Java Generics Example Tutorial – Generic Method, Class, Interface

APRIL 4, 2018 BY PANKAJ — 53 COMMENTS

Java Genrics is one of the most important feature introduced in Java 5. If you have been working on Java Collections and with version 5 or higher, I am sure that you have used it. **Generics in Java** with collection classes is very easy but it provides a lot more features than just creating the type of collection and we will try to learn features of generics in this article. Understanding generics can become confusing sometimes if we go with jargon words, so I would try to keep it simple and easy to understand.

Java Generics – Generics in Java

We will look into below topics of generics in java.

1. Generics in Java
2. Java Generic Class
3. Java Generic Interface
4. Java Generic Type
5. Java Generic Method
6. Java Generics Bounded Type Parameters
7. Java Generics and Inheritance
8. Java Generic Classes and Subtyping
9. Java Generics Wildcards
    1. Java Generics Upper Bounded Wildcard
    2. Java Generics Unbounded Wildcard
    3. Java Generics Lower bounded Wildcard
10. Subtyping using Generics Wildcard
11. Java Generics Type Erasure

Java Arrays Annotation and Enum
Java Collections Java IO Operations
Java Exception Handling
MultiThreading and Concurrency
Regular Expressions Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial

Generics was added in Java 5 to provide **compile-time type checking** and removing risk of `ClassCastException` that was common while working with collection classes. The whole collection framework was re-written to use generics for type-safety. Let's see how generics help us using collection classes safely.

```
List list = new ArrayList();
list.add("abc");
list.add(new Integer(5)); //OK

for(Object obj : list){
        //type casting leading to ClassCastException at runtime
    String str=(String) obj;
}
```

Above code compiles fine but throws ClassCastException at runtime because we are trying to cast Object in the list to String whereas one of the element is of type Integer. After Java 5, we use collection classes like below.

```
List<String> list1 = new ArrayList<String>(); // java 7 ? List<String> list1 =
new ArrayList<>();
list1.add("abc");
//list1.add(new Integer(5)); //compiler error

for(String str : list1){
     //no type casting needed, avoids ClassCastException
}
```

› Apache Axis 2
› JAX-RS
› Memcached Tutorial

Notice that at the time of list creation, we have specified that the type of elements in the list will be String. So if we try to add any other type of object in the list, the program will throw compile time error. Also notice that in for loop, we don't need type casting of the element in the list, hence removing the ClassCastException at runtime.

Java Generic Class

We can define our own classes with generics type. A generic type is a class or interface that is parameterized over types. We use angle brackets (<>) to specify the type parameter.

To understand the benefit, lets say we have a simple class as:

```
package com.journaldev.generics;

public class GenericsTypeOld {

	private Object t;

	public Object get() {
		return t;
	}

	public void set(Object t) {
		this.t = t;
	}

	public static void main(String args[]){
		GenericsTypeOld type = new GenericsTypeOld();
		type.set("Pankaj");
```

```
and can cause ClassCastException
        }
}
```

Notice that while using this class, we have to use type casting and it can produce ClassCastException at runtime. Now we will use java generic class to rewrite the same class as shown below.

```
package com.journaldev.generics;

public class GenericsType<T> {

        private T t;

        public T get(){
                return this.t;
        }

        public void set(T t1){
                this.t=t1;
        }

        public static void main(String args[]){
                GenericsType<String> type = new GenericsType<>();
                type.set("Pankaj"); //valid

                GenericsType type1 = new GenericsType(); //raw type
                type1.set("Pankaj"); //valid
                type1.set(10); //valid and autoboxing support
        }
```

Notice the use of GenericsType class in the main method. We don't need to do type-casting and we can remove ClassCastException at runtime. If we don't provide the type at the time of creation, compiler will produce a warning that "GenericsType is a raw type. References to generic type GenericsType<T> should be parameterized". When we don't provide type, the type becomes `Object` and hence it's allowing both String and Integer objects but we should always try to avoid this because we will have to use type casting while working on raw type that can produce runtime errors.

