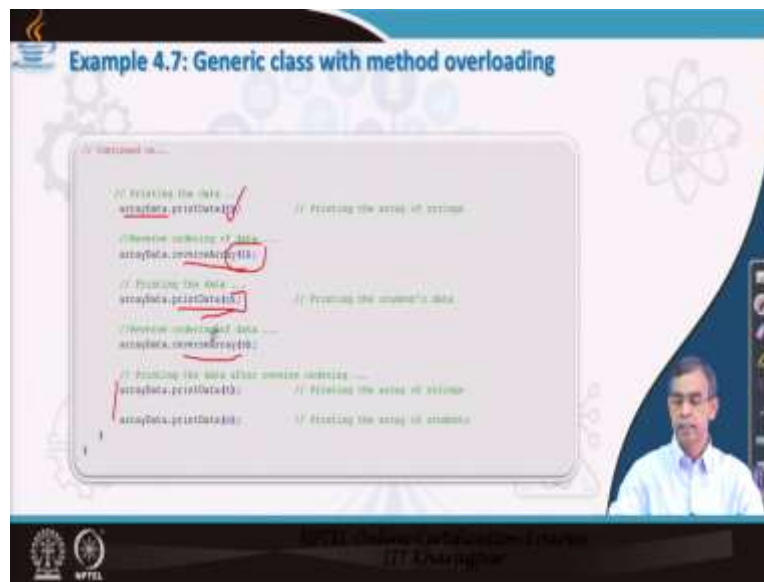
(Refer Slide Time: 26:47)



And here is an example of main method as we see here, this is the demonstration of using the generic class that we have declared in the last slides. In this example first let us create an array of string objects which includes three elements in it, then also we declare three objects of type student stored in an array S, the three objects are initialized with these are the different values, this is the string for name, this is the three integer numbers storing their marks.

Now, we create an object to store two values T and S string and student for example here passing this T and S which we have created here to this one. So, here array data basically includes two arrays, one this one T and S and then store it. Now, so a generic program with two types different string and then student, one is student is user-defined is declared and used here. Now, let us continue this program. The next we see the different method call namely getPrint and then reverse how it work.

(Refer Slide Time: 28:27)



Now, here in this example so array data is basically your correction with two types of arrays and print data T is basically call the print data method for the type T. So, it will print the elements which is stored there in the string. Then array data reverse T, so it will basically reverse the elements which are stored there in array of integers objects.

PrintData S basically in this case S is student so it will print the student data stored in the arrays, reverse array are elements those are stored in the array S will be reversed and after this reversing it will again print the data. So, two type of arrays, one storing integer values, another storing string will be handled here in one single program, not only this any type of other you can run this and taste it of your own so that it works for you. So, this basically signifies how the different types it can works for it.

(Refer Slide Time: 29:50)

**Important Notes**

6. If a class A is declared as generic with type parameter <T>, then object of class can be created any type. This is fine, but it may causes in several situation error during execution.

**Example 4.8**

```
GenericError <T> {  
    // T array // an array of type T  
    // This the constructor a reference to an array of type T.  
    GenericError (T[] t) {  
        array = t;  
    }  
  
    double average () { // Return type double in all cases  
        double sum = 0.0;  
        for (int i = 0; i < array.length; i++)  
            sum += array[i].doubleValue(); // Here is a compiler error!  
        return sum / array.length;  
    }  
}
```

Now, in this context we want to mention one important check which is very important. So, if a class say a is declared as generic with any parameter, any number of parameter no issue, any parameter let it be T. Then object of class can be created of that type, type can be of anything, that is fine, we have learned about it, it works, it is okay. But if you are not careful about using the different methods belongs to those objects type then it may invites some problem for you, it can occur during the time of compilation that means at the compile time error or sometime it is runtime error also.

Now, let us illustrate this concept that what sort of problem that it may arise whenever we can create a generic class. This is an example just you can look at. We declare as a generic class, name of the class is GenericError, type is T specified here. So, according to our understanding, this T can be of any type, means it is integer objects or it can be string, it can be double, it can be student or anything.

So, this is array of objects for this class and this is a constructor, it is as usual. So, these two things are obvious things in generic class discussion, so absolutely no problem in. Now, let us come to here. Now, there is one method we have declared, the name of the method is average. The objective is that whatever the value it is stored here in this array, this method will find the average of this one.

Now, here you note down, this basically is an array of objects, the method average will calculate the average value. So, the simple logic will look like this, is a for loop for each elements into it, it will basically calculate the sum of all elements and then it is basically total number of elements that is stored there array dot length sum divided by this average is there.

Now, here you see array, the ith elements which is stored here as it is stored as an object because T is an object, so from this object representation if we want to calculate this average that means this is basically the simple type should be converted into either integer, or double, or float, so it is basically double value.

Double value usually if we call for any numeric type it will basically return value from object to its primitive type because here we are storing is an object and for the calculation of sum we need the primitive form actually. So, double value it is required to be in invoked there.

(Refer Slide Time: 33:38)

**Important Notes**

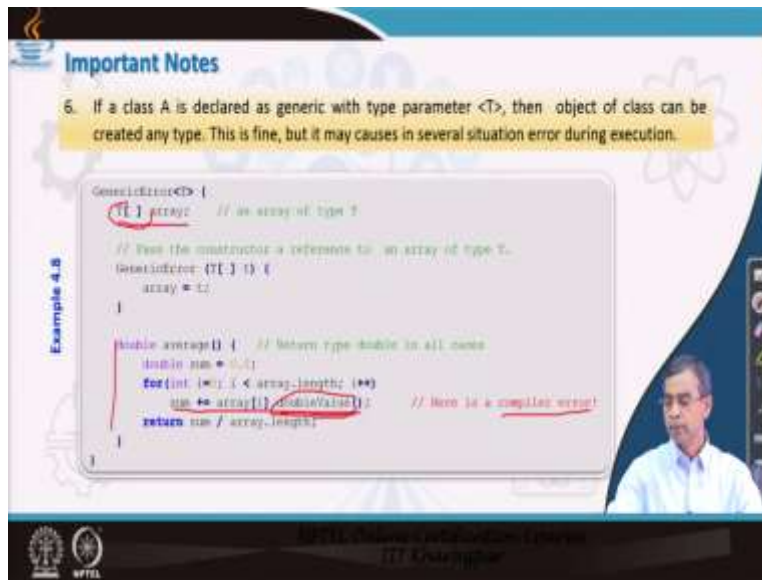
6. If a class A is declared as generic with type parameter <T>, then object of class can be created any type. This is fine, but it may causes in several situation error during execution.

**Example 4.8**

```
GenericError<T> {  
    T[] array; // an array of type T  
  
    // Pass the constructor a reference to an array of type T.  
    GenericError(T[] t) {  
        array = t;  
    }  
  
    double average() { // Return type double in all cases.  
        double sum = 0.0;  
        for(int i=0; i < array.length; i++)  
            sum += array[i]; // Here is a compiler error!  
        return sum / array.length;  
    }  
}
```

That means the simple, that means without this double value if we write it this is not valid, you will get error during compile time it is an error, so that is a double value we have to do it.

(Refer Slide Time: 33:51)



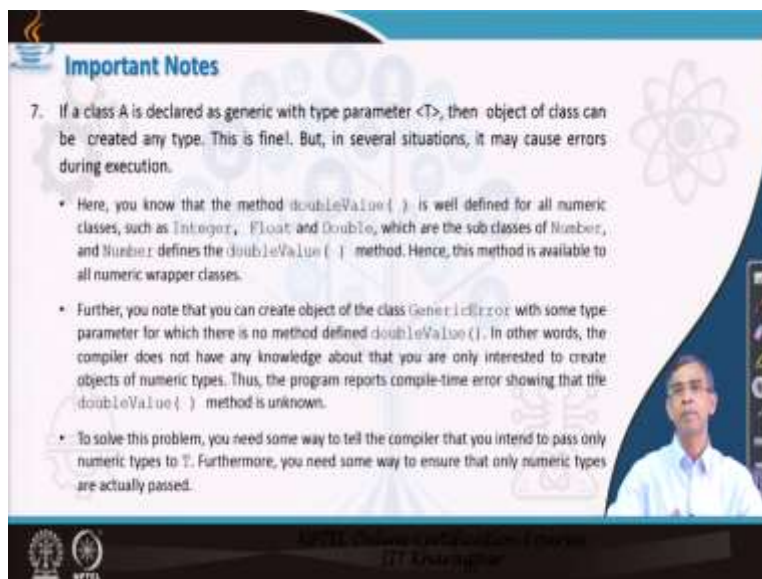
**Important Notes**

6. If a class A is declared as generic with type parameter  $\langle T \rangle$ , then object of class can be created any type. This is fine, but it may causes in several situation error during execution.

**Example 4.8**

```
GenericError<T> {
    T[] array; // an array of type T
    // Pass the constructor a reference to an array of type T.
    GenericError(T[] t) {
        array = t;
    }

    double average() { // Return type double in all cases
        double sum = 0.0;
        for(int i=0; i < array.length; i++)
            sum += array[i].doubleValue(); // Here is a compile error!
        return sum / array.length;
    }
}
```



**Important Notes**

7. If a class A is declared as generic with type parameter  $\langle T \rangle$ , then object of class can be created any type. This is fine. But, in several situations, it may cause errors during execution.

- Here, you know that the method `doubleValue()` is well defined for all numeric classes, such as `Integer`, `Float` and `Double`, which are the sub classes of `Number`, and `Number` defines the `doubleValue()` method. Hence, this method is available to all numeric wrapper classes.
- Further, you note that you can create object of the class `GenericError` with some type parameter for which there is no method defined `doubleValue()`. In other words, the compiler does not have any knowledge about that you are only interested to create objects of numeric types. Thus, the program reports compile-time error showing that the `doubleValue()` method is unknown.
- To solve this problem, you need some way to tell the compiler that you intend to pass only numeric types to `T`. Furthermore, you need some way to ensure that only numeric types are actually passed.

Now, once you just declare it, here is basically you will see if you run this program this statement which is here it will basically gives an error, the error is call compile error, why? This is because this T type assuming that can be any. Now, if it is numeric T, absolutely no problem, this program will work for you, but T not necessary to be only numeric that mean not necessary that it will be integer, it will be double, it will be float, if, what will happen if T is of type string?

That mean array store a set of string objects, then double value method is basically not valid for string, it is valid for only integer, but T can assume anything, it can assume string, it can assume

student also, but for student this double value method is not defined, is undefined, as a consequence it will be an error, compilation error.

So, this is the one problem it is there, while you are writing the program you may not aware about and it can invite double, then compile error. Now, so this is the one problem or rather is a concern about when you want to deal with this one. So, it needs to be checked, definitely there should not be the point of (())(35:28) Java has the facility for dealing with this kind of situation, that situation is called bounding the arguments.

That mean we can define that whatever the template type that we have decided can be limited to only numeric type or it can be only limited to some other type, so we can do that and which is very much essential in many situations.

(Refer Slide Time: 35:56)



In the next lecture we will discuss about this bounding of parameter arguments. You just go through this video lectures to understand the merits, the strength of generic methods and generic class whether is a single type or multiple type whatever it is there. Now, few more advanced topics regarding the bounded argument parameters while defining generic class is also very important issues which will be discussed in the next two video lectures, thank you.