Tip: We can use `@SuppressWarnings("rawtypes")` annotation to suppress the compiler warning, check out **java annotations tutorial**.

Also notice that it supports java autoboxing.

Java Generic Interface

Comparable interface is a great example of Generics in interfaces and it's written as:

```
package java.lang;
import java.util.*;

public interface Comparable<T> {
    public int compareTo(T o);
}
```

In similar way, we can create generic interfaces in java. We can also have multiple type parameters as in Map interface. Again we can provide parameterized value to a parameterized type also, for example `new HashMap<String, List<String>>();` is valid.

Java Generic Type Naming convention helps us understanding code easily and having a naming convention is one of the best practices of java programming language. So generics also comes with it's own naming conventions. Usually type parameter names are single, uppercase letters to make it easily distinguishable from java variables. The most commonly used type parameter names are:

- E – Element (used extensively by the Java Collections Framework, for example ArrayList, Set etc.)
- K – Key (Used in Map)
- N – Number
- T – Type
- V – Value (Used in Map)
- S,U,V etc. – 2nd, 3rd, 4th types

Java Generic Method

Sometimes we don't want whole class to be parameterized, in that case we can create java generics method. Since constructor is a special kind of method, we can use generics type in constructors too.

Here is a class showing example of java generic method.

```
package com.journaldev.generics;

public class GenericsMethods {

	//Java Generic Method
	public static <T> boolean isEqual(GenericsType<T> g1, GenericsType<T>
g2){
		return g1.get().equals(g2.get());
	}
```

```
        public static void main(String args[]){
                GenericsType<String> g1 = new GenericsType<>();
                g1.set("Pankaj");

                GenericsType<String> g2 = new GenericsType<>();
                g2.set("Pankaj");

                boolean isEqual = GenericsMethods.<String>isEqual(g1, g2);
                //above statement can be written simply as
                isEqual = GenericsMethods.isEqual(g1, g2);
                //This feature, known as type inference, allows you to invoke
a generic method as an ordinary method, without specifying a type between
angle brackets.
                //Compiler will infer the type that is needed
        }
```

Notice the *isEqual* method signature showing syntax to use generics type in methods. Also notice how to use these methods in our java program. We can specify type while calling these methods or we can invoke them like a normal method. Java compiler is smart enough to determine the type of variable to be used, this facility is called as **type inference**.

Java Generics Bounded Type Parameters

Suppose we want to restrict the type of objects that can be used in the parameterized type, for example in a method that compares two objects and we want to make sure that the accepted objects are Comparables. To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its upper bound, similar like below method.

```
public static <T extends Comparable<T>> int compare(T t1, T t2){
```

```
        }
```

The invocation of these methods is similar to unbounded method except that if we will try to use any class that is not Comparable, it will throw compile time error.

Bounded type parameters can be used with methods as well as classes and interfaces.

Java Generics supports multiple bounds also, i.e <T extends A & B & C>. In this case A can be an interface or class. If A is class then B and C should be interfaces. We can't have more than one class in multiple bounds.

Java Generics and Inheritance

We know that Java inheritance allows us to assign a variable A to another variable B if A is subclass of B. So we might think that any generic type of A can be assigned to generic type of B, but it's not the case. Lets see this with a simple program.

```
package com.journaldev.generics;

public class GenericsInheritance {

        public static void main(String[] args) {
                String str = "abc";
                Object obj = new Object();
                obj=str; // works because String is-a Object, inheritance in
java

                MyClass<String> myClass1 = new MyClass<String>();
                MyClass<Object> myClass2 = new MyClass<Object>();
```

```
is not a MyClass<Object>
                obj = myClass1; // MyClass<T> parent is Object
        }

        public static class MyClass<T>{}

}
```

We are not allowed to assign MyClass<String> variable to MyClass<Object> variable because they are not related, in fact MyClass<T> parent is Object.

Java Generic Classes and Subtyping

We can subtype a generic class or interface by extending or implementing it. The relationship between the type parameters of one class or interface and the type parameters of another are determined by the extends and implements clauses.

For example, ArrayList<E> implements List<E> that extends Collection<E>, so ArrayList<String> is a subtype of List<String> and List<String> is subtype of Collection<String>.

The subtyping relationship is preserved as long as we don't change the type argument, below shows an example of multiple type parameters.

```
interface MyList<E,T> extends List<E>{
}
```

The subtypes of List<String> can be MyList<String,Object>, MyList<String,Integer> and so on.

Java Generics Wildcards

Question mark (?) is the wildcard in generics and represent an unknown type. The wildcard can be used as the type of a parameter, field, or local variable and sometimes as a return type. We can't use wildcards while invoking a generic method or instantiating a generic class. In following sections, we will learn about upper bounded wildcards, lower bounded wildcards, and wildcard capture.

Java Generics Upper Bounded Wildcard

Upper bounded wildcards are used to relax the restriction on the type of variable in a method. Suppose we want to write a method that will return the sum of numbers in the list, so our implementation will be something like this.

```
public static double sum(List<Number> list){
                double sum = 0;
                for(Number n : list){
                        sum += n.doubleValue();
                }
                return sum;
        }
```

Now the problem with above implementation is that it won't work with List of Integers or Doubles because we know that List<Integer> and List<Double> are not related, this is when upper bounded wildcard is helpful. We use generics wildcard with **extends** keyword and the **upper bound** class or interface that will allow us to pass argument of upper bound or it's subclasses types.

The above implementation can be modified like below program.

```
package com.journaldev.generics;

import java.util.ArrayList;
import java.util.List;

public class GenericsWildcards {

	public static void main(String[] args) {
		List<Integer> ints = new ArrayList<>();
		ints.add(3); ints.add(5); ints.add(10);
		double sum = sum(ints);
		System.out.println("Sum of ints="+sum);
	}

	public static double sum(List<? extends Number> list){
		double sum = 0;
		for(Number n : list){
			sum += n.doubleValue();
		}
		return sum;
	}
}
```

It's similar like writing our code in terms of interface, in above method we can use all the methods of upper bound class Number. Note that with upper bounded list, we are not allowed to add any object to the list except null. If we will try to add an element to the list inside the sum method, the program won't compile.

Java Generics Unbounded Wildcard

Sometimes we have a situation where we want our generic method to be working with all types, in this case unbounded wildcard can be used. Its same as using <? extends Object>.

```
public static void printData(List<?> list){
                for(Object obj : list){
                        System.out.print(obj + "::");
                }
        }
```

We can provide List<String> or List<Integer> or any other type of Object list argument to the *printData* method. Similar to upper bound list, we are not allowed to add anything to the list.

Java Generics Lower bounded Wildcard

Suppose we want to add Integers to a list of integers in a method, we can keep the argument type as List<Integer> but it will be tied up with Integers whereas List<Number> and List<Object> can also hold integers, so we can use lower bound wildcard to achieve this. We use generics wildcard (?) with **super** keyword and lower bound class to achieve this.

We can pass lower bound or any super type of lower bound as an argument in this case, java compiler allows to add lower bound object types to the list.

```
public static void addIntegers(List<? super Integer> list){
                list.add(new Integer(50));
        }
```

```
List<? extends Integer> intList = new ArrayList<>();
List<? extends Number>  numList = intList;  // OK. List<? extends Integer> is a
subtype of List<? extends Number>
```

Java Generics Type Erasure

Generics in Java was added to provide type-checking at compile time and it has no use at run time, so java compiler uses **type erasure** feature to remove all the generics type checking code in byte code and insert type-casting if necessary. Type erasure ensures that no new classes are created for parameterized types; consequently, generics incur no runtime overhead.

For example if we have a generic class like below;

```
public class Test<T extends Comparable<T>> {

    private T data;
    private Test<T> next;

    public Test(T d, Test<T> n) {
        this.data = d;
        this.next = n;
    }

    public T getData() { return this.data; }
}
```

The Java compiler replaces the bounded type parameter T with the first bound interface, Comparable, as

```
public class Test {

    private Comparable data;
    private Test next;

    public Node(Comparable d, Test n) {
        this.data = d;
        this.next = n;
    }

    public Comparable getData() { return data; }
}
```

Generics in Java – Further Readings

- Generics doesn't support sub-typing, so `List<Number> numbers = new ArrayList<Integer>();` will not compile, learn why generics doesn't support sub-typing.
- We can't create generic array, so `List<Integer>[] array = new ArrayList<Integer>[10]` will not compile, read why we can't create generic array?.

Thats all for **generics in java**, java generics is a really vast topic and requires a lot of time to understand and use it effectively. This post here is an attempt to provide basic details of generics and how can we use it to extend our program with type-safety.

FILED UNDER: JAVA

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.
Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

« Java FutureTask Example Program Exception Handling in Java »

Comments

ranjit kumar yadav says
MAY 11, 2018 AT 10:56 AM

```
public class GenericsTypeOld {
private Object t;
public Object get() {
return t;
}
public void set(Object t) {
this.t = t;
}
public static void main(String args[]){
GenericsTypeOld type = new GenericsTypeOld();
type.set("Pankaj");
112 //String str = (String) type.get(); //type casting, error prone and can cause ClassCastException
}
```

I think line number 112 is not giving erroe
Reply

Gaurav says
MARCH 31, 2018 AT 12:08 PM
Hi Pankaj,
quite amid follower of your tutorials. Short but well explained topics. However, felt some topics are missing in this article. Like PECS of generics, how shall we use them. or maybe it's me only who is finding it difficult.
Reply

priya says
SEPTEMBER 28, 2017 AT 12:03 AM
Generics forces the java programmer to store specific type of objects. The language is very easy to understand. Thanks for sharing.
Reply

Yash Rathore says
APRIL 16, 2017 AT 10:32 PM
sir plz muje bato ki generics methode me hum 2 object ko jo generics h ko "+" operant se add kyo nahi kr pa rahe.
Reply

Raj Gopal Bhallamudi says

FEBRUARY 14, 2017 AT 10:07 PM

```
interface Parent {
}
interface Child extends Parent {
}
public class Subtyping implements Child {
T tobj;
U uobj;
public Subtyping(T t, U u) {
tobj = t;
uobj = u;
}
public static void main(String[] args) {
Parent obj = new Subtyping(4, “raj”);
Parent obj1 = new Subtyping(4, 40.0);
/*
* The subtypes of Parent can be
* Subtyping , Subtyping and so on.
* but not Subtyping
*
* this statement will give error
* Parent obj2 = new Subtyping( “raj”,4);
*/
System.out.println(obj);
System.out.println(obj1);
}
```

```
@Override
public String toString() {
return " t= " + tobj + " " + uobj;
}
}
```

Reply

Raj Gopal Bhallamudi says

FEBRUARY 14, 2017 AT 10:06 PM

An example for Java Generic Classes and Subtyping .

```
interface Parent {
}
interface Child extends Parent {
}
public class Subtyping implements Child {
T tobj;
U uobj;
public Subtyping(T t, U u) {
tobj = t;
uobj = u;
}
public static void main(String[] args) {
Parent obj = new Subtyping(4, "raj");
Parent obj1 = new Subtyping(4, 40.0);
/*
```

```
* Subtyping , Subtyping and so on.
* but not Subtyping
*
* this statement will give error
* Parent obj2 = new Subtyping( “raj”,4);
*/
System.out.println(obj);
System.out.println(obj1);
}
@Override
public String toString() {
return ” t= ” + tobj + ” ” + uobj;
}
}
```

Reply

swathi says

DECEMBER 16, 2016 AT 7:58 AM

```
public class GenTest
{
T t;
public void setType(T t)
{
this.t=t;
}
```

```
{
return t;
}
public static void main(String[] arg)
{
GenTest gt=new GenTest();
Kamala k=new Kamala();
gt.setType(k);
//Object o=gt.getType();
System.out.println(gt.getType().aboutu());
}
}
//kamala class
public class Kamala
{
public void aboutu()
{
System.out.println("I am kamala ! i am helping class to check generic concept");
}
}
```

Sir above program i am getting error: ' void' type not allowed here

Reply

shubham says

DECEMBER 17, 2016 AT 11:19 PM

1. First of all while using T t in GenTest you need to declare T as type parameter as class GenTest.

3. calling aboutu in sysout – no need to use System.out.println(gt.getType().aboutu());
use gt.getType().aboutu() as method return type is void.
Hope this solves your problem and I recommend you to study the generics post thoroughly.
Reply

Shafali says
MAY 5, 2016 AT 10:11 AM
Thanks for the nice article.
I couldnt understand Raw & Unbounded generics.
What is the difference in below?
Set setOfRawType = new HashSet();
setOfRawType = new HashSet();
Set setOfUnknownType = new HashSet();
setOfUnknownType = new HashSet();
Please explain its very urgent.
TIA!
Reply

GOPINATH M B says
JULY 8, 2016 AT 9:39 PM
1. a raw type (Set) treats the type as if it had no generic type information at all. Note the subtle effect that not only will the type argument T be ignored, but also all other type arguments that methods of that type might have. You can add any value to it and it will always return Object

3. Set is a Set that accepts all objects of some specific, but unknown type and will return objects of that type. Since nothing is known about this type, you can't add anything to that set (except for null) and the only thing that you know about the values it returns is that they are some sub-type of Object.
http://stackoverflow.com/a/7360664
Reply

Shashikumar says
APRIL 25, 2016 AT 10:59 PM
Hi..
Have a doubt. Could you please help?
Three classes are there. A,B & C. Both B & C extends A.
So in generics how can I make the class C not to be included. It should allow only B
Thanks,
Shashi
Reply

Rashmi says
APRIL 14, 2016 AT 4:30 AM
Hi,
I have a question what it exactly means?

sairam says
MARCH 15, 2016 AT 11:44 AM
really helpfull thanks
Reply

Abhinav Joshi says
APRIL 15, 2015 AT 7:45 PM
Sir as I was going through your tutorial. I got stuck at a point
public Object get() {
return t;
}
public void set(Object t) {
this.t = t;
As I have seen in getter(), setter() methord, we write in such a way…
public Object getT() {
return t;
}
public void setT(Object t) {
this.t = t;
What you have done is OK for 1 variable???
Reply

Corey says
MAY 26, 2015 AT 7:30 PM
public Object getT() {
return t;
}
public void setT(Object t) {
this.t = t;
Should be …
public T getT() {
return t;
}
public void setT(T t) {
this.t = t;
This allows any type. If you do it how you had it originally, you're just making 2 Object type getters/setters with different names.
Reply

rohit bhardwaj says
NOVEMBER 3, 2015 AT 10:22 PM
this program gets error bro
Reply

Satpal Singh says
OCTOBER 2, 2014 AT 6:07 AM
How converting int to String is autoboxing in the Class generic example given above.
type1.set(10); //valid and autoboxing support
Reply

anonymous says
SEPTEMBER 29, 2014 AT 10:49 PM
good explanation
Reply

Pramod Bablad says
SEPTEMBER 13, 2014 AT 5:20 AM
Very good explanation… Thanks
Reply

sony says
SEPTEMBER 4, 2014 AT 4:09 AM
great explanation!!!
Reply

Abbas says
AUGUST 1, 2014 AT 5:16 PM
I was looking for this:
Thanks 🙂
Reply

Manish Verma says
AUGUST 1, 2014 AT 12:26 PM
Excellent article. I am wondering for simple cases, is there a reason to use Wildcard instead of generics? e.g. The printData method in the "Generics Unbounded Wildcard" could have been written like that?
public static void printData(List list) {
for(T obj : list) {
System.out.print(obj + "::");
}
}
And we don't have the limitations of wildcard either (add method not allowed). So, is there any benefit of using wildcard in such cases?
Reply

SUDIP GHOSH says
JULY 21, 2014 AT 4:45 AM
Hi Pankaj

I have a query regarding generics.
A sample progam:

```
package main.test;
import java.util.ArrayList;
import java.util.List;
public class MainClass {
/**
* @param args
*/
public static void main(String[] args) {
List list = new ArrayList();
list.add(8);
list.add(7);
printList1(list);
printList2(list);
printList3(list);
printList4(list);
}
/**
* Using Generics Bounded Type Parameters
* @param list
*/
public static void printList1(List list){
System.out.println(list);
}
/**
* Using Generics Upper Bounded Wildcard
```

```
*/
public static void printList2(List list){
System.out.println(list);
}
/**
* Using Simple Generics Method
* @param list
*/
public static void printList3(List list){
System.out.println(list);
}
/**
* Using Generics Unbounded Wildcard
* @param list
*/
public static void printList4(List list){
System.out.println(list);
}
}
```

There are 4 methods printing a list:
printList1: Using Generics Bounded Type Parameters
printList2: Using Generics Upper Bounded Wildcard
printList3: Using Simple Generics Method
printList4: Using Generics Unbounded Wildcard
I have two questions:
1) When to use Generics Upper Bounded Wildcard? Because the same thing can be done using Generics Bounded Type Parameters?(methods: printList1 and printList2)

2) Similar Question regarding method: printList3 and printList4.When to use the Generics Unbounded Wildcard?Because the same thing can be done using Using Simple Generics Method.

Reply

SUDIP GHOSH says

JULY 21, 2014 AT 5:09 AM

The generics syntax(angel bracket) are not coming while I'm pasting it. 😕

Reply

Pankaj says

JULY 21, 2014 AT 6:58 AM

Generic Upper/Lower Bounded Wildcard and Bounded Type parameters are similar, however bounded type parameters help us in writing generic algorithms, for example in above code, we are using bounded type parameters with Comparable.

Reply

Amishi Shah says

JUNE 9, 2014 AT 8:15 PM

Could you please explain what is that which can be done in a generic class but not in a normal java class without generics?

Reply

Finishing School says
JUNE 14, 2014 AT 11:16 AM
Firstly u need to understand that GENERIC in java represents a General Model(be it a method or a class).
Secondly, by creating generic class u r creating a General Class which can operate on any type of Object.
Thirdly, without using generics u need to create separate class for supporting various types of objects.
thus use of generic class helps u in reusability of code i.e less LOC & Compile time Type Checking.
thanks
Finishing School
Reply

Amishi Shah says
JUNE 18, 2014 AT 3:55 PM
Thanks a lot.
Reply

viswam says
MAY 28, 2014 AT 5:02 AM
Nice explanation, easy to understand

Thanks for the post.

Reply

laukendra says

MAY 26, 2014 AT 10:51 PM

Awesome man 🙂

Reply

Anup says

APRIL 25, 2014 AT 6:39 AM

Awesome work!

Really informative

Reply

Pablo says

APRIL 26, 2014 AT 10:34 AM

Thank you very much, really well explained.

Reply

Mayur Bote says
APRIL 25, 2014 AT 2:36 AM
How can I use binary operators in generics ???
Reply

smokes says
APRIL 22, 2014 AT 10:06 AM
Tnx man.
really great explanation for beginners.
Much appreciated.
Reply

Ashish says
APRIL 18, 2014 AT 12:16 AM
Dear sir i have a query regarding generics plz solve this…
I have a generic super class Employee and this super class have two sub class Admin and Finance…so tell me can i use Admin data in Finance sub class and Finance data in Admin sub class.
Reply

Pankaj says
APRIL 18, 2014 AT 6:49 AM

This is not generics, its inheritance. Think of it like Integer and String class, both have super class as Object. Can you assign Integer to String and vice versa, NO.

Reply

Dev says

FEBRUARY 4, 2014 AT 11:56 AM

Hi

Respected Sir

My expectation is growing high after getting answer from you and solving my previous problem.

I face another problem while using java generic in my program.

The problem is i have to GROUP BY a ganeric list based on one or two field.

Suppose i have a POJO class named StationInformation with CODE,STATION,NAME,ADDRESS

i have created a list ArrayList StationInformationList= new ArrayList

and populate this list with station data.

After some operation with this list object i have to GROUP BY this list based on CODE field and after that CODE and STATION field. This GROUP BY should be same as GROUP BY in database.

There is no way to go database from here because data is coming from api.

This is very essential to me and i have to resolve it as soon as possible.

I hope i'll get another feedback from you with solution my problem.

This GROUP BY is vary urgent and please give me a solution.

I am waiting for your answer/reply.

Please please sir give me a solution.

If possible please give me code example.

Reply

Siddu says
FEBRUARY 4, 2014 AT 11:14 AM
What are the drawbacks of Generics?
Reply

Dev says
JANUARY 13, 2014 AT 6:08 AM
Thanks a lot. I made this change in my code and this is working correctly.
Reply

Mohsen says
JANUARY 12, 2014 AT 8:26 AM
Thanks for this great article … it was very helpful for me
Reply

Dev says
JANUARY 11, 2014 AT 10:26 AM
I want to create xlsx file from a ArrayList.

I have three (3) list
1. ArrayList CompanyBeanList (CompanyBean has 4 fileds with getter – setter)
2.ArrayList EmployeeBeanList (EmployeeBean has 10 fileds with getter – setter)
3.ArrayList ClientBeanList (ClientBeanBean has 8 fileds with getter – setter)
I want to call exportToExcel method as exportToExcel("D:\Back Up\PROJECT\",CompanyBeanList ,"CompanyBean")
and exportToExcel("D:\Back Up\PROJECT\",EmployeeBeanList ,"EmployeeBean")
and exportToExcel("D:\Back Up\PROJECT\",ClientBeanList ,"ClientBean")
exportToExcel is a method which create an .xlsx file in the specified path with data from the beanList (2 nd parameter)
But problem is i can not pass these 3 different type of list in exportToExcel.
Error says The method exportToExcel(String, ArrayList, String) in the type Utility is not applicable for the arguments (String, List, String) (same for other two class)
I do not want to write BeanClass name in parameter. it should be parametrized.

```
public static void exportToExcel(String outputPath , ArrayList beanClassName,String sheetName)
{
try{
Map data = new TreeMap();
XSSFWorkbook workbook = new XSSFWorkbook();
XSSFSheet sheet = workbook.createSheet(sheetName);
Object[] colName =new Object[20];
if(beanClassName.size()>0)
{
Field[] field=beanClassName.get(0).getClass().getDeclaredFields();
for(int i=0;i<field.length;i++)
{
colName[i]=field[i].getName().toUpperCase();
```

```
}
data.put("1", colName);
for(int i=0;i<beanClassName.size();i++)
{
Object[] colValue =new Object[20];
Field[] field1=beanClassName.get(0).getClass().getDeclaredFields();
for(int f=0;i<field1.length;i++)
{
Method method=null;
try {
method=beanClassName.get(i).getClass().getMethod("get"+field1[f].getName().toUpperCase(), null);
}catch (IllegalArgumentException e) {
log.error("User: "+getUserName()+" : "+"exportToExcel..1.."+e.getMessage());
e.printStackTrace();
} catch (Exception e) {
log.error("User: "+getUserName()+" : "+"exportToExcel..2...."+e.getMessage());
e.printStackTrace();
}
colValue[f]=method.invoke(beanClassName.get(i), null);
}
data.put((i+2)+"", colValue);
}
Set keyset = data.keySet();
int rownum = 0;
for (String key : keyset)
{
Row row = sheet.createRow(rownum++);
```

```
int cellnum = 0;
for (Object obj : objArr)
{
Cell cell = row.createCell(cellnum++);
if(obj instanceof String)
cell.setCellValue((String)obj);
else if(obj instanceof Integer)
cell.setCellValue((Integer)obj);
else if(obj instanceof Double)
cell.setCellValue((Double)obj);
else if(obj instanceof Float)
cell.setCellValue((Float)obj);
else if(obj instanceof Number)
cell.setCellValue((Double)obj);
}
}
try
{
//Write the workbook in file system
FileOutputStream out = new FileOutputStream(new File(outputPath+File.separator+"OrderCounts.xlsx"));
workbook.write(out);
out.close();
}
catch (Exception e)
{
e.printStackTrace();
}
```

```
}
else
{
return;
}
}catch (Exception e) {
log.error("User: "+getUserName()+" : "+"exportToExcel....."+e.getMessage());
e.printStackTrace();
}
}
```

I only want to call
exportToExcel("D:\Back Up\PROJECT\",CompanyBeanList ,"CompanyBean")
and exportToExcel("D:\Back Up\PROJECT\",EmployeeBeanList ,"EmployeeBean")
and exportToExcel("D:\Back Up\PROJECT\",ClientBeanList ,"ClientBean")
and it will create .xlsx file in the path. That is my main aim.
Is it possible.??
Which modification is necessary in exportToExcel method. ???
If solution is known to you please write / modify exportToExcel method as you wish and reply me over mail or post in this site.
It is very urgent to me. I am waiting for your reply.
I hope i'll get a suitable solution from you.I believe on you.
Reply

Pankaj says
JANUARY 12, 2014 AT 2:36 AM
its simple, the error is coming because your method argument is expecting ArrayList but you might

All you need to do is change the method argument type to List and you should be good to go.

Reply

RANJITH KUMAR says

NOVEMBER 26, 2013 AT 1:31 PM

i want usecase digram for generics vs collection, its for my job,please give quickly

Reply

varun says

NOVEMBER 12, 2013 AT 8:54 AM

List str=null;

str.add("varun");

str.add(new Integer(10), str);

im getting compile time error..plz give me solution .how to add string and integer both in arrylist ???

Reply

Pankaj says

NOVEMBER 15, 2013 AT 8:04 AM

Create list of Object and then you can add any type of Object. In above code snippet str is not initialized.

Reply

sriram says
OCTOBER 26, 2013 AT 7:44 AM
hi, I want to define my arraylist which allows to insert only class A,B,C .How to achieve in genrics
Reply

Pankaj says
OCTOBER 26, 2013 AT 4:24 PM
The classes should have something in common, if nothing specified then Object is the common parent class that we can use to define the type of list.
Reply

Sarah says
AUGUST 3, 2013 AT 12:01 PM
hi! i want to devlop my future in java ,i want to learn it i am beginner no from whrere do i start please help me!!!
Reply

Pankaj says

Start with core java and then move on to J2EE technologies.

Reply

Avinash says

JULY 27, 2013 AT 4:40 PM

Hi

Thanks for the very informative article..

However could you please explain the difference between:

public static void printData(List list){

for(Object obj : list){

System.out.print(obj + "::");

}

}

And

public static void printData(List list){

for(Object obj : list){

System.out.print(obj + "::");

}

}

Reply

Pankaj says

JULY 28, 2013 AT 4:50 AM

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